niva

User Guide · Reference · Cookbook · Algorithm Appendix (v0.62.4)

2026-07-08

Contents

About niva **4**
The idea . . . 4
Goals . . . 4
Status & open questions . . . 4

niva User Guide **6**
1. What niva is . . . 6
Dependencies — required & optional . . . 6
2. Inside QGIS — the plugin . . . 7
3. Standalone — CLI . . . 8
4. Standalone — Python API . . . 9
5. Configuration & environment . . . 10
6. Scratch space & large rasters . . . 10
7. The run journal . . . 11
8. Export to / import from PyQGIS . . . 11
9. Databases & credentials . . . 11
10. Troubleshooting . . . 11

niva Reference **13**
1. The flow model . . . 14
2. Syntax . . . 14
3. Value types & units . . . 15
4. Built-in verbs . . . 16
5. Alias verbs (the registry) . . . 23
6. The `run` escape hatch & `describe` . . . 33
7. Database connections (`@conn`) . . . 33
8. Environment variables . . . 34
9. Command-line interface . . . 35
10. Python API . . . 36

niva Cookbook **37**
A. First steps — one transform . . . 37
B. Multi-step chains . . . 38
C. Attributes, fields & joins . . . 38
D. Overlay & spatial selection . . . 39
E. Batch processing with each . . . 39
F. Raster & terrain . . . 40
G. Spatial SQL in SpatiaLite . . . 40
H. Spatial SQL in PostGIS . . . 41
I. Projects & templates . . . 41
K. Reaching every provider with `run` . . . 42
L. LiDAR & point clouds . . . 44
M. Complex, value-added pipelines . . . 45

N. Data discovery & inventory — `find` and `catalog` . . . 47
Capstone — full pipelines . . . 48

Template projects **49**
How instantiation works . . . 49
What a template can carry . . . 50
Caveats when authoring . . . 50
Authoring a template, end to end . . . 51
The bundled `example` template . . . 51

FAQ **52**
What software libraries are needed to use niva? . . . 52
Do I need to know Python or PyQGIS? . . . 52
How do I run niva — in QGIS or from the command line? . . . 52
I typed `niva` in a terminal and nothing happened (or ”command not found”) . . . 53
Is niva on PyPI? How do I install it? . . . 53
Does `uv tool install qgis-niva` connect to QGIS? Do I need the plugin? . . . 53
Which QGIS versions are supported? . . . 56
A raster job fails with ”disk quota exceeded” or ”No space left” even though I have free disk — why? . . . 56
How do I read and write a database? . . . 56
How do I see what layers/tables are in a file or database? . . . 57
How do I find my database connection names (the `@conn` values)? . . . 57
Standalone niva shows different connections than my QGIS — why? . . . 57
Can niva use a connection from another QGIS profile? . . . 57
Can I use an algorithm that doesn't have a niva verb? . . . 58
How do I turn a niva flow into a normal PyQGIS script? . . . 58
Does niva send my data anywhere, or need an internet connection? . . . 58
How do I get the whole guide as one document? . . . 58

Editor / IDE integration **59**
Language server — `niva lsp` (real completion, diagnostics, hover) . . . 59
One command (Linux / macOS) . . . 60
Coverage at a glance . . . 60
VS Code / VSCodium . . . 62
Any TextMate-compatible editor (Sublime, Zed, JetBrains, bat) . . . 62
Vim / Neovim . . . 62
nano . . . 62
Mousepad / gedit / GNOME Builder / Pluma / xed (GtkSourceView) . . . 62
Kate / KWrite / KDevelop (KSyntaxHighlighting) . . . 63
Notepad++ (Windows) . . . 63
GitHub . . . 63
Emacs / Helix — not yet . . . 63
Keeping the definitions in sync . . . 63

PDAL & LAStools in QGIS 4 on Ubuntu (native install) **64**
PDAL (native point cloud provider) . . . 64
LAStools (rapidlasso) — install & configure . . . 68
Orfeo ToolBox & SAGA (extra providers) . . . 69
Troubleshooting on this host (Ubuntu 26.04 — QGIS 4.0.3) . . . 70
The native-CLI harness — `pdalcli:` and `saga:` . . . 71

Point-cloud (PDAL) backend — setup for Windows, macOS & Linux **74**
The dependency, stated plainly . . . 74
Windows . . . 75
macOS . . . 75

Linux . . . 75
Verify it works (any platform) . . . 76
Gotchas . . . 76
Appendix — building from source (not recommended) . . . 77

Publishing a QGIS plugin — a reusable playbook **78**
1. `metadata.txt` — the contract with the repository . . . 78
2. QGIS 4 / Qt6 readiness — the part that trips people up . . . 78
3. License . . . 79
4. Packaging the zip (niva's `plugin/build_plugin.sh` pattern) . . . 79
5. Publishing to plugins.qgis.org . . . 80
6. Security scanning — clear it BEFORE you upload . . . 80
7. Pre-publish checklist . . . 81
niva-specific extras . . . 81

Quick start **83**

QGIS algorithm appendix **86**

`native:` algorithms **87**

`gdal:` algorithms **260**

`qgis:` algorithms **292**

`pdal:` algorithms **312**

`3d:` algorithms **325**

`grass:` algorithms **326**

`otb:` algorithms **520**

About niva

The idea

Automating geoprocessing in QGIS today means writing PyQGIS — initialize a `QgsApplication`, memorize algorithm IDs like `native:buffer`, build `ALL_CAPS` parameter dicts, juggle `TEMPORARY_OUTPUT`, and thread each tool's output into the next. That is a *programming* task, which puts everyday automation out of reach for the GUI-first analysts who most need it — and is tedious even for those who can.

niva's answer is a **short, readable text grammar** — a whole pipeline on one line that a non-programmer can write *and* read:

```
load roads.gpkg | buffer 100 dissolve | clip city.gpkg | save roads_local.gpkg
```

…running on QGIS's own Processing algorithms underneath.

Goals

- **Readable over powerful.** Favor a grammar a non-programmer can pick up — not a new programming language.
- **Thin over clever.** A wrapper over PyQGIS / QGIS Processing (friendly alias names → `native:*` algorithms), not a reimplementation of GIS.
- **Meet people where they are.** A text-pipeline grammar, a Python API (`niva.flow(...)`), a CLI, and a QGIS plugin dock — pick the surface that fits.
- **One backend first.** In-process PyQGIS for `v1`; headless `qgis_process` later for batch.
- **Clean-room.** The grammar is derived from QGIS Processing's own model and plain readability — not from any proprietary GIS scripting language.
- **Provenance as a byproduct.** Every `save` records lineage; `assess` profiles data quality. You get a documented, reproducible pipeline without extra effort.

Status & open questions

niva **runs** (v0.31.0): the full path — grammar → registry/binder → engine → PyQGIS backend — executes real geoprocessing, validated against real GIS data on QGIS 4.0.3. The surface is now broad: **45 alias verbs** plus **14 built-in verbs** (`load`, `save`, `sql`, `run`, `metadata`, `assess`, `catalog`, `show`, `info`, `project`, `style`, `split`, `notify`, `email`), with **every** QGIS algorithm reachable via `run`, PostGIS/SpatiaLite read·write·analyse, project & template files, a QGIS plugin, and a standalone CLI/API. It is covered by a QGIS-free unit/integration tier (**330+** tests) plus a live-QGIS tier.

The design is worked out in `planning/` (PRD, architecture, grammar spec, security model, the `Oscar_the_Grouch.md` failure register, and the traceability matrix of every verb ↔ algorithm).

Deliberately still-open questions:

- library vs CLI vs plugin as the *primary* surface for the target user;

- how far the friendly-error layer should go in translating cryptic QGIS/GDAL messages;
- whether the premise holds — that GUI-first analysts will adopt a text grammar at all (the risk `Oscar_the_Grouch.md` calls out first).

niva User Guide

How to install and run niva — **inside QGIS** (the plugin) and **standalone** (the CLI and Python API) — plus configuration, logging, and troubleshooting. For the full verb/option catalogue see the Reference; for worked examples see the Cookbook.

- 1. What niva is
- 2. Inside QGIS — the plugin
- 3. Standalone — CLI
- 4. Standalone — Python API
- 5. Configuration & environment
- 6. Scratch space & large rasters
- 7. The run journal
- 8. Export to / import from PyQGIS
- 9. Databases & credentials
- 10. Troubleshooting

---

1. What niva is

niva is a small text language for QGIS geoprocessing. You write a **flow** — a chain of verbs joined by `|` — and niva resolves each verb to a QGIS Processing algorithm and runs it:

```
load roads.gpkg | reproject EPSG:2262 | buffer 100m dissolve | save roads_buf.gpkg
```

It runs the same two ways: as a **QGIS plugin** (a dock panel, using the QGIS session's own Processing engine) and **standalone** (a `niva` CLI / Python API that boots QGIS headless). The language and results are identical; pick whichever fits the job. niva is pure Python with zero runtime dependencies beyond QGIS itself.

Dependencies — required & optional

Required — that's the whole list:

Requirement	Notes
QGIS 3.22+ (Qt5) or 4.x (Qt6)	niva runs *inside* QGIS's own Python and Processing engine. Install QGIS from qgis.org/download (Windows standalone/OSGeo4W, macOS, or Linux packages).
Python	Provided by QGIS — you do **not** install a separate Python. niva itself is pure Python, vendored in the plugin zip (no `pip` step for the plugin).

Everything below is **optional** — install a piece only when you use the feature that needs it. The bundled providers (`native:`, `gdal:`, `qgis:`, and usually `grass:` and `pdal:`) come with QGIS and need nothing extra; `run <id>` reaches all of them.

Feature	Needs	How to get it
`figure` / `map` (rendering)	Nothing extra — QGIS renders both. `basemap=osm` needs network access	built in
`run grass:*` (terrain/hydrology/classification)	GRASS	Bundled with most QGIS installs; nothing to do
`run pdal:*` (QGIS point-cloud provider)	`pdal_wrench` + `QGIS_WRENCH_EXECUTABLE`; raw `.las`/`.laz` may need a COPC index	Usually built into QGIS on Windows/macOS; on Linux install it yourself — full per-platform steps in the **point-cloud setup guide**
`run pdalcli:*` (PDAL on raw LAS/LAZ/COPC — DTM/DSM/classify/merge/clip)	`pdal_wrench` + `QGIS_WRENCH_EXECUTABLE`	`conda install -c conda-forge pdal pdal_wrench`, then set the env var — see the **point-cloud setup guide** (Windows/macOS/Linux)
`run saga:*` (SAGA CLI harness)	`saga_cmd`	Linux `apt install saga` · macOS `brew install saga-gis` · conda `saga` · set `NIVA_SAGA_CMD` if not on PATH
`run otb:*` (Orfeo ToolBox)	OTB binaries + the OTB provider plugin	orfeo-toolbox.org/download + set OTB folders (see guide)
`run lastools:*` (LAStools)	LAStools binaries (+ `libjpeg62` on Linux; a licence for production)	rapidlasso — optional; PDAL covers most of it
`notify` (ntfy push)	Network + an ntfy topic	Set `NIVA_NTFY_TOPIC` (and optionally `NIVA_NTFY_SERVER` / `NIVA_NTFY_TOKEN`)
`email` (SMTP)	An SMTP account	Set the `NIVA_SMTP_*` env vars (see §5)
PostGIS `@conn`	A saved QGIS PostgreSQL connection	Configure it in QGIS's Data Source Manager — niva only ever sees the connection *name*
Editor highlighting/snippets	Your editor	One command: `bash .vscode/niva/install.sh` — see the editor guide

Nothing here is needed to write, dry-run, `describe`, or `search` flows — those work with niva alone. For LiDAR/point-cloud work start with the **point-cloud (PDAL) setup guide** (plain per-platform install for Windows/macOS/Linux); the deeper **PDAL/LAStools/OTB/SAGA guide** covers every algorithm and the other raster providers, and **editor integration** covers IDE support.

---

2. Inside QGIS — the plugin

Install

1. Build the plugin zip from the repo: `bash plugin/build_plugin.sh` → produces `plugin/niva_qgis.zip`. (Each tagged release also attaches `niva_qgis.zip` — download it from the Releases page.)
2. In QGIS: **Plugins ▸ Manage and Install Plugins ▸ Install from ZIP**, choose `niva_qgis.zip`, install.
3. If it doesn't appear, enable **Show also experimental plugins** in the Plugin Manager settings (niva is marked experimental). It works on QGIS 3.22+ (Qt5) and QGIS 4 (Qt6), with no pip step — the package is vendored inside the zip.
4. Click the **niva** toolbar button (or the `niva` menu) to toggle the dock.

The dock — three tabs

Flow tab — write and run flows.

- Type one or more flows in the editor (one flow per line; # comments). **Open…** loads a `.niva` file so `call` and relative paths resolve from it.

- **Run** executes for real in a background task (the UI stays responsive); progress streams to the output panel and the final output layer is added to the map.
- **Dry-run** validates without touching data — it parses, resolves each verb to its QGIS algorithm and parameters, and lists what it *would* run.
- **Stop** cancels a running flow; **Clear output** clears the panel.

Convert tab — bridge to PyQGIS (see §8). Export the current flow to a standalone PyQGIS script, save it, or import a niva-shaped script back into a flow.

Setup tab — configure the session:

- **Raster scratch** — the folder for large raster intermediates (sets `NIVA_TMPDIR`). Defaults to a real-disk folder under your QGIS profile. Point it at a roomy drive for big raster jobs (see §6).
- **Run log** — log each run to one file per QGIS session, in a folder you choose (§7).
- **Email & notifications** — fields for the `notify`/`email` environment (ntfy topic/server, SMTP host/user/from, and the "Notify on errors / warnings" toggles), with **Send test notification / email** buttons. Non-secret values persist; secrets (passwords, tokens) are session-only unless you save them to the QGIS encrypted store.
- **Environment report** — niva version, install path, the verbs and algorithms available, your saved database connections, and QGIS/GDAL/PROJ versions.

---

3. Standalone — CLI

> **A bare `niva` in a fresh shell usually does nothing — "command not found".** That's expected: there's no `niva` command until you put one on your PATH, *and* it must run on **QGIS's own Python** (niva imports `qgis.core` + the Processing providers — the system `python3` typically can't). Fix it one of two ways:

Make `niva` runnable

Option A — no install (run it as a module). Point QGIS's Python at the niva package and run the CLI module. **Two placeholders below must be replaced for your machine — copying the line verbatim will fail:**

- **`/path/to/niva`** — the directory holding the `niva/` package (your repo root).
- **`/path/to/qgis/python3`** — the **full path to QGIS's Python interpreter**, *not* a bare `python3`. A bare `python3` frequently resolves to a Homebrew / pyenv / conda Python that cannot import QGIS, giving `No module named 'niva'` or a PyQGIS `ImportError`. On Linux with a distro QGIS this is typically `/usr/bin/python3.NN` (match QGIS's exact minor version).

```
# Substitute both placeholders:
PYTHONPATH=/path/to/niva:/usr/share/qgis/python:/usr/lib/python3/dist-packages \
  QT_QPA_PLATFORM=offscreen /path/to/qgis/python3 -m niva.cli.main describe buffer
```

Wrap it in a shell alias so `niva …` just works (put it in your `~/.bashrc`):

```
# Template:
alias niva='PYTHONPATH=/path/to/niva:/usr/share/qgis/python:/usr/lib/python3/dist-packages
 ↪  QT_QPA_PLATFORM=offscreen /path/to/qgis/python3 -m niva.cli.main'

# Concrete example (Linux, distro QGIS 4.x, repo at ~/Github/niva):
alias niva='PYTHONPATH=/home/jcz/Github/niva:/usr/share/qgis/python:/usr/lib/python3/dist-
 ↪  packages QT_QPA_PLATFORM=offscreen /usr/bin/python3.14 -m niva.cli.main'
```

Not sure of QGIS's Python or paths? In the QGIS **Python Console** run: `import sys, qgis; print(sys.executable); print([p for p in sys.path if 'qgis' in p.lower()])` — use that `sys.executable` as the interpreter. Then run `niva info`: if it lists your database connections, the alias is correctly wired to your real QGIS profile.

Option B — install the `niva` command. Install niva *into QGIS's Python* so it creates a `niva` console script, then make sure that interpreter's `bin/Scripts` is on your PATH:

```
<qgis-python> -m pip install git+https://github.com/johnzastrow/niva.git
```

On **Windows** run this from the **OSGeo4W shell**; on **macOS** use `QGIS.app`'s bundled Python — both already have the bindings importable (no `PYTHONPATH` needed). On Linux, the QGIS bindings may still need to be on `PYTHONPATH`, so Option A's alias is often simplest.

> `--dry-run` and `--explain` work on **any** Python (they don't import QGIS), so `python3 -m niva.cli.main "…" --explain` is a quick way to sanity-check your install.

Commands

```
niva run <file.niva> [--dry-run | --explain] [--log <base>]
niva <file.niva>     [--dry-run | --explain] [--log <base>]   # run is optional for a .niva
 ↪  file
niva "<flow>"        [--dry-run | --explain] [--log <base>]
niva describe <verb-or-algorithm-id> [to=<file>]
niva export <file.niva> [-o <file.py>]
niva import <file.py>   [-o <file.niva>]
```

- `niva run flow.niva` / `niva flow.niva` / `niva "load … | save …"` — execute a file (the `run` is optional for a `.niva`/`.flow` file) or an inline flow.
- `--dry-run` — print the plan and validate it with no QGIS, no data touched.
- `--explain` — print the resolved algorithm and parameters for each stage.
- `--log <base>` — also write the journal (`<base>.jsonl` + `<base>.log`).
- `niva describe buffer` / `niva describe gdal:warpreproject [to=out.md]` — introspect a verb or algorithm (with a runnable example).
- `niva "search <keyword>"` / `niva "docs <keyword> to=guide.md"` — fuzzy-find functions across the verbs and the live QGIS catalog, or build a saved mini-guide. (Flow verbs, so they also run in the plugin dock; both need QGIS.)

Exit codes: `0` ok · 1 runtime error · 2 usage/parse error · 3 QGIS not importable.

Running headless (Linux example)

```
export QT_QPA_PLATFORM=offscreen          # no display needed
export QGIS_PREFIX_PATH=/usr              # your QGIS install prefix
export PYTHONPATH=/usr/share/qgis/python:/usr/lib/python3/dist-packages
export NIVA_TMPDIR=$HOME/niva_scratch     # roomy disk for raster scratch (§6)

niva run myflow.niva
# or, if the console script isn't on PATH:
python3 -m niva.cli.main run myflow.niva
```

On Windows/macOS use the Python that ships with QGIS (the OSGeo4W shell on Windows, or `QGIS.app`'s bundled Python on macOS); the same env vars apply with platform-appropriate paths. `--dry-run` and `--explain` work on **any** Python (they don't import QGIS).

---

4. Standalone — Python API

```
import niva

# run an inline flow (needs QGIS on PYTHONPATH)
layer = niva.flow('load "data.gpkg|layername=roads" | buffer 100m dissolve | save
 ↪  out.gpkg')
```

```
print(layer.ref)          # '/abs/path/out.gpkg'

# run a file, with live progress
niva.run_file("myflow.niva")
niva.flow(open("myflow.niva").read(), file="myflow.niva",
          progress=lambda msg: print(msg))

# introspect a verb (no QGIS needed)
print(niva.describe("buffer"))
```

`flow(text, *, backend=None, file=None, log=None, log_append=False, progress=None, cancel=None)` returns the final layer handle (or None for a terminal flow). Pass `backend=niva.engine.MockBackend()` to dry-run without QGIS. Importing `niva` is always safe; QGIS is imported lazily only when a real run executes. See the Reference for the full signatures.

---

5. Configuration & environment

niva is configured entirely through environment variables (the plugin Setup tab sets them for you). The full table is in the Reference; the ones you are most likely to set:

Variable	Why
`NIVA_TMPDIR`	scratch dir for raster intermediates — set it to a roomy disk (§6)
`NIVA_LOG`	default journal base path (§7)
`NIVA_TEMPLATES`	extra directory of named project templates
`NIVA_NTFY_TOPIC` (+ `…_SERVER`, `…_TOKEN`)	the `notify` verb
`NIVA_SMTP_HOST` / `…_USER` / `…_PASSWORD` / `…_FROM`	the `email` verb
`QGIS_PREFIX_PATH`, `QT_QPA_PLATFORM`	standalone QGIS bootstrap

Credentials for `notify`/`email` and for databases come **only** from the environment / QGIS — never from flow text.

---

6. Scratch space & large rasters

Raster steps (`warp`, `clipraster`, `hillshade`, resampling, …) each write a full intermediate GeoTIFF — often gigabytes — before save re-encodes the final product. By default these land in the system temp dir, which on Linux is frequently a small RAM-backed `/tmp` (e.g. 16 GB). A multi-GB raster pipeline can exhaust it and abort with a "disk quota exceeded" error **even with terabytes of real disk free**.

Fix: set `NIVA_TMPDIR` (or the plugin's *Raster scratch* field) to a roomy, disk-backed folder:

```
export NIVA_TMPDIR=$HOME/niva_scratch
```

niva routes its intermediates there (and points GDAL's own `CPL_TMPDIR` at the same place). Scratch is **purged on every run, even a failed one**, so a crash never strands gigabytes; the run's final output is spared, and a niva-created scratch dir is removed when emptied. The plugin defaults this to a folder under your QGIS profile, always on real disk.

---

7. The run journal

Pass `--log <base>` (CLI), `log=<base>` (`flow`), or `NIVA_LOG`, or enable per-session logging in the plugin Setup tab, to record a run to two files:

- **`<base>.log`** — human-readable, one line per operation: timestamp, the stage text, the resolved algorithm, the output path, and the duration; warnings (⚠) and failures (✗ FAILED) are inlined.
- **`<base>.jsonl`** — machine-readable JSON Lines: a run-start record, one record per operation (including the exact `processing.run(...)` equivalent), and a run-finished summary.

The journal records the niva stage text and output paths — never raw parameter dicts, credentials, or SQL text. `log_append=True` (the plugin's per-session mode) keeps one growing log; the default truncates per invocation.

---

8. Export to / import from PyQGIS

niva flows convert to and from plain PyQGIS, so you can hand a script to someone without niva, or drop a flow into a larger Python tool:

- **Export** — `niva export flow.niva -o flow.py` (or the plugin Convert tab) writes a standalone PyQGIS script, one `processing.run(...)` per step.
- **Import** — `niva import flow.py -o flow.niva` recovers a flow from a **niva-shaped** script (a flat list of `processing.run(...)` calls, like Export produces). Arbitrary PyQGIS with loops/conditionals/functions can't round-trip and is reported, not guessed.

This is the audit trail: every niva run is reducible to the exact QGIS calls it made.

---

9. Databases & credentials

niva reads and writes SpatiaLite and PostGIS through **saved QGIS connections**, referenced as `@name` (see the Reference):

```
load @pg.public.roads | clip aoi.gpkg | save @pg.public.roads_clip mode=replace
sql @pg "SELECT id, ST_Buffer(geom,100) AS geom FROM homes WHERE has_cat" | save
 ↪  targets.gpkg
```

Set up the connection once in the **QGIS Browser** (Data Source Manager → PostgreSQL / SpatiaLite → New) or the plugin Setup tab. Only the connection *name* appears in a flow; host, database, user, and password stay in QGIS's own store. niva never logs or transmits credentials. See the SQL recipes in the Cookbook.

---

10. Troubleshooting

"could not import QGIS — run niva with QGIS's own Python." You ran the CLI/API on a plain Python that has no PyQGIS. If niva is `pip installed` **into QGIS's interpreter**, it auto-finds the bindings (0.55.1+) — this message then means you're on the wrong interpreter entirely; use QGIS's Python. Running from a **clone**, set PYTHONPATH to its bindings (§3). If your QGIS is in a non-standard place, point niva at it with `NIVA_QGIS_PYTHONPATH=/path/to/qgis/python`. `--dry-run` / `--explain` work without QGIS if you only need to validate.

A raster run aborts with "disk quota exceeded" / "No space left" despite free disk. Your scratch is on a small RAM-backed `/tmp`. Set `NIVA_TMPDIR` to a roomy disk folder (§6).

`notify` fails with "needs a topic." Set `NIVA_NTFY_TOPIC` (or pass `to=<topic>`), or fill the ntfy fields in the plugin Setup tab. niva won't send to an unconfigured topic.

`load mydata.gpkg` errors and lists layer names. A multi-layer container needs an explicit layer: `load "mydata.gpkg|layername=roads"`.

`save @pg.table` fails because the table exists. That's the fail-closed default (`mode=create`). Use `mode=replace` to overwrite or `mode=append` to add rows.

A `sql` write piped into another stage errors. A write statement (`CREATE`/`UPDATE`/…) is terminal — it returns no layer. Only `SELECT`/`WITH`/`VALUES`/`TABLE`/`EXPLAIN`/`SHOW` produce a pipeable layer.

One file in an each batch failed but the run continued. That's intended — a bad *data* item is skipped, logged, and counted so it can't abort the batch. A usage error (bad option, bad target) still stops everything. Check the output/journal for the skipped item.

Headless QGIS exits with a segfault after the run finishes. Known QGIS teardown race; the CLI hard-exits to avoid it, and results are already written. If you script the Python API directly, let the process exit rather than re-running in a loop.

niva Reference

The complete reference — every verb, alias, option, type, environment variable, CLI command, and Python entry point. For a task-oriented tour see the Cookbook; for setup and day-to-day use see the User Guide.

Algorithm coverage at a glance

niva gives **48 alias verbs** friendly names; **every** QGIS Processing algorithm — **878** in QGIS 4 — is reachable through the `run` escape hatch. The complete, per-algorithm appendix (parameters, defaults, enum options, descriptions, and which verb aliases each) is in `docs/algorithms/`.

Provider	Algorithms	niva alias verbs
`native:`	339	40
`gdal:`	59	5
`grass:`	307	0
`qgis:`	39	0
`pdal:`	24	0
`otb:`	109	0
`3d:`	1	0
Total	**878**	**45**

Plus niva's native-CLI harness (not QGIS providers): `pdalcli:` (12 PDAL commands on raw LAS/LAZ/COPC) and `saga:` (`saga_cmd`) — and **3 more alias verbs** wrap it for point clouds: **`dtm`**, **`dsm`**, **`hag`** (48 aliases total). See the PDAL/LAStools guide.

Data types. Discovery (`each`/`find`) and reporting (`show`/`catalog`) span every type QGIS reads — **vector · raster · point cloud · mesh** — from a broad built-in format list plus the live GDAL driver list when running under QGIS. Point clouds (`.las`/`.laz`/`.copc.laz`/`.vpc`/`.e57`/…) and mesh (`.2dm`/Telemac/UGRID/…) are recognised and summarised; processing verbs exist for point clouds today (`dtm`/`dsm`/`hag`), with mesh reachable via `run`.

- 1. The flow model
- 2. Syntax
- 3. Value types & units
- 4. Built-in verbs
- 5. Alias verbs (the registry)
- 6. The `run` escape hatch & `describe`
- 7. Database connections (`@conn`)
- 8. Environment variables
- 9. Command-line interface
- 10. Python API

---

1. The flow model

A niva program is one or more **flows**, one per line. A flow is a chain of **stages** separated by the pipe |. Each stage is a **verb** with optional arguments and options. A layer handle flows left-to-right: a stage receives the upstream layer, transforms it, and passes the result on.

```
load roads.gpkg | reproject EPSG:2262 | buffer 100m dissolve | save roads_buf.gpkg
```

Verbs fall into three behavioural classes:

Class	Meaning	Examples
Producing	returns a new layer to pipe onward	`load`, `run`, `split`, `sql` (SELECT), and every alias verb
Pass-through	returns the upstream layer unchanged, so it chains	`save`, `assess`, `metadata`, `style`, `notify`, `email`
Terminal	returns nothing; a following stage is an error	`catalog`, `show`, `info`, `describe`, `search`, `docs`, `project`, `sql` (write)

A flow normally begins with `load` (or `run`/`sql`, which produce a layer), or with `each` to run the rest of the flow once per dataset in a directory/glob/container.

Where verb output goes

Verbs produce three kinds of output, and each reaches you the same way no matter how niva is running (the QGIS plugin dock, the CLI, or the Python API):

1. **The layer** — *producing* and *pass-through* verbs hand a layer down the pipe. You persist it with `save`; in the plugin the flow's final layer is also added to the map and summarised ("done — , N feature(s)").
2. **A report** (the verbs that exist to *tell you something*: `show`, `info`, `describe`, `search`, `docs`). The **same report text** is routed to one of three sinks:
   - **`to=<file>`** — written to that text file, plus a one-line `… → <file>` status. This is how you capture a reply in a file from the CLI: `niva describe buffer to=buffer.md`, `niva "show @gisdb3.public to=tables.md"`, `niva info to=env.md`.
   - **the plugin dock** — when run inside QGIS, the report streams into the dock's output panel line-by-line, so you read it without leaving the dock.
   - **stdout** — from the CLI or the Python API with no `to=`, the report prints to stdout, so it stays pipeable and shell-redirectable: `niva describe buffer > buffer.md`.
3. **A status line** — verbs that *act* rather than report (`save`, `assess`, `catalog`, `style`, `metadata`, `notify`, `email`, `remove`, `project`) emit one short confirmation — e.g. `assessment → report.md`, `style saved → out.qml`, `notified → https://ntfy.sh/…`, `metadata set: title (persisted on next save)` — which shows in the dock and the CLI alike, so an action is never silent.

In every case progress is also reported per stage (a ▶ start line and a ✓ elapsed line) and recorded in the run journal (see the User Guide, "The run journal"). `describe` is available both as a flow stage (so it works in the dock and mid-script) and as the `niva describe` CLI subcommand.

---

2. Syntax

Stages, args, options, flags

A stage is `verb [positional-args…] [key=value options…] [flags…]`, whitespace-separated.

- **Positional arguments** are bound in order (e.g. `buffer 100m` → the distance).
- **Options** are key=value (e.g. `segments=12`, `cap=flat`).
- **Flags** are bare words that switch a boolean on (e.g. `dissolve`, `percent`).

```
buffer 100m dissolve cap=flat segments=12
#      ^arg  ^flag    ^option  ^option
```

Quoting, comments, line joins

- **Quotes** keep a value with spaces or special characters atomic: `filter "landuse = 'R'"`, `load "my data.gpkg|layername=roads"`. Single or double quotes; the outer pair is stripped.
- # starts a **comment** to end of line. A line that is only a comment is ignored.
- A flow **continues across lines** when a | sits at the **end of a line or the start of the next** (`niva/grammar/parser.py`) — split long flows *between stages* on the pipe. A backslash \ is **not** a line-continuation; a single stage's args/options must stay on one line, and a quoted string cannot span lines.
- Blank lines are ignored. Each non-blank line is its own flow.

Paths

A path argument has a leading ~ expanded to your home directory (`clip "~/aoi.gpkg"`), and **environment variables** expanded — $VAR / ${VAR} — so $HOME, $NIVA_TMPDIR, or any exported variable works inside a `.niva` file and the repl, not just a shell one-liner (`save "$OUTDIR/{name}.tif"`). Expansion is applied to **path values only**, so QGIS expression variables ($area, $geometry, $id) in a `filter` are never touched. A layer inside a multi-layer container is addressed with the OGR suffix `|layername=<name>`: `load "data.gpkg|layername=roads"`.

Per-item placeholder {name}

Inside an each batch, {name} in a save path or as clause is replaced with the current item's (sanitised) name — `save "out/{name}.tif"`. Outside a batch it is an error.

Option keys

Option keys are identifier-like and may contain internal hyphens (`from-template=`, `to-template=`). Only tokens containing = are treated as options, so flags such as `-deep` are never mistaken for options.

---

3. Value types & units

Argument and option values are coerced by **type** (the binder in `niva/registry/binder.py`):

Type	Accepts	Notes
`distance`	number + optional unit, e.g. `100`, `100m`, `1km`, `0.5deg`	unit resolved against the layer CRS at run time; **no unit = the layer's own CRS units**
`number`	decimal, e.g. `2`, `0.25`, `-9999`	
`int`	integer, e.g. `12`	
`enum`	one word from a fixed vocab, e.g. `cap=flat`	word → QGIS code; invalid word lists the valid set
`enumlist`	comma-separated enum words, e.g. `stats=mean,min,max`	
`fields` / `list`	comma-separated list, e.g. `fields=pop,income`	
`layer` / `raster`	a dataset path or `@conn.table`	~ expanded; @conn passes through
`crs`	a CRS string, e.g. `EPSG:2262`	verbatim

Type	Accepts	Notes
`field`	an attribute field name	verbatim
`string`	any text	verbatim
`expression`	a QGIS expression, e.g. `"area > 2000"`	verbatim; quote it

Supported distance units: `m`, `km`, `cm`, `mm`, `ft`, `yd`, `mi`, `nmi`, `deg`. An unknown unit is a clear error. A distance with no unit is interpreted in the layer's CRS units (so on a geographic CRS, `buffer 100` means 100 *degrees* — usually you want `100m`).

Enum-by-word: options that map to a QGIS dropdown take a readable word, not a number — `simplify 5m method=area`, not `method=2`. Each alias below lists its vocab.

---

4. Built-in verbs

Built-in verbs are handled directly by the engine (not the alias registry).

`load` — read a dataset *(producing)*

```
load <path-or-uri>
load "<container.gpkg>|layername=<layer>"
load @conn[.schema].table
```

Exactly one argument; no options. ~ is expanded. A bare multi-layer container (`load data.gpkg`) errors and lists its layers — name one with `|layername=`. The `@` form reads a table from a **saved database connection** (§7); a bare `@conn` with no table errors (use `sql` to query it), and an `@` pointing at a file extension is rejected (use the path form).

A **`.csv/.tsv/.txt` with longitude/latitude columns** (`longitude`/`latitude`, `lon`/`lat`, `lng`, `long`) loads as **points in EPSG:4326**, so it's directly geoprocessable. Detection is conservative — projected x/y (CRS unknowable) and CSVs with no coordinate columns load as an aspatial table, unchanged. For non-standard column names, load an OGR `.vrt` that names them.

```
load "parcels.gpkg|layername=residential"
load @pg.public.roads
load "earthquakes.csv"          # longitude/latitude → points (EPSG:4326)
```

`save` — write the current layer *(pass-through)*

```
save <path>
save <path> as <layer>                      # named layer inside a .gpkg/.sqlite/.db
save @conn[.schema].table [mode=create|replace|append]
```

- **File form:** extension chooses the format; parent directories are created. No options. `as <layer>` writes a named layer into a container (GeoPackage/SQLite only); writing into an existing container with a layer name **appends**.
- **Database form:** `mode` is the only option — `create` (default; **fails if the table exists** — never a silent overwrite), `replace` (drop + recreate), or `append` (insert, fields matched by name, CRS-transformed as needed). Rasters cannot be saved to a database.
- `save` chains: `… | save out.gpkg | style apply s.qml | notify "done"`.
- **In an each batch:** a file container gets **one layer per item** (named after the source); a `{name}` path gives **one file per item**; `save @conn` writes **one table per item** (a trailing qualifier is the schema, e.g. `save @pg.niagara`).

```
load roads.gpkg | clip aoi.gpkg | save out/roads_clip.gpkg
load roads.gpkg | save data.gpkg as roads
load roads.gpkg | save @pg.public.roads_clip mode=replace
```

`remove` — delete a file output and its sidecars *(terminal)*

```
remove <path> [force] [-dryrun]
each "<glob|dir>" | remove [force] [-dryrun]      # batch — each file checked individually
```

Deletes a **file** and its **sidecar family** (a shapefile's `.shx`/`.dbf`/`.prj`/…, a GeoPackage's `-wal`/`-shm`, a project's `_attachments.zip`, plus any `.aux.xml`/`.qml`/`.qmd`). It is the one destructive verb, so it runs a strict, fail-closed safety gate — every refusal is a specific message, and nothing is deleted unless all checks pass:

- **`@conn` refs are refused** — `remove` is files-only; drop a table with `sql @conn "DROP TABLE …"`.
- **Globs are refused** — batch with `each "<glob>" | remove` so each file is validated.
- **Directories are refused** — name a file (or `each "<dir>" | remove` its contents).
- **A missing path is a no-op** (success) — safe to re-run as cleanup.
- Only **recognised geodata / niva-output types** are deletable (`.gpkg .shp .tif .qml .qgs …`); anything else needs **`force`**, which deletes *only* that one file (no sidecar guessing).
- **`-dryrun`** logs the exact deletion plan without deleting — a safe ”what would this remove?”.

```
load roads.gpkg | clip aoi.gpkg | save tmp_clip.gpkg | …
remove tmp_clip.gpkg                       # tidy up the scratch output (+ its sidecars)
each "/tmp/scratch/*.gpkg" | remove        # batch cleanup
remove run_report.md force                 # delete a non-geodata file niva wrote
each "out/*.tif" | remove -dryrun          # preview only
```

`sql` — query or mutate a database connection *(producing OR terminal)*

```
sql @conn "<statement>"
```

Exactly two arguments — a **bare** connection ref and a quoted statement — and no options. The leading keyword routes it:

- **`SELECT` / `WITH` / `VALUES` / `TABLE` / `EXPLAIN` / `SHOW`** → a **query layer** you can pipe onward (producing). `CREATE TABLE … AS SELECT …` reads as a **write** (leading `CREATE`); `WITH … SELECT …` reads as a query (leading `WITH`).
- **anything else** (`CREATE`/`INSERT`/`UPDATE`/`DELETE`/`DROP`/`ALTER`/…) → runs server-side and returns nothing (terminal).

Works against any saved connection — **SpatiaLite** or **PostGIS** — using that engine's spatial SQL (`ST_Buffer`, `ST_Area`, spatial joins, …). Only the connection *name* leaves the flow; credentials never appear in flow text, errors, or logs (§7).

```
sql @cats_pg "SELECT id, ST_Buffer(geom, 100) AS geom FROM homes WHERE has_cat" | save
 ↪  targets.gpkg
sql @pg "CREATE TABLE roads_buf AS SELECT id, ST_Buffer(geom,100) AS geom FROM roads"
```

`run` — call any QGIS algorithm directly *(producing)*

```
run <algorithm-id> [KEY=value …]
```

The escape hatch: one positional (the algorithm id, e.g. `native:slope`), then QGIS parameters as `KEY=value`. Values are coerced (`true`/`false` → bool, numbers → int/float, else string); `~` is expanded; a `;`-joined value becomes a list; a path with `*`/`?`/`[` is globbed. `INPUT` (from the pipe) and `OUTPUT` (a temp file) are supplied automatically when omitted. See §6.

```
load dem.tif | run native:slope Z_FACTOR=2 | save slope.tif
run gdal:merge INPUT="a.tif;b.tif;c.tif" DATA_TYPE=5 | save mosaic.tif
```

`split` — keep one geometry type *(producing)*

```
split <point|line|polygon>
```

Extract a single geometry type from a mixed layer (native:filterbygeometry). Accepts singular or plural (points, lines, polygons). Pipe again to extract another type. Multipart features are preserved; GeometryCollection features match none of the sinks.

```
load mixed.gpkg | split line | save lines.gpkg
```

metadata — stamp descriptive metadata *(pass-through)*

```
metadata set <field=value> [field=value …]
```

Fields: title, abstract, keywords (comma-separated), identifier, license. The metadata persists on the next save.

```
load parcels.gpkg | metadata set title="Residential parcels" keywords="parcels,zoning" |
 ↪  save parcels.gpkg
```

assess — data-quality report *(pass-through)*

```
assess [deep] to <report.md>
```

Profiles the current layer to a Markdown report: type, CRS, extent, fields, metadata, and — with deep — invalid/empty/duplicate-geometry counts and per-field null counts. Sits mid-pipe (returns the layer unchanged). deep may also be written -deep.

```
load raw_parcels.gpkg | assess deep to docs/raw_quality.md
```

catalog — inventory a directory *(terminal)*

```
catalog <dir> [to=<out.md>]
```

Recursively profiles every geospatial dataset under <dir> (multi-layer containers expand per layer) to a Markdown inventory — type, CRS, geometry/bands, feature count, extent — listing unreadable files separately. Default output is <dir>/catalog.md.

```
catalog "data/" to=reports/inventory.md
```

show — list available data at a location *(terminal)*

```
show <path|@conn[.schema[.table]]|service-url> [deep] [to=<out.md>]
```

Lists the loadable layers/tables at **one** location and what each is — a quick *"what can I load here, and what's its name?"* glance, the lighter cousin of catalog (which deep-profiles CRS/extent/fields/counts). Per entry: the **name**, **kind** (vector/raster/table), **type** (geometry like MultiPolygon, or a raster's N band · Float32), **format** (the file driver GPKG / GTiff / SQLite / …, or the DB provider), and a copy-pasteable **source** you can hand straight to load (or ogrinfo for files). No feature counts — that keeps it instant even on big databases.

<location> may be:

- a **file** — show roads.shp, show dem.tif, or a multi-layer container show data.gpkg (one row per layer);
- a **directory** — show data/ lists the immediate children; add the **deep** flag (show data/ deep) to recurse the whole tree. **Any QGIS-readable format is picked up** — not just a fixed extension list — so SpatiaLite (.sqlite/.db), FileGDB (.gdb), and any other GDAL/OGR driver appear; dataset sidecars (a shapefile's .dbf/.shx/…) and obviously non-geospatial files are skipped. Directory-based datasets like .gdb are listed as a container, not descended into;
- a **database connection** — show @conn (every table), show @conn.schema (one schema), show @conn.schema.table (one table). Connection names containing dots (e.g. @actual_spatialite.sqlite) resolve correctly. Only the connection **name** is used — credentials stay in QGIS;

- a **remote service** — a **WFS** endpoint lists its feature types, a **WMS** endpoint its layers, an **ArcGIS REST** service (…/FeatureServer, …/MapServer, …/ImageServer) its layers and tables, and an **XYZ** {z}/{x}/{y} tile template is itself one layer. The kind is read from a service=WFS/service=WMS query parameter, the URL path/shape, or a WFS:/WMS: prefix (if it can't be told, show asks you to specify). Standard-library HTTP only — public services, no credentials sent; responses are fetched with a timeout and size cap, XML is parsed with DOCTYPE/entity expansion refused (no XXE), and ArcGIS REST JSON is read with json (no entity risk).

```
show "data/basemap.gpkg"          # layers in a GeoPackage
show data/                         # everything directly under data/
show data/ deep                    # recurse the whole tree
show @gisdb3                       # all PostGIS tables
show @gisdb3.public to=tables.md   # one schema, written to a file
show "https://demo.mapserver.org/cgi-bin/wfs?service=WFS"   # WFS feature types
show "https://ows.terrestris.de/osm/service?service=WMS"    # WMS layers
show "https://server/arcgis/rest/services/Foo/FeatureServer"  # ArcGIS REST layers
show "https://tile.osm.org/{z}/{x}/{y}.png"                # XYZ tile layer
```

The listing's footer includes **two runnable examples** built from the first row, so you can copy a real Source and pipe it onward (e.g. load "data.gpkg|layername=roads" | buffer 100m | save out.gpkg). show is terminal and can't itself be piped (neither it nor catalog produce or consume a layer); for a deep per-dataset report (CRS, extent, fields, feature counts) run catalog on the same location.

info — inspect the local QGIS environment *(terminal)*

```
info [to=<report.md>]
```

Reports what a CLI user needs to know before writing a flow, especially when working **outside** QGIS where the Browser and connection dialogs aren't in front of you. Most useful: the **registered database connection names** — the valid @conn references for PostGIS and SpatiaLite — since a flow names them but you can't guess them. Because connections are **per QGIS profile**, info also lists **every profile and the connections in each**, and marks the active one (niva reads the active profile by default; NIVA_QGIS_PROFILE=<name> targets another). Also surfaces the Processing **providers** and reachable **algorithm count** (so you know run grass:… / run pdal:… will work), the **versions** (QGIS, Qt/PyQt, GDAL, PROJ, GEOS, SpatiaLite/SQLite, Python), niva's own build + import path, the verb list, and the **environment variables** niva honours (secrets masked — only *set* / *unset* is shown). With to=, writes the Markdown to a file; otherwise prints it to stdout. This is the CLI counterpart of the plugin's Setup-tab **Environment report**.

```
info                       # print the report
info to=env-report.md      # save it
```

(Not to be confused with project info <src.qgs>, which inventories a *project file*; bare info inventories the *QGIS environment*.)

describe — introspect a verb or algorithm *(terminal)*

```
describe <verb|algorithm-id> [to=<report.md>]
```

Shows how a niva verb maps to its QGIS algorithm — positional args, options (with defaults and enum values), flags, and a **runnable example** (a curated one for common verbs, otherwise one synthesised from the signature). Given an **algorithm id** (anything containing :, e.g. native:buffer, gdal:warpreproject) it introspects the *live* algorithm's parameters and outputs, and ends with a synthesised run <id> … example — making the run <id> KEY=value escape hatch (see §6) discoverable: describe an algorithm, then run it. Describing a verb is pure (no QGIS needed); describing an algorithm id needs QGIS's Python. With to= the report is written to a file; otherwise it streams to the plugin dock or prints to stdout (see ”Where verb output goes” in §1).

```
describe buffer                     # the buffer verb → native:buffer, options, flags,
↪  example
describe gdal:warpreproject         # a live algorithm's parameters + a run example
describe buffer to=buffer.md        # capture the report to a file
```

`describe` runs both as a flow stage (so it works inside the QGIS plugin dock and mid-script) and as the `niva describe <verb-or-algorithm-id> [to=<file>]` CLI subcommand (which works without QGIS for a verb).

`search` — find verbs and algorithms by keyword *(terminal)*

```
search <keyword> [to=<report.md>]
```

Fuzzy-searches everything niva knows about every function — niva verb names, summaries, options and flags; the built-in verbs; and the **live QGIS algorithm catalog** (id, display name, group, description; 878 in QGIS 4.0.3) — and lists the ranked matches (name · kind · score · summary). The match is tolerant: substrings, token prefixes, and `difflib` similarity all count, so a typo or a partial word still finds things. A multi-word keyword is OR-matched (`search "raster reproject"` finds anything about rasters *or* reprojection). Needs QGIS (it reads the live catalog).

```
search reproject                    # verbs + algorithms related to reprojection
search "raster slope" to=hits.md    # save the ranked list
```

Then run `describe <name>` for any single result, or `docs <keyword>` to get the full reference of every match at once.

`docs` — build a mini-guide for a keyword *(terminal)*

```
docs <keyword> [to=<report.md>]
```

Fuzzy-searches like `search`, then emits the **full `describe`** (args, options, flags, example) for *every* match, concatenated — a made-to-order reference for whatever you're doing in the moment. With `to=<file>` you get your own task-specific guide as a Markdown file. (This is the combined "search → describe → file" workflow as one verb, since niva's pipe carries a layer, not a list of names.) Needs QGIS.

```
docs buffer                         # full reference for every buffer-related function
docs "raster terrain" to=terrain_guide.md   # a saved guide for terrain work
```

`project` — manipulate QGIS project files *(terminal)*

Five forms; all use a standalone `QgsProject` (safe off the GUI thread) and all are terminal.

Repoint / copy / convert — copy a project, optionally repointing layer datasources:

```
project <src.qgs|qgz> to=<out.qgs|qgz> [repoint=<target>]
        [missing=fail|keep|drop] [rasters=<dir>] [paths=relative|absolute]
        [bookmark=<name> [at="x,y"] [scale=<N>] [width=<W>]]
```

`repoint=` is a `.gpkg` path **or** an `@conn[.schema]` connection; vector layers match by name, subset filters preserved. `missing=` (default **`fail`**) handles a layer absent from the target. `rasters=<dir>` repoints raster layers by basename. `paths=` rewrites datasource path storage. `bookmark=<name>` adds a spatial bookmark — the layers' **union** extent, or a centred box via `at="x,y"` with `width=` (exact) or `scale=` (approx). Omit `repoint=` to just copy/convert (`.qgs`↔`.qgz` by extension).

`project new` — build a fresh project from a folder of outputs:

```
project new from=<dir|glob> to=<out.qgs|qgz> [crs=<crs>] [title=<text>]
```

`project info` — inventory a project to Markdown:

```
project info <src.qgs|qgz> [to=<out.md>]
```

`project from-template=` — instantiate a template against your data (see Template projects):

```
project from-template=<name|path> to=<out.qgs|qgz> data=<dir|glob>
 ↪  [missing=keep|fail|drop]
```

The template is any existing project; each layer **slot** is repointed to the same-(display)- named dataset under `data=`, symbology and print layouts riding along. Resolves a bare name from `$NIVA_TEMPLATES` → `~/.niva/templates` → bundled (`example`). `missing=` default keep.

`project to-template=` — register an existing project as a reusable template:

```
project to-template=<name|path> from=<src.qgs|qgz> [paths=relative|absolute]
```

```
project "basemap.qgs" to="out/basemap.qgs" repoint="out/basemap_clip.gpkg" missing=keep
project from-template=example to="region_atlas.qgz" data="data/clips/"
```

`style` — apply or export a layer style *(pass-through)*

```
style apply <file.qml|.qmd>
style save  <file.qml|.qmd|.sld|.qlr>
```

`apply` reads a QGIS sidecar and **persists** it (a GeoPackage layer style goes into the container's `layer_styles` table; a single-file layer gets a same-basename sidecar). `save` exports the current layer's style/metadata — `.qml` (symbology), `.qmd` (metadata), and export-only `.sld` (OGC) and `.qlr` (portable layer definition). Chains after `save`.

```
load roads.gpkg | clip aoi.gpkg | save roads_clip.gpkg | style apply house.qml
```

`figure` — render a quick map image *(pass-through)*

```
figure <out.png|.jpg> [size=WxH] [dpi=N] [extent=layer|x1,y1,x2,y2|<layer>]
                      [layers="a;b"] [basemap=osm|<xyz-url>] [bg=<colour>] [labels=<field>]
```

Renders a map **image** of the current layer — **vector or raster** — with sensible defaults, so `load x | figure out.png` alone shows something useful: the full data extent (with a small margin), a 1200 px-wide canvas at the data's aspect, antialiasing, a white background, single-band rasters min/max-stretched, and each layer's own style (labels included when its style has them). Options layer more on top: `layers=` overlays extra sources (drawn beneath the piped one), `basemap=osm` adds an OpenStreetMap tile layer underneath, `labels=<field>` turns on simple labeling, and `extent`/`size`/`dpi`/`bg` control the framing. Pass-through, so it chains.

```
load dem.tif | figure hillshade.png                                  # simplest — just works
load parcels.gpkg | figure map.png layers="roads.gpkg" basemap=osm labels=owner
 ↪  size=1600x1000
load flood.gpkg | save flood.gpkg | figure flood.png                  # chains after save
```

For a composed cartographic layout (title, legend, scale bar, north arrow → PDF/SVG), use the richer `map` verb below.

`map` — composed cartographic layout *(pass-through)*

```
map <out.pdf|.png|.jpg|.svg> [title="…"] [legend|nolegend] [scalebar|noscalebar]
    [northarrow|nonortharrow] [bare] [page=A4|A3|Letter|WxH] [portrait|landscape] [dpi=N]
    [layers="a;b"] [basemap=osm|<xyz-url>] [labels=<field>] [extent=…]
map <out.pdf> from=<project.qgz> [layout=<name>]
```

Builds a page **layout** with the map plus a **legend, scale bar, and north arrow — on by default**, so a bare `load x | map out.pdf` yields a complete map with no template. Shares `figure`'s layer model (piped layer + `layers=` overlays + `basemap`, `labels`, `extent`) and handles vector *and* raster. `bare` strips the decorations; `no<element>` drops one; `page`/`orientation`/`dpi` control the sheet (A4 landscape, 300 dpi by default). Writes PDF/PNG/JPG/SVG.

The `from=` form exports an **existing QGIS print layout** (named, else the first) at full fidelity — the way to ship hand-designed plates and atlases. Pass-through, so it chains.

```
load dem.tif | map dem.pdf                                        # complete map, one line
load flood.gpkg | style apply=flood.qml | map flood.pdf title="Flood Risk"
 ↪  layers="roads.gpkg" basemap=osm labels=risk
load aoi.gpkg | map atlas.pdf from=study.qgz layout="Overview"
```

Rendering large datasets can take a while; niva streams progress (per-layer load + a render heartbeat) so a long render never looks frozen.

`notify` — push a message via ntfy *(pass-through)*

```
notify "<message>" [to=<topic>] [title=<t>] [priority=<p>] [server=<url>] [tags=<tags>]
```

Posts to an ntfy topic. `to=` falls back to `NIVA_NTFY_TOPIC`; `server=` to `NIVA_NTFY_SERVER` (default `https://ntfy.sh`); a bearer token comes only from `NIVA_NTFY_TOKEN`. The message interpolates `{elapsed}`, `{last}`, `{now}`, `{started}`, `{ops}`, `{errors}`. See also the auto-alert env vars in §8.

```
… | save out.gpkg | notify "done in {elapsed}, {errors} errors" to=geo-jobs priority=high
```

`email` — send via SMTP *(pass-through)*

```
email to=<address> [subject=<s>] [body=<b>] [attach=<file>]
```

SMTP host/credentials come only from the environment (`NIVA_SMTP_*`, §8); a `@gmail.com` sender auto-uses `smtp.gmail.com:587` (needs a Gmail App Password). TLS is mandatory.

```
load r.gpkg | assess to q.md | email to=ops@example.com subject="Daily run" attach=q.md
```

`each` — batch over many datasets *(flow prefix)*

```
each "<dir>"  | <stages…> | save <target>
each "<glob>" | <stages…> | save out.gpkg
each <file.gpkg> | <stages…> | save out.gpkg
```

Must be the **first** stage. Resolves a directory (recursed), a glob, or a multi-layer container (expanded per layer) to a list of datasets, then runs the rest of the flow once per dataset. A failing item is **skipped** (logged, counted, warned) so one bad file can't abort the batch; a usage error (bad option/target) still stops everything. See save for per-item output behaviour.

```
each "in/*.shp" | reproject EPSG:6346 | save out.gpkg
```

Filters — narrow the batch with flat `option=value` filters, the same vocabulary as `niva find` (so a filtered each selects exactly what `find` would show). Offline filters — `ext`, `minsize`, `maxsize`, `newerthan`, `format` — need nothing; the GDAL-enriched filters — `geom`, `crs`, `minfeatures`, `maxfeatures`, `hasfield` — need QGIS's Python (each loads via QGIS anyway). Sizes take k/M/G (e.g. `minsize=1M`); `newerthan` takes `30s`/`24h`/`7d`.

```
each "data/**/*.gpkg" geom=polygon minfeatures=1 | dissolve | save out.gpkg
each "tiles/*.tif" newerthan=7d | hillshade | save "out/{name}.tif"
```

`call` — run another `.niva` file inline *(statement)*

```
call <other.niva>
```

A top-level statement (not a piped stage): runs another flow file's flows in place, resolved relative to the calling file, with cycle detection — for composing reusable flow libraries.

---

5. Alias verbs (the registry)

Each alias maps one verb to one QGIS algorithm, hiding its parameter names behind readable args/options/flags. The primary input is the piped layer; the output is the temp/saved destination. Source of truth: `niva/registry/definitions.py`. Run `niva describe <verb>` for the live signature.

Reading the tables: *Args* are positional (in order); a `[name]` is optional. *Options* are key= with the default in parentheses. *Flags* are bare switches (off by default). Enum options list their words.

Vector — core overlay & selection

Verb	Algorithm	Args	Options	Flags
clip	native:clip	overlay (layer)	—	—
intersect	native:intersection	overlay (layer)	—	—
difference	native:difference	overlay (layer)	—	—
union	native:union	[overlay] (layer)	—	—
symdifference	native:symmetricaldifference	overlay (layer)	—	—
dissolve	native:dissolve	[field]	—	separate
filter	native:extractbyexpression	expression	—	—
selectloc	native:extractbylocation	against (layer)	predicate= (intersect) ⟨intersect, contain, disjoint, equal, touch, overlap, within, cross⟩ — comma-list	—
spatialjoin	native:joinattributesbylocation	—	with=(req, layer) · predicate=(intersect) ⟨intersect, contain, equal, touch, overlap, within, cross⟩ · method=(one-to-many) ⟨one-to-many, first, largest⟩ · fields= · prefix=	discard
join	native:joinattributestable	—	with=(req) · field=(req) · field2=(req) · fields= · prefix= · method=(one-to-one) ⟨one-to-many, one-to-one⟩ · unmatched=	discard
sample	native:randomextract	number (int)	method=(count) ⟨count, percent⟩	—

Vector — buffering & geometry

Verb	Algorithm	Args	Options	Flags
buffer	native:buffer	distance	segments=(5) · cap=(round) ⟨round, flat, square⟩ · join=(round) ⟨round, miter, bevel⟩ · miter=(2)	dissolve · separate
offset	native:offsetline	distance	segments=(8) · join=(round) ⟨round, miter, bevel⟩ · miter=(2)	—
simplify	native:simplifygeometries	tolerance	method=(douglas) ⟨douglas, grid, area⟩	—
smooth	native:smoothgeometry	—	iterations=(1) · offset=(0.25) · max_angle=(180)	—
densify	native:densifygeometriesgivenaninterval	interval	—	—
subdivide	native:subdivide	—	max_nodes=(256)	—
centroid	native:centroids	—	—	—
pointonsurface	native:pointonsurface	—	—	all_parts
convexhull	native:convexhull	—	—	—
boundingbox	native:boundingboxes	—	—	—
minrect	native:orientedminimumboundingbox	—	—	—
vertices	native:extractvertices	—	—	—
explode	native:multiparttosingleparts	—	—	—
promote	native:promotetomulti	—	—	—
collect	native:collect	[field]	—	—
fixgeom	native:fixgeometries	—	—	—
swapxy	native:swapxy	—	—	—
forcerhr	native:forcerhr	—	—	—
snap	native:snapgeometries	reference (layer), tolerance	behavior=(align) ⟨align, closest, align-keep, closest-keep, ends-align, ends-closest, ends-only, anchor⟩	—

Vector — CRS, attributes, counting

Verb	Algorithm	Args	Options	Flags
reproject	native:reprojectlayer	target_crs	operation=	convert_curved · transform_z
renamefield	native:renametablefield	field, name	—	—
dropfields	native:deletecolumn	fields (list)	—	—
keepfields	native:retainfields	fields (list)	—	—
countpoints	native:countpointsinpolygon	—	points=(req, layer) · field=(NUMPOINTS) · weight= · classfield=	—
zonalstats	native:zonalstatisticsfb	—	raster=(req) · band=(1) · stats=(count,sum,mean) (count, sum, mean, median, stdev, min, max, range, minority, majority, variety, variance) — comma-list · prefix=(_)	—

Point clouds — the `pdalcli:` harness (needs `pdal_wrench`, `niva pdal check`)

Load a raw `.las`/`.laz`/`.copc.laz` and pipe it in; these produce a raster (pipes into `save`) or, for `hag`, a normalised cloud (pipe into `dsm` for a CHM). Filter/aggregation defaults come from the verb; drop to `run pdalcli:<cmd>` for the full option set.

Verb	Harness id	Options	Notes
`dtm`	`pdalcli:to_raster`	`resolution=(1)`	ground/bare-earth raster — forces `attribute=Z` `filter="Classification==2"`
`dsm`	`pdalcli:to_raster`	`resolution=(1)`	surface raster from **all** returns — forces `attribute=Z`
`hag`	`pdalcli:height_above_ground`	—	height-above-ground per point; `load x.las | hag | dsm | save chm.tif` = CHM

Vector — point creation

Verb	Algorithm	Args	Options	Flags
voronoi	native:voronoipolygons	—	buffer=(0)	—
delaunay	native:delaunaytriangulation	—	—	—
pointsalong	native:pointsalonglines	distance	start= · end=	—

Raster

Raster verbs default CREATION_OPTIONS to lossless COMPRESS=DEFLATE|TILED=YES so intermediates aren't left huge; the final product's compression is governed by save.

Verb	Algorithm	Args	Options	Flags
warp	gdal:warpreproject	target_crs	source_crs= · resampling=(nearest) ⟨nearest, bilinear, cubic, cubicspline, lanczos, average, mode, max, min, median, q1, q3⟩ · nodata= · resolution=	—
clipraster	gdal:cliprasterbymasklayer	mask (layer)	nodata=	—
hillshade	native:hillshade	—	z_factor=(1) · azimuth=(300) · altitude=(40)	—
slope	gdal:slope	—	band=(1) · scale=(1)	percent
aspect	gdal:aspect	—	band=(1)	trig · zero_flat
polygonize	gdal:polygonize	—	band=(1) · field=(DN)	eight

Anything not aliased is still reachable with `run <algorithm-id> …` (§6). For the full catalogue of all 878 algorithms — parameters, defaults, enum options, and descriptions — see the algorithm appendix.

---

6. The `run` escape hatch & `describe`

`run <id> KEY=value …` calls any installed QGIS Processing algorithm with its native parameter names — the full catalogue (878 in QGIS 4.0.3: `native:`, `gdal:`, `qgis:`, `grass:`, `pdal:`, `otb:`, `3d:` algorithms), not just the aliased verbs. Beyond the QGIS providers, niva's native-CLI harness adds `pdalcli:<command>` (PDAL on raw LAS/LAZ/COPC) and `saga:<library>:<tool>` (`saga_cmd`) — see the PDAL/LAStools guide. Use `run` for anything without an alias, or when you need a parameter an alias doesn't expose.

- `INPUT` is filled from the piped layer; `OUTPUT` from a temp file — supply either to override.
- Multi-input parameters take a `;`-separated list: `INPUT="a.tif;b.tif"`.
- A value with `*`/`?`/`[` is globbed relative to the flow file: `INPUT="tiles/*.jp2"`.
- Values coerce: `true`/`false` → bool, integers → int, decimals → float, else string.

```
load dem.tif | run native:slope Z_FACTOR=2 NODATA=-9999 | save slope.tif
run gdal:translate INPUT=in.vrt OUTPUT=out.jp2 CREATION_OPTIONS="QUALITY=25"
```

`describe <name>` introspects either a niva verb (its alias mapping, args, options, flags — no QGIS needed) or a QGIS algorithm id (its live parameters and outputs — QGIS needed). It runs both as a **flow stage** (so it works in the plugin dock and mid-script, with `to=<file>`) and as the **`niva describe`** CLI subcommand — see the `describe` verb in §4:

```
niva describe buffer                # CLI subcommand (no QGIS needed for a verb)
niva describe gdal:warpreproject
niva "describe buffer to=buffer.md"  # the same thing as a flow, captured to a file
```

The **algorithm appendix** lists all 878 algorithms by provider — each with its parameters (type, required, default, enum options), description, outputs, and niva alias (★) — so you can find and copy a `run <id> …` for anything without an alias.

---

7. Database connections (`@conn`)

niva reads and writes databases through **saved QGIS connections**. A reference is:

```
@conn                 # the connection itself (for sql)
@conn.table           # a table in the connection's default schema
@conn.schema.table    # a schema-qualified table
```

`@` always means a saved DB connection — never a file (a path is just a path). Used by `load @conn.table`, `save @conn.table`, `sql @conn "…"`, and `project … repoint=@conn`.

Default schema: explicit wins; otherwise `public` for PostgreSQL/PostGIS, and the provider default (`''`) for SpatiaLite/file databases.

Credentials never enter a flow. Only the connection *name* crosses from the flow into niva; host, database, user, and password are resolved from QGIS's own connection store (the single source of truth, configured in the QGIS Browser or the plugin Setup tab). niva never stores, logs, or transmits credentials — errors carry only the connection name and table, never the URI, password, or query text. SpatiaLite and PostGIS connections are used identically; the only difference is which engine's spatial SQL functions you write.

---

8. Environment variables

Variable	Default	Purpose
NIVA_TMPDIR	system temp	**Disk-backed scratch dir for raster intermediates.** Point it at a roomy real-disk folder so a multi-GB raster pipeline doesn't exhaust a small RAM-backed /tmp. Strongly recommended for raster work.
CPL_TMPDIR	follows NIVA_TMPDIR	GDAL's own scratch dir; niva points it at NIVA_TMPDIR unless you set it.
NIVA_LOG	unset	Default journal base path; writes <base>.jsonl + <base>.log. CLI --log overrides.
NIVA_TEMPLATES	~/.niva/templates	Extra directory for named project templates; shadows the user library and bundled templates. Also where project totemplate=<name> writes.
NIVA_QGIS_PROFILE	active desktop profile	**Which QGIS user profile to read** for database connections (the @conn names) and other settings, when running standalone. By default niva uses the profile your QGIS desktop last used; set this to another profile's name to target its connections. Run info to see all profiles and their connections.
QGIS_PREFIX_PATH	/usr	QGIS install prefix for standalone bootstrap.
NIVA_QGIS_PYTHONPATH	unset	**Where QGIS's Python bindings live**, if niva can't find them. When a standalone niva can't import qgis, it auto-discovers the bindings by probing this variable first, then QGIS_PREFIX_PATH, an inferred macOS .app bundle, and the OS defaults (/usr/share/qgis/python, /Applications/QGIS.app/…). Set it (the dir containing the qgis/ package, os.pathsep-joined for several) only when your QGIS is in a non-standard location.
QGIS_WRENCH_EXECUTABLE / NIVA_PDAL_WRENCH	pdal_wrench on PATH	Path to **pdal_wrench** — the point-cloud backend behind dtm/dsm/hag and run pdalcli:*. QGIS_WRENCH_EXECUTABLE is reused (it also configures QGIS's pdal: provider); niva pdal check diagnoses it.
NIVA_SAGA_CMD	saga_cmd on PATH	Path to saga_cmd for the run saga:* harness.
QT_QPA_PLATFORM	offscreen (set if unset)	Headless Qt platform for standalone runs.
NIVA_NTFY_TOPIC	unset	Default ntfy topic for notify.
NIVA_NTFY_SERVER	https://ntfy.sh	ntfy server URL.
NIVA_NTFY_TOKEN	unset	Bearer token for protected ntfy topics (never logged).
NIVA_NTFY_ON_ERROR	unset	When truthy, auto-send a high-priority ntfy when a run fails.
NIVA_NTFY_ON_WARNING	unset	When truthy, auto-send an ntfy on warnings (mixed geometry, datum transforms, skipped batch items), de-duplicated per run.
NIVA_SMTP_HOST	unset (auto for Gmail)	SMTP server for email.
NIVA_SMTP_PORT	587	SMTP port; 465 = implicit TLS.
NIVA_SMTP_USER	unset	SMTP username.
NIVA_SMTP_PASSWORD	unset	SMTP password (Gmail: an App Password).
NIVA_SMTP_FROM	= NIVA_SMTP_USER	From address; a @gmail.com value auto-selects the Gmail host.

Truthy = 1, true, yes, on (case-insensitive). Inside QGIS, the plugin's **Setup** tab sets these for you (and can store secrets in the QGIS encrypted store).

9. Command-line interface

The package installs a `niva` console script (`niva.cli.main`).

```
niva run <file.niva> [--dry-run | --explain] [--log <base>]
niva "<flow>"        [--dry-run | --explain] [--log <base>]
niva validate <file.niva> [more.niva …]
niva plan <file.niva> | "<flow>"
niva manifest [to=<file>]
niva describe <verb-or-algorithm-id> [to=<file>]
niva export <file.niva> [-o <file.py>]
niva import <file.py>   [-o <file.niva>]
```

Command / flag	**Effect**
`niva run flow.niva`	execute a `.niva` file (real, via PyQGIS)
`niva flow.niva`	same — the `run` is optional for a `.niva`/`.flow` file
`niva "load a.gpkg | buffer 100m | save b.gpkg"`	execute an inline flow
`--dry-run`	print the plan and validate it over the mock backend (no QGIS, no data touched)
`--explain`	parse + bind only; print the resolved algorithm + parameters per stage
`--log <base>`	also write the journal `<base>.jsonl` + `<base>.log`
`niva validate flow.niva …`	**offline linter** — grammar + closed-set verbs + alias args/options/enums + `run <id>` params, then a mock-backend dry-run; reports errors and style warnings with line numbers and did-you-mean hints (no QGIS)
`niva plan flow.niva` / `niva plan "<flow>"`	emit the **resolved plan** as JSON — the compiled flow as an *intermediate representation* (IR): each stage's resolved `provider:algorithm`, parameters, injected defaults, and diagnostics. The machine-readable contract downstream tools read (no QGIS)
`niva manifest [to=<file>]`	emit the **machine-readable verb catalog** as JSON — every verb's algorithm, parameters, defaults, enums, synonyms, and example, for IDEs / LSP / LLM agents (no QGIS)
`niva describe <name> [to=<file>]`	introspect a verb or algorithm id (with a runnable example)
`niva setup doctor`	**environment health check** — niva version, the QGIS runtime niva discovered (or a ✗ with the fix), Processing providers + algorithm count, geo stack, the PDAL point-cloud backend, the config file, and `@conn` connections. Read-only; exits non-zero only if something blocking (no QGIS) is found
`niva setup wizard`	**guided config** — an interactive walk-through of the portable settings (QGIS Python, log/scratch/templates dirs, QGIS profile, ntfy, SMTP): Enter keeps, a value sets, `-` clears. Secrets are never prompted — it only reminds which env vars to set. Writes the portable `config.toml`
`niva lsp`	**Language Server** over stdio for `.niva` files — completion (verbs → options → enums → paths), live diagnostics (the offline validator), and hover (`describe`) in any LSP editor (Neovim, Helix, VS Code, Kate). Same engine as the repl (`niva.intelligence`); stdlib JSON-RPC, no extra deps. See editor integration
`niva find [glob] [in <dir>…] [filters]`	discover spatial data on the filesystem (offline; GDAL filters when on QGIS's Python). Output: aligned table (default), `--json`, `--as-flow` (an each … skeleton), or **`--paths`** / **`-0`** — bare absolute paths for piping into other tools (`… --paths | xargs …`, `… -0 | xargs -0 …`, `… --paths > list.txt`)
`niva "search <kw>"` / `niva "docs <kw>"`	fuzzy-find functions / build a mini-guide (flow verbs; need QGIS)
`niva export flow.niva [-o out.py]`	transpile a flow to a standalone PyQGIS script
`niva import script.py [-o out.niva]`	recover a flow from a niva-shaped PyQGIS script

Exit codes: `0` ok · `1` runtime error (`OpError`) · `2` usage / parse error (`FlowError`) · `3` QGIS not importable. Progress and errors go to stderr; result markers to stdout. A real run needs QGIS's Python (see the User Guide).

---

10. Python API

The public API (`import niva`) is small and stable:

```
flow(text, *, backend=None, file=None, log=None, log_append=False, progress=None,
 ↪  cancel=None)
run_file(path, *, backend=None)
describe(name)
__version__
NivaError, FlowError, OpError          # exception types
```

`flow(text, …)`

Parse and execute a flow string; returns the final layer handle (its `.ref` is an on-disk path for a saved file, or a live `QgsMapLayer`), or `None` for a terminal flow.

Parameter	Default	Meaning
`text`	—	the flow string
`backend`	real `PyqgisBackend` (needs QGIS)	pass `MockBackend()` to dry-run without QGIS
`file`	`None`	path of the source `.niva` file — sets the base dir for relative paths / `call`
`log`	`NIVA_LOG`	journal base path → `<log>.jsonl` + `<log>.log`
`log_append`	`False`	`True` appends to the journal; `False` truncates
`progress`	`None`	`callable(str)` — a `▶ <stage>` line per stage plus 45% ticks
`cancel`	`None`	`callable() -> bool` — return `True` to abort a long step

```
import niva
layer = niva.flow('load "data.gpkg|layername=roads" | buffer 100m dissolve | save
 ↪  out.gpkg')
print(layer.ref)            # '/abs/path/out.gpkg'
```

`run_file(path, *, backend=None)`

Read a `.niva` file and execute it (sets `file=path` so relative paths resolve).

```
import niva
niva.run_file("myflow.niva")
```

`describe(name)`

Return a formatted description string for a verb or an algorithm id (see §6). Pure for a verb; needs QGIS for an algorithm id.

Importing `niva` is safe on any interpreter — QGIS is imported lazily only when a real run executes. To run for real outside QGIS you need QGIS's Python on `PYTHONPATH`; see the User Guide for the exact recipe.

niva Cookbook

Over a hundred worked recipes, from a single transform to full pipelines — including a large block of spatial **SQL** for both **SpatiaLite** and **PostGIS**, data **discovery** with `find` and `catalog`, plus a closing tour that reaches **every QGIS provider** (GDAL, GRASS, QGIS, PDAL, native, 3D) through the `run` escape hatch. Pair this with the Reference for exact signatures and the User Guide for setup.

Conventions

- Replace file paths, layer names, CRS codes, and SQL table/column names with your own.
- A multi-layer GeoPackage layer is addressed as `"file.gpkg|layername=<layer>"`.
- `@name` is a **saved QGIS database connection** (set up in the QGIS Browser or the plugin Setup tab) — SpatiaLite or PostGIS. Credentials live in QGIS, never in the flow.
- SQL column names below (`geom`, `geometry`, `id`, …) are illustrative — match your schema.
- Run any recipe with `niva "…"`, `niva run file.niva`, or the plugin Flow tab. Add `--dry-run` to validate without touching data.

A. First steps — one transform

1. Buffer a layer

```
load roads.gpkg | buffer 100m | save roads_buf.gpkg
```

Distances take a unit (`m`, `km`, `ft`, …); with no unit they're in the layer's CRS units.

2. Buffer and merge overlaps

```
load wells.gpkg | buffer 1km dissolve | save well_zones.gpkg
```

`dissolve` merges the overlapping buffers into one feature.

3. Reproject to another CRS

```
load parcels.shp | reproject EPSG:2262 | save parcels_spcs.gpkg
```

4. Clip to an area of interest

```
load buildings.gpkg | clip aoi.gpkg | save buildings_aoi.gpkg
```

5. Keep only matching features (attribute filter)

```
load parcels.gpkg | filter "zoning = 'R1'" | save residential.gpkg
```

The expression is a QGIS expression — quote it.

6. Repair invalid geometries

```
load messy.shp | fixgeom | save clean.gpkg
```

7. Reduce a layer to centroids

```
load parcels.gpkg | centroid | save parcel_points.gpkg
```

B. Multi-step chains

8. Filter → reproject → buffer

```
load roads.gpkg | filter "class = 'primary'" | reproject EPSG:2262 | buffer 100m dissolve |
↪  save corridors.gpkg
```

9. Reproject → fixgeom → buffer with options

```
load buildings.gpkg | reproject EPSG:26918 | fixgeom | buffer 15m dissolve cap=flat
↪  segments=12 | save footprints.gpkg
```

10. Clip → fixgeom → dissolve

```
load parcels.gpkg | clip township.gpkg | fixgeom | dissolve zoning | save
↪  zoning_blocks.gpkg
```

11. Simplify then split to single parts

```
load streets.gpkg | reproject EPSG:26918 | simplify 5m method=area | explode | save
↪  street_segments.gpkg
```

12. Stamp metadata and assess in one pass

```
load wells.gpkg | metadata set title="Monitoring wells" keywords="wells,gw" | assess deep
↪  to docs/wells_quality.md | save wells.gpkg
```

`metadata` and `assess` are pass-through, so the chain still ends at `save`.

C. Attributes, fields & joins

13. Keep only the fields you need

```
load parcels.gpkg | keepfields owner,zoning,area | save parcels_slim.gpkg
```

14. Drop fields / rename a field

```
load parcels.gpkg | dropfields temp1,temp2 | renamefield zoning zone_code | save
↪  parcels.gpkg
```

15. Attribute join from a table

```
load homes.gpkg | join with=census.csv field=tract field2=GEOID fields=pop,income
↪  prefix=cen_ | save homes_enriched.gpkg
```

16. Join, keeping only matches

```
load homes.gpkg | join with=owners.gpkg field=parcel_id field2=pid discard | save
↪  owned_homes.gpkg
```

`discard` drops input rows that found no match.

17. Count points inside each polygon

```
run native:countpointsinpolygon POLYGONS=parcels.gpkg POINTS=trees.gpkg FIELD=n_trees |
↪  save parcels_treecount.gpkg
```

`countpoints` takes both layers as options, so call it without a piped input.

D. Overlay & spatial selection

18. Intersection of two layers

```
load parcels.gpkg | intersect floodzone.gpkg | save parcels_in_flood.gpkg
```

19. Difference (erase)

```
load parcels.gpkg | fixgeom | difference buildings.gpkg | save open_land.gpkg
```

20. Select by location (spatial filter)

```
load buildings.gpkg | selectloc floodzone.gpkg predicate=intersect,within | save
↪  at_risk.gpkg
```

21. Spatial join — attach attributes by location

```
load incidents.gpkg | spatialjoin with=districts.gpkg predicate=within method=first
↪  fields=district_name | save incidents_tagged.gpkg
```

22. Zonal statistics (raster × polygons)

```
load watersheds.gpkg | zonalstats raster=dem.tif stats=mean,min,max prefix=elev_ | save
↪  watersheds_elev.gpkg
```

E. Batch processing with `each`

23. Reproject every file in a folder into one GeoPackage

```
each "in/*.shp" | reproject EPSG:6346 | save out.gpkg
```

Each input becomes its own layer inside `out.gpkg` (named after the file).

24. Process every layer of a multi-layer GeoPackage

```
each "basemap.gpkg" | fixgeom | clip aoi.gpkg | save basemap_clip.gpkg
```

25. Recurse a directory tree

```
each "data/" | reproject EPSG:6346 | save collected.gpkg
```

26. Per-item output files with `{name}`

```
each "rasters/*.tif" | warp EPSG:6346 | save "warped/{name}.tif"
```

27. Batch-load a folder into a PostGIS schema (one table per file)

```
each "deliverables/" | reproject EPSG:6346 | save @pg.staging
```

A trailing qualifier on a batch DB save is the **schema** (`staging`); each layer becomes a table there.

28. Filter the batch — only the datasets that match

```
each "data/**/*.gpkg" geom=polygon minfeatures=1 | dissolve | save merged.gpkg
each "tiles/*.tif" newerthan=7d | hillshade | save "out/{name}.tif"
```

`each` takes the same filters as `niva find` (flat `option=value`): offline `ext` / `minsize` / `maxsize` / `newerthan` / `format`, and — on QGIS's Python — `geom` / `crs` / `minfeatures` / `maxfeatures` / `has-field`.

29. Pipe `find` into other tools (non-niva)

```
niva find "*.gpkg" in ~/data --paths > inventory.txt      # a plain file list
niva find "*.tif" in ~/data --paths | wc -l               # count them
niva find "*.shp" in ~/data -0 | xargs -0 -n1 ogrinfo -so # feed any CLI, spaces-safe
```

`--paths` prints just the absolute paths (one per line, nothing else); `-0` NUL-separates them for `xargs -0`. Use `--as-flow` instead when you want a runnable `each …` skeleton.

F. Raster & terrain

30. Reproject (warp) a raster

```
load dem.tif | warp EPSG:6346 | save dem_utm.tif
```

31. Warp and resample to a target pixel size

```
load ortho.tif | warp EPSG:6346 resolution=5 resampling=average | save ortho_5m.tif
```

`average` is the right resampler when down-sampling imagery.

32. Clip a raster to a mask layer

```
load dem.tif | clipraster aoi.gpkg | save dem_aoi.tif
```

33. Hillshade from a DEM

```
load dem.tif | hillshade z_factor=2 azimuth=315 altitude=45 | save hillshade.tif
```

34. Slope (in percent) and aspect

```
load dem.tif | slope percent | save slope_pct.tif
load dem.tif | aspect | save aspect.tif
```

35. Vectorise a classified raster

```
load landcover.tif | polygonize field=class | save landcover_zones.gpkg
```

G. Spatial SQL in SpatiaLite

These use a saved **SpatiaLite** connection `@sl`. A `SELECT`/`WITH` returns a layer you can pipe; anything else runs in the database. Adjust table/column/geometry names to your schema (SpatiaLite geometry columns are often `geometry` or `geom`).

36. Read a filtered subset as a layer

```
sql @sl "SELECT * FROM parcels WHERE zoning = 'R1'" | save residential.gpkg
```

37. Filter by a spatial measure

```
sql @sl "SELECT id, zoning, geometry FROM parcels WHERE ST_Area(geometry) > 2000" | save
↪  big_parcels.gpkg
```

38. Compute geometry server-side, then keep processing in niva

```
sql @sl "SELECT id, ST_Buffer(geometry, 50) AS geometry FROM wells" | reproject EPSG:6346 |
↪  save well_buffers.gpkg
```

39. Spatial join — points within polygons

```
sql @sl "SELECT p.id, z.zone, p.geometry FROM points p JOIN zones z ON
↪  ST_Within(p.geometry, z.geometry)" | save points_zoned.gpkg
```

40. Aggregate to an attribute table

```
sql @sl "SELECT zoning, COUNT(*) AS n, SUM(ST_Area(geometry)) AS total_area FROM parcels
↪  GROUP BY zoning" | save zoning_summary.gpkg
```

41. Dissolve by attribute with `ST_Union`

```
sql @sl "SELECT zoning, ST_Union(geometry) AS geometry FROM parcels GROUP BY zoning" | save
↪  zoning_dissolved.gpkg
```

42. CTE, then hand off to a niva verb

```
sql @sl "WITH big AS (SELECT * FROM parcels WHERE ST_Area(geometry) > 5000) SELECT * FROM
 ↪  big" | centroid | save big_centroids.gpkg
```

H. Spatial SQL in PostGIS

These use a saved **PostGIS** connection `@pg`. Reads (`SELECT`/`WITH`) return a pipeable layer; writes (`CREATE`/`UPDATE`/`INSERT`/`CREATE INDEX`/…) run server-side and end the flow.

43. Read a table (schema-qualified)

```
load @pg.public.roads | save roads_local.gpkg
```

Equivalent read via SQL: `sql @pg "SELECT * FROM public.roads" | save roads_local.gpkg`.

44. Filter in the database, finish in niva

```
sql @pg "SELECT * FROM roads WHERE class = 'primary'" | buffer 50m dissolve | save
 ↪  primary_corridors.gpkg
```

45. Do the spatial work in SQL (the server-side lever)

```
sql @pg "SELECT id, ST_Buffer(geom, 100) AS geom FROM homes WHERE has_cat AND NOT has_dog"
 ↪  | save targets.gpkg
```

46. Cross-table spatial join

```
sql @pg "SELECT a.id, a.geom FROM parcels a JOIN flood b ON ST_Intersects(a.geom, b.geom)"
 ↪  | save parcels_at_risk.gpkg
```

47. Write a niva result back into PostGIS

```
load parcels.gpkg | clip aoi.gpkg | save @pg.public.parcels_clip mode=replace
```

`mode=create` (default) refuses to overwrite; `replace` drops + recreates; `append` inserts.

48. Analyse-in-place: materialise a derived table

```
sql @pg "CREATE TABLE roads_buf AS SELECT id, ST_Buffer(geom, 100) AS geom FROM roads"
```

A leading `CREATE` routes server-side (terminal) — even `CREATE TABLE … AS SELECT …`.

49. Maintenance — update a column and add a spatial index

```
sql @pg "UPDATE parcels SET area_m2 = ST_Area(geom)"
sql @pg "CREATE INDEX ON roads_buf USING GIST (geom)"
```

I. Projects & templates

50. Repoint a QGIS project to clipped data

```
project "city.qgs" to="out/city_aoi.qgs" repoint="out/clipped.gpkg" missing=keep
```

Copies the project and points each layer at the same-named layer in `clipped.gpkg`.

51. Build a fresh project from a folder of outputs

```
project new from="out/" to="deliverable.qgz" crs=EPSG:6346 title="Deliverable"
```

52. Instantiate a styled, laid-out template against your data

```
project from-template=example to="report.qgz" data="mydata/"
```

The bundled `example` template (boundary/roads/places slots + print layout) repoints to your same-named data, symbology and layout riding along. Register your own designed project with `project to-template=<name> from="MyMap.qgz" paths=relative`, then call it by name. See Template projects.

K. Reaching every provider with `run`

niva gives 45 algorithms friendly verbs; the other ~830 — across **every** Processing provider (native, gdal, grass, qgis, pdal, otb, 3d — **878** total) — are reachable with `run <id> KEY=value …`. A few per provider below. Find any algorithm's parameters with `niva describe <id>` or the algorithm appendix.

Beyond the QGIS providers, niva's **native-CLI harness** adds two id families that shell out to a tool directly (see §L): `pdalcli:<command>` (PDAL on **raw LAS/LAZ/COPC**, no COPC step) and `saga:<library>:<tool>` (`saga_cmd`).

Two things to know when using `run` directly:

- **Enum options take their integer index**, not the alias word — `format=0`, not `format=degrees`. (The index list is in each algorithm's appendix entry.)
- **Some providers (GRASS, PDAL) write *named* outputs** (`output=`, `slope=`, …) instead of OUTPUT, so the step is terminal — give the output path directly rather than piping to `save`. GRASS and PDAL also need their backends installed.

GDAL (`gdal:`) — raster/vector via GDAL/OGR

53. Contour lines from a DEM

```
load dem.tif | run gdal:contour INTERVAL=10 FIELD_NAME=ELEV | save contours.gpkg
```

54. Rasterize a vector field (10-unit pixels)

```
load parcels.gpkg | run gdal:rasterize FIELD=value UNITS=1 WIDTH=10 HEIGHT=10 | save
↪  parcels_value.tif
```

55. Proximity (distance-to-target) raster

```
load targets.tif | run gdal:proximity UNITS=0 MAX_DISTANCE=500 | save distance.tif
```

56. Fill small NoData gaps by interpolation

```
load gappy.tif | run gdal:fillnodata DISTANCE=20 | save filled.tif
```

GRASS (`grass:`) — named outputs (terminal), enums as integers

57. Slope and aspect from a DEM

```
run grass:r.slope.aspect elevation=dem.tif format=0 slope=slope.tif aspect=aspect.tif
```

58. Watershed flow accumulation and basins

```
run grass:r.watershed elevation=dem.tif threshold=1000 accumulation=flowacc.tif
↪  basin=basins.tif
```

59. Least-cost surface from a start point

```
run grass:r.cost input=cost_surface.tif start_coordinates=600100,4800100
↪  output=cumulative_cost.tif
```

60. Viewshed from an observer

```
run grass:r.viewshed input=dem.tif coordinates=600100,4800100 observer_elevation=1.75
 ↪  max_distance=5000 output=visible.tif
```

QGIS (`qgis:`) — the QGIS-Python toolbox

61. Virtual-layer SQL across files (`input1`, `input2`, …)

```
run qgis:executesql INPUT_DATASOURCES="roads.gpkg;parcels.gpkg" INPUT_QUERY="SELECT * FROM
 ↪  input1 WHERE class = 'primary'" | save primary.gpkg
```

This is the file-backed SQL form (the `sql` verb itself only queries `@conn` databases).

62. Concave hull (k-nearest neighbour)

```
load points.gpkg | run qgis:knearestconcavehull KNEIGHBORS=5 | save hull.gpkg
```

63. Random sample points inside polygons

```
load tracts.gpkg | run qgis:randompointsinsidepolygons STRATEGY=0 VALUE=100
 ↪  MIN_DISTANCE=50 | save sample.gpkg
```

64. Spread overlapping (stacked) points apart

```
load stacked.gpkg | run qgis:pointsdisplacement PROXIMITY=5 DISTANCE=10 HORIZONTAL=false |
 ↪  save displaced.gpkg
```

PDAL (`pdal:`) — point clouds (use `run`; load/save are vector/raster)

> The QGIS `pdal:` provider loads point clouds as **COPC/EPT** layers (raw `.las` may need a COPC index first). For **raw LAS/LAZ with no COPC step** — and classification-aware LiDAR workflows — use niva's `pdalcli:` harness in **§L** below.

65. Export a DEM from a LiDAR cloud, then hillshade it

```
run pdal:exportraster INPUT=lidar.copc.laz ATTRIBUTE=Z RESOLUTION=1 | hillshade z_factor=2
 ↪  | save lidar_hillshade.tif
```

`exportraster` outputs a raster (`OUTPUT`), so it pipes into niva's raster verbs.

66. Classify ground returns

```
run pdal:classifyground INPUT=lidar.laz OUTPUT=ground_classified.laz
```

67. Extract the cloud's data-boundary polygon

```
run pdal:boundary INPUT=lidar.laz RESOLUTION=10 THRESHOLD=10 OUTPUT=cloud_extent.gpkg
```

68. Thin a dense cloud by sampling radius

```
run pdal:thinbyradius INPUT=lidar.laz SAMPLING_RADIUS=2 OUTPUT=thinned.laz
```

Native (`native:`) — beyond the alias verbs

69. Join the nearest feature from another layer

```
load schools.gpkg | run native:joinbynearest INPUT_2=parcels.gpkg NEIGHBORS=1
 ↪  MAX_DISTANCE=500 | save schools_parcel.gpkg
```

70. DBSCAN point clustering

```
load incidents.gpkg | run native:dbscanclustering MIN_SIZE=5 EPS=250 | save
 ↪  incident_clusters.gpkg
```

71. Shortest path over a road network

```
load roads.gpkg | run native:shortestpathpointtopoint STRATEGY=0
 ↪  START_POINT="600100,4800100" END_POINT="601000,4801000" | save route.gpkg
```

72. Hub-and-spoke (spider) lines

```
run native:hublines HUBS=depots.gpkg HUB_FIELD=id SPOKES=stops.gpkg SPOKE_FIELD=depot_id |
↪  save spider.gpkg
```

3D (`3d:`)

73. Tessellate polygons into 3D geometry

```
load buildings.gpkg | run 3d:tessellate | save buildings_3d.gpkg
```

The `3d:` provider ships a single algorithm in QGIS 4.0.3 — `tessellate` — so this is the whole provider.

L. LiDAR & point clouds

Point clouds are first-class: each/show/catalog recognise `.las`/`.laz`/`.copc.laz`/`.vpc`/ `.e57`/... (so a folder of tiles batches), and the friendly verbs **`dtm`** / **`dsm`** / **`hag`** wrap niva's `pdalcli:` harness (pdal_wrench, reading **raw LAS** — no COPC step). Needs the point-cloud backend (`niva pdal check`; see the PDAL/LAStools guide). The raw `run pdalcli:<command>` escape hatch is still there for everything the verbs don't cover.

74. DTM — bare-earth terrain from GROUND returns (class 2)

```
load "tile.las" | dtm resolution=1 | save dtm.tif
```

`dtm` = `run pdalcli:to_raster attribute=Z filter="Classification==2"` — just friendlier.

75. DSM — top surface from all returns

```
load "tile.las" | dsm resolution=1 | save dsm.tif
```

76. CHM — canopy height, the clean way (`hag` normalises, `dsm` rasterises)

```
load "tile.las" | hag | dsm resolution=1 | save chm.tif
```

hag sets each point's height-above-ground; `dsm` then gives the max height per cell = canopy height. (Alternative: DSM − DTM via `run grass:r.mapcalc.simple expression="A-B" a=dsm.tif b=dtm.tif output=chm.tif`.)

77. Batch a whole folder of tiles into DTMs

```
each "~/lidar/tiles/*.las" | dtm resolution=1 | save "~/lidar/dtm/{name}.tif"
```

One DTM per tile, named after the tile. Add filters: `each "*.las" minsize=50M | dtm | save …`.

78. Batch → single mosaic DTM (rasterise per tile, then merge)

```
each "~/lidar/tiles/*.las" | dtm resolution=1 | save "~/lidar/dtm/{name}.tif"
run gdal:merge INPUT="~/lidar/dtm/*.tif" OUTPUT="~/lidar/dtm_mosaic.tif"
```

Per-tile rasterising is parallel-friendly and avoids one giant merged cloud. (Cloud-level merge: `run pdalcli:merge files="a.las;b.las" output=merged.laz`, then `load "merged.laz" | dtm …`.)

79. `$VAR` in output paths (portable studies)

```
export LIDAR=~/lidar
niva 'each "$LIDAR/tiles/*.las" | dtm resolution=1 | save "$LIDAR/dtm/{name}.tif"'
```

$VAR/${VAR} expand in path values (in `.niva` files and the repl too, not just the shell) — paths only, so a `filter "$area > 100"` expression is untouched.

80. Raw-harness extras — extract a class, clip, classify, density

```
load "tile.las" | run pdalcli:translate filter="Classification==6" output=buildings.laz
 ↪  # class 6 = buildings
load "merged.laz" | run pdalcli:clip polygon=aoi.gpkg output=study.laz
load "tile.las" | run pdalcli:classify_ground output=classified.laz
load "tile.las" | run pdalcli:density resolution=1 | save point_density.tif
 ↪  # coverage QA
```

81. Inventory a LiDAR folder

```
show ~/lidar/tiles            # each tile: kind=pointcloud, point count, loadable source
catalog ~/lidar/tiles to=lidar_inventory.md
```

show/catalog report point clouds (and vector/raster/**mesh**) — discovery spans every data type QGIS reads.

M. Complex, value-added pipelines

Multi-provider chains that turn raw data into finished products. Each is verified end to end.

82. LiDAR → a full bare-earth terrain set (DTM once, then three derivatives)

```
load tile.las | run pdalcli:to_raster attribute=Z filter="Classification==2" resolution=1
 ↪  | save dtm.tif
load dtm.tif | hillshade z_factor=2 | save dtm_hillshade.tif
load dtm.tif | slope | save dtm_slope.tif
load dtm.tif | run gdal:contour BAND=1 INTERVAL=5 FIELD_NAME=elev OUTPUT=contours_5m.gpkg
```

83. Canopy height model → mean tree height per parcel (PDAL → GRASS → zonal stats)

```
load tile.las | run pdalcli:to_raster attribute=Z filter="Classification==2" resolution=1
 ↪  | save dtm.tif
load tile.las | run pdalcli:to_raster attribute=Z resolution=1 | save dsm.tif
run grass:r.mapcalc.simple expression="A-B" a=dsm.tif b=dtm.tif output=chm.tif
load parcels.gpkg | zonalstats raster=chm.tif prefix="canopy_" stats=mean | save
 ↪  parcels_canopy.gpkg
```

84. Hydrology from LiDAR, with GRASS — DTM → flow accumulation + direction + streams + basins, then vector catchments and vector streams. `r.watershed` writes several rasters at once (each is an output *param*), and run now creates the output folders for you.

```
load tile.las | run pdalcli:to_raster attribute=Z filter="Classification==2" resolution=1
 ↪  | save dtm.tif
run grass:r.watershed elevation=dtm.tif accumulation=hydro/flow_accum.tif
 ↪  drainage=hydro/flow_dir.tif stream=hydro/stream.tif basin=hydro/basin.tif
run grass:r.to.vect input=hydro/basin.tif type=2 output=catchments.gpkg
run grass:r.stream.extract elevation=dtm.tif accumulation=hydro/flow_accum.tif
 ↪  threshold=20000 stream_vector=streams.gpkg
```

85. Hydrology with SAGA — Strahler-ordered stream network + drainage basins (a complete run). GRASS's Strahler tool (`r.stream.order`) is an addon that isn't always installed; SAGA's channel network gives vector channels carrying an ORDER (Strahler) attribute plus vector basins in one tool. `saga:<library>:<tool>` reaches `saga_cmd` (set `NIVA_SAGA_CMD` if it isn't on PATH).

```
run saga:ta_preprocessor:4 ELEV=dtm.tif FILLED=hydro/filled.tif
run saga:ta_hydrology:0 ELEVATION=hydro/filled.tif FLOW=hydro/flow_accum.tif
run saga:ta_channels:5 DEM=hydro/filled.tif ORDER=hydro/strahler.tif
 ↪  SEGMENTS=streams_ordered.gpkg BASINS=basins.gpkg THRESHOLD=6
```

86. Landform classification — geomorphons (valleys, ridges, slopes, pits, peaks) from the DTM

```
load dtm.tif | run grass:r.geomorphon elevation=dtm.tif forms=landforms.tif search=15
```

87. Building footprint candidates — extract the building class, then its coverage polygons

```
load tile.las | run pdalcli:translate filter="Classification==6" output=buildings.laz
load buildings.laz | run pdalcli:boundary | fixgeom | simplify 0.5m | save
 ↪  building_footprints.gpkg
```

88. The simplest possible map — one layer, one line, no options; `figure` picks a sensible extent, size, and stretch so it just works for vector *or* raster

```
load dem.tif | figure dem.png
```

89. Push it — a full thematic map using every knob — themed primary layer, a raster hillshade plus two vector overlays, an OSM basemap, field labels, a borrowed extent, and print-scale output

```
load flood_zones.gpkg | style apply=flood.qml
  | figure flood_map.png layers="hillshade.tif;roads.gpkg;places.gpkg" basemap=osm
   ↪  labels=zone_name extent=study_area.gpkg size=2400x1600 dpi=200 bg="#eef3f7"
```

Draw order is top-down: `flood_zones` (styled) over the overlays over the basemap. `extent=` borrows another layer's bounds; `labels=` labels by an attribute; `dpi=200` sizes symbols/text for print. Being pass-through, `figure` can also snapshot a mid-pipe step: `… | figure step.png | save out.gpkg`.

`map` — composed cartographic layouts (tiny → extreme)

Where `figure` is a bare image, **`map`** builds a page **layout** (→ PDF/PNG/SVG) with a legend, scale bar, and north arrow **on by default** — a proper map with no template required. These six climb from one line to a full multi-layer plate.

90. Tiny — one layer, one line; a complete A4 map (legend + scale bar + north arrow)

```
load dem.tif | map dem.pdf
```

91. Add a title

```
load parcels.gpkg | map parcels.pdf title="Parcels — 2026"
```

92. Label it, pick a page and format — PNG on US Letter, features labelled by a field

```
load zones.gpkg | map zones.png title="Zoning" labels=zone_type page=Letter dpi=200
```

93. Themed, with an overlay and a basemap — styled primary layer over roads over OSM tiles

```
load flood.gpkg | style apply=flood.qml | map flood.pdf title="Flood Risk"
 ↪  layers="roads.gpkg" basemap=osm labels=risk portrait
```

94. Many layers, many types — line, two rasters, polygon, and point layers on one A3 plate

```
load contours.gpkg | map terrain.pdf title="Terrain — Multi-Layer"
 ↪  layers="dtm.tif;dsm.tif;building_footprints.gpkg;control_points.gpkg" labels=elev
 ↪  extent=dsm.tif page=A3 landscape dpi=300
```

95. Extreme — build every derivative from raw LiDAR, then compose one rich A3 plate that stacks a hillshade, contours, footprints and markers over a basemap, framed to a study area at print DPI

```
load tile.las | run pdalcli:to_raster attribute=Z filter="Classification==2" resolution=1
 ↪  | save dtm.tif
load tile.las | run pdalcli:to_raster attribute=Z resolution=1 | save dsm.tif
load dtm.tif  | hillshade z_factor=2 | save hillshade.tif
load dtm.tif  | run gdal:contour BAND=1 INTERVAL=5 FIELD_NAME=elev OUTPUT=contours.gpkg
load tile.las | run pdalcli:translate filter="Classification==6" output=buildings.laz
load buildings.laz | run pdalcli:boundary | fixgeom | simplify 0.5m | save footprints.gpkg
load contours.gpkg | style apply=contours.qml
```

```
| map plate.pdf title="Bare-Earth Terrain & Structures"
↪ layers="hillshade.tif;dsm.tif;footprints.gpkg;control_points.gpkg" basemap=osm
↪ labels=elev extent=study_area.gpkg page=A3 landscape dpi=300 legend scalebar
↪ northarrow
```

For a fully hand-designed plate (custom frames, insets, an atlas of per-feature pages), design it once in QGIS and export it verbatim: `load aoi.gpkg | map out.pdf from=study.qgz layout="0verview"`.

N. Data discovery & inventory — `find` and `catalog`

`find` locates data on the filesystem; `catalog` profiles it into a Markdown report. Both are terminal (they report, they don't transform), and every recipe here is runnable as shown.

`find` — produce output for various uses

96. A plain file list (redirect to a file)

```
niva find "*.gpkg" in ~/data --paths > inventory.txt
```

`--paths` prints just absolute paths, one per line, nothing else — ideal for a manifest.

97. Count matches

```
niva find "*.tif" in ~/data --paths | wc -l
```

98. Feed any external tool, spaces-safe

```
niva find "*.shp" in ~/data -0 | xargs -0 -n1 ogrinfo -so
```

`-0` (alias `--print0`) NUL-separates paths so `xargs -0` handles names with spaces/newlines.

99. Machine-readable records for scripts / LLMs (`--json` + `jq`)

```
niva find "*.gpkg" in ~/data --json | jq -r '.[] | select(.features > 1000) | .path'
```

Each record carries `path`, `format`, `kind`, `size`, and — on QGIS's Python — `geometry`, `crs`, `features`, `fields`, `fid_column`.

100. Only polygon layers with data (GDAL filters), as paths

```
niva find "*.gpkg" in ~/data --geom Polygon --min-features 1 --paths
```

101. Turn a search into a runnable batch (`--as-flow`)

```
niva find "*.tif" in ~/rasters --as-flow > batch.txt
```

Emits one each `"<path>" | <stages> | save …` line per match — fill in the stages and run.

`catalog` — inventory to a report

102. Inventory a directory tree

```
niva catalog "data/" to=reports/inventory.md
```

Recurses `data/`, profiling every dataset (type, CRS, geometry/bands, feature count, extent); multi-layer containers expand per layer. Omit `to=` to write `data/catalog.md`.

103. Deep profile — add data-quality checks

```
niva catalog "data/" deep to=reports/inventory_deep.md
```

deep adds per-layer invalid/empty/duplicate-geometry counts and per-field null counts.

104. Catalog one container

```
niva catalog "basemap.gpkg" to=reports/basemap.md
```

105. Catalog a database connection

```
niva catalog @gisdb3 to=reports/db.md
```

`catalog` accepts any source `show` does — a file/container, a directory, an `@conn` database (optionally `@conn.schema`), or a remote OWS service.

Capstone — full pipelines

Two complete, runnable flows ship in `examples/`:

- **`analyst_plan.niva`** — catalog data sources → reproject/warp everything to a target CRS → clip to a study-area bounding box → repoint QGIS projects to the clips. A whole regional compilation, batched end to end.
- **`youngstown_cat_canvassing.niva`** — a multi-step analysis: assess → reproject → clip → geocode → server-side SQL selection → terrain routing → atlas export.

Run one with `niva run examples/analyst_plan.niva` (set `NIVA_TMPDIR` to a roomy disk first for the raster steps — see the User Guide).

Template projects

A **niva template is just a saved QGIS project.** There is no special format: you design a project in QGIS — print layout, styled layers, bookmarks, the lot — and niva reuses it against fresh data. `project from-template=` copies the template and **repoints each layer to your same-named data**, so everything you designed rides along.

```
# instantiate a template against your data
project from-template="my_basemap.qgz" to="acme.qgz" data="acme/clips/"

# …or use the bundled example, by name
project from-template=example to="acme.qgz" data="acme/clips/"
```

This page explains **what a template can carry**, **how slots are matched**, and **how to author a good one** — including the bundled `example` you can open in QGIS as a starting point.

How instantiation works

`project from-template=<name|path> to=<out> data=<dir|glob> [missing=keep|fail|drop]`

1. niva **copies the whole template project** — so every project-level thing you designed (layouts, bookmarks, themes, layer tree, title, CRS) is preserved verbatim.
2. For each layer (a **slot**), niva **repoints its datasource** to the dataset in `data=` whose name matches the slot, **preserving the layer's symbology and subset filter**.
3. The result is written to `to=` (`.qgs` or `.qgz` by extension). Nothing else changes.

`data=` is resolved like each / `project new`: a **directory** (recursed), a **glob**, or a **multi-layer container** (a GeoPackage, expanded per layer). Each dataset's **name** is its file stem (`parcels.gpkg` → `parcels`) or its `|layername=` inside a container.

Slot matching — by display name

A slot's identity is the **layer's display name** — the label in the QGIS *Layers* panel — falling back to the datasource name (`|layername=`, else file stem) when the display name isn't found. So a layer shown as **`parcels`** is filled by **`parcels.gpkg`** in `data=`, *regardless of the placeholder data the template currently points at*.

> **Author tip:** name your template's layers exactly what you'll name the matching data files. Rename a layer in the *Layers* panel (double-click) to set its slot name.

Slots with no match in `data=` follow `missing=`:

`missing=`	**Unmatched slot behaviour**
keep *(default)*	left pointing at the template's example data — **preserves layout/legend structure**
`drop`	removed from the project

missing=	Unmatched slot behaviour
fail	error (never silently produce a half-filled map)

What a template can carry

Because `from-template` copies the project and only swaps datasources, **everything in the project file rides along.** Author it once; it all comes through.

Project-level — carried verbatim

Element	Carried?	Notes
Print layouts	✓	All items: map frames, titles/labels, **legend**, **scale bar**, pictures / north arrows, attribute tables, HTML frames, and **atlas** configuration.
Spatial bookmarks	✓	Jump-to extents (see `project … bookmark=`).
Map themes	✓	Visibility / style presets.
Layer tree	✓	Group structure, layer order, visibility, expanded state.
Project title & CRS	✓	
Project metadata, colors, variables	✓	Project color scheme, custom project variables.

Per-layer (per-slot) — carried, datasource repointed

Element	Carried?	Notes
Symbology / renderer	✓	The *style* — single/categorized/graduated/rule-based, including data-defined overrides. Rides along onto the new data.
Display name	✓	This is the **slot key**.
Opacity & blend mode	✓	
Labels	✓	Label placement / expressions.
Subset / filter string	✓	Re-applied to the **new** data (so the filter's fields must exist there — see caveats).
Layer metadata	✓	
Joins, diagrams, actions, form/field config	✓	Carried as layer properties; schema-dependent ones assume matching fields.
Datasource	↻ **repointed**	To the same-named dataset in `data=` (vector → `ogr`, raster → `gdal`).

Not carried / unchanged

- **Layers that aren't vector or raster** are left untouched.
- **Slots with no matching data** follow `missing=` (above).
- `from-template` does **not** rewrite datasource **path storage** — that's a `to-template` option (`paths=relative`).

Caveats when authoring

- **Schema-dependent styling.** A categorized/graduated/rule-based renderer, a subset filter, a join, or a label expression references **field names / values**. The style rides along, but it only *renders* correctly if your data has those fields. Keep slot schemas consistent, or use simple symbology in the template.

- **Layout map extent.** A layout map frame stores a fixed extent (the template's example area). After instantiation it still shows that extent — which may not cover your data. Pan the layout map to your area, or add a **bookmark** at your AOI and use it. (A future option may auto-fit the layout map to the new data.)
- **Geometry type.** Symbology assumes a geometry type — fill the `boundary` (polygon) slot with polygon data, `roads` (line) with lines, etc. Mismatched geometry still repoints but the symbol won't apply meaningfully.

Authoring a template, end to end

1. **Design in QGIS.** Add your layers, style them, build a print layout, set bookmarks / themes / title / CRS. Save as `.qgz`.
2. **Name the slots.** Rename each layer in the *Layers* panel to the name you'll give the matching data file (e.g. `parcels`, `roads`, `boundary`).
3. *(Optional)* **Make it portable.** Register it into your library with relative paths:
   `project to-template=parcel_map from="MyParcelMap.qgz" paths=relative`
   `to-template=<name>` copies the project into the **template library** — `$NIVA_TEMPLATES` if set, else `~/.niva/templates` — so `from-template=<name>` finds it. A **path** value (`to-template="out/tmpl.qgz"`) writes anywhere instead.
4. **Instantiate** against any dataset whose layers are named for your slots:
   `project from-template=parcel_map to="acme.qgz" data="acme/parcels/"`

Where named templates resolve

`from-template=<name>` (a bare name, no path) is looked up in order:

1. `$NIVA_TEMPLATES` (if set)
2. `~/.niva/templates` (your personal library)
3. the **bundled** templates that ship inside niva (e.g. `example`)

An earlier hit shadows a later one, so a personal `example.qgz` overrides the bundled one. An unknown name errors with the available names listed.

The bundled `example` template

niva ships a fully-populated `example` template (under `niva/templates/`, regenerated by `templates/build_example.py`) so you can see exactly what a good template contains. It has:

- **Three styled slots:** `boundary` (polygon outline), `roads` (line), `places` (point) — each with distinct symbology.
- **A print layout** (`Map`): title, map frame, legend, scale bar.
- **A spatial bookmark** (`Study Area`) and a project **title** + **CRS**.

Open `niva/templates/example.qgz` in QGIS to inspect it, or instantiate it directly:

```
project from-template=example to="my_map.qgz" data="my_data/"
```

where `my_data/` holds `boundary.*`, `roads.*`, and `places.*` (any OGR vector format). The slots repoint to your data; the layout, legend, scale bar, bookmark, title, and CRS all ride along. Copy it into `~/.niva/templates/` and edit it to make your own.

FAQ

Short answers to common questions. See the User Guide for setup, the Reference for the full surface, and the Cookbook for recipes.

What software libraries are needed to use niva?

Almost nothing beyond **QGIS** itself. niva is **pure Python with zero third-party runtime dependencies** — it runs on QGIS's own bundled Python.

- **Required: QGIS 3.22+ (or 4.x)** with its Processing framework. QGIS already ships everything niva relies on — GDAL/OGR, PROJ, GEOS, and the `native:`, `qgis:`, and `gdal:` algorithm providers. There is **no `pip install`** for the plugin (it bundles niva), and even the standalone package installs into QGIS's Python with no extra dependencies.
- **Optional providers** (only if you call those algorithms via `run`): **GRASS GIS** for the `grass:` algorithms and **PDAL** for the `pdal:` point-cloud algorithms. Install these the usual way for your platform; niva just calls them through QGIS Processing.
- **Databases:** handled entirely by QGIS's own providers — **PostgreSQL/PostGIS** (libpq) and **SpatiaLite** (mod_spatialite), both bundled with QGIS. You configure a connection in QGIS; niva references it as `@conn`.
- **`notify` / `email`:** Python standard library only (`urllib`, `smtplib`) — no extra libs.

The only thing that needs extra tools is **building the guide as a PDF** (`scripts/build_guide_pdf.py`): that needs **pandoc** + a LaTeX engine (e.g. `xelatex`). Those are *not* needed to use niva.

Do I need to know Python or PyQGIS?

No. That's the point — you write a readable one-line flow instead of PyQGIS code:

```
load roads.gpkg | buffer 100m dissolve | clip city.gpkg | save roads_local.gpkg
```

If you *want* the Python, niva can **export any flow to a standalone PyQGIS script** (`niva export`, or the plugin's Convert tab).

How do I run niva — in QGIS or from the command line?

Both, identically:

- **In QGIS** — install the plugin, open its dock, type a flow, hit **Run**. No pip step.
- **Standalone** — the `niva` CLI / Python API, run with QGIS's own Python (it boots QGIS headless). See the User Guide.

I typed `niva` in a terminal and nothing happened (or "command not found")

Two things have to be true for the `niva` command to work, and a fresh shell usually has neither:

1. **There's a `niva` command on your `PATH`.** niva isn't installed as a command until you either `pip install` it into a Python (which creates the `niva` script) or alias it (below).
2. **It runs on QGIS's Python.** niva needs the PyQGIS bindings (`qgis.core`, Processing) — your system `python3` usually can't import them, only the Python that QGIS ships with.

The quickest fix — **no install, just an alias**. **Two parts must point at *your* system — don't paste the line as-is:**

- replace `/path/to/niva` with the directory that contains the `niva/` package (the repo root), and
- use the **full path to QGIS's Python**, not a bare `python3`. A bare `python3` often resolves to a Homebrew / pyenv / conda Python that *can't* import QGIS — you'll get `No module named 'niva'` or a PyQGIS `ImportError`.

```
# Template — substitute both placeholders:
alias niva='PYTHONPATH=/path/to/niva:/usr/share/qgis/python:/usr/lib/python3/dist-packages
 ↪  QT_QPA_PLATFORM=offscreen /path/to/qgis/python3 -m niva.cli.main'

# Concrete example (Linux, distro QGIS, repo at ~/Github/niva):
alias niva='PYTHONPATH=/home/jcz/Github/niva:/usr/share/qgis/python:/usr/lib/python3/dist-
 ↪  packages QT_QPA_PLATFORM=offscreen /usr/bin/python3.14 -m niva.cli.main'
niva describe buffer            # now it works
```

To find QGIS's Python + paths, open the QGIS **Python Console** and run: `import sys, qgis; print(sys.executable); print([p for p in sys.path if 'qgis' in p.lower()])` — use that `sys.executable` as the interpreter above. Then `niva info` confirms it's wired to your real QGIS profile (it lists your `@conn` connections). Or install the command into QGIS's Python — full instructions in the User Guide. Sanity-check the parser without QGIS using `--explain`: `<qgis-python> -m niva.cli.main "load a.gpkg | buffer 100m | save b.gpkg" --explain`.

Is niva on PyPI? How do I install it?

Yes — as **`qgis-niva`** (the import package and the `niva` command stay `niva`, like `scikit-learn` → `sklearn`). Pick by what you want to do:

- **Run inside QGIS (GUI):** the **plugin zip** (`niva_qgis.zip`, from the latest release or `plugin/build_plugin.sh`) → *Install from ZIP*. Bundles niva; no pip step.
- **Execute from the terminal:** install into **QGIS's own Python** so `niva run` has PyQGIS — `<qgis-python> -m pip install qgis-niva` (see Quick start).
- **Author offline anywhere:** `uv tool install qgis-niva` (or `uvx --from qgis-niva niva …`) — the offline CLI, no QGIS required. This isolated install can't *execute* flows; see the next question.

(For a source/dev install: `<qgis-python> -m pip install git+https://github.com/johnzastrow/niva.git`.)

Does `uv tool install qgis-niva` connect to QGIS? Do I need the plugin?

Short version: **`uv tool install`** (and uvx, and pipx) put niva in their **own isolated Python that has no PyQGIS**, so you get the complete **offline** CLI but **not** execution. niva doesn't bundle or

auto-discover QGIS — *running* a flow imports `qgis`, which lives only in QGIS's own Python. So a uv `tool install qgis-niva` on a clean OS gives you:

- ✓ `validate`, `explain`, `search`, `setup`, `plan`, `manifest`, `repl`, `export` — author and check flows, no QGIS needed;
- ✗ `run`, `show @conn`, executing a one-liner for real — these need PyQGIS.

To actually run geoprocessing, add **one** of these (they're independent front-ends over the same niva, so you can keep the uv install for authoring and add either for execution):

You want…	**Do this**
Author / validate flows fast, anywhere	`uv tool install qgis-niva` — keep it for the offline CLI
Run flows in the QGIS GUI	Install the **plugin** (*Install from ZIP*) — it bundles niva and runs in QGIS's Python
Run flows from the terminal	Install into QGIS's Python — `<qgis-python> -m pip install qgis-niva` **or** `uv pip install --python <qgis-python> qgis-niva` — then alias `niva` to that interpreter (Quick start)

It's the install *mode*, not pip-vs-uv

`pip` and uv don't produce different *kinds* of niva — **the install mode does**, and both tools can do either mode. What decides whether niva can execute is always the same single question: **does the Python niva lands in have PyQGIS?**

Install mode	pip family	uv family	Can `niva run`?
Into a chosen interpreter	`<qgis-python> -m pip install qgis-niva`	`uv pip install --python <qgis-python> qgis-niva`	✓ when it's QGIS's Python
Isolated app venv	`pipx install qgis-niva`	`uv tool install qgis-niva` · `uvx …`	□ offline only

The common commands just pick opposite **defaults**: plain `pip install` goes *into* the interpreter you ran it with, while `uv tool` / `uvx` / `pipx` deliberately build a private venv — which is why they *look* like different kinds of install. Point either tool at QGIS's Python and you get full execution; let either build an isolated venv and you get the offline CLI.

Then what good is the offline install? It's niva's *compiler half*, and it earns its place exactly where a multi-GB QGIS is unwanted:

- **CI / pre-commit / editors** — `validate` + `explain` your `.niva` files in a pipeline or on save, catching errors long before they reach QGIS (no QGIS in the CI image).
- **Tooling & agents** — `manifest` + `plan` emit the machine-readable verb catalog and resolved plan IR that power editor completion, an LSP, and LLM agents.
- **Author / learn / scaffold anywhere** — `repl`, `search`, `setup`, or a one-off `uvx --from qgis-niva niva …` on any laptop.
- **`export`** — transpile a flow to a standalone PyQGIS script to run wherever QGIS lives.

Think **compiler front-end** (installs anywhere) vs **runtime** (needs the platform). And it won't stay execution-less forever: a planned executor bridge will let `niva run` shell out to a QGIS you already have installed (`qgis_process`), so even an isolated uv install could execute against a system QGIS.

So do you need the plugin? Only if you want to run *inside QGIS* — the dock's **Run** button or the QGIS Python console. If you'd rather drive execution from a shell, skip the plugin and `pip install qgis-niva` into <qgis-python> instead. Either way, note that **`uv tool install` does not install QGIS itself** — on a clean OS you still need QGIS on the machine before anything can execute, no matter which path you choose.

Which QGIS versions are supported?

QGIS **3.22+** (Qt5) and **QGIS 4.x** (Qt6). The plugin declares `qgisMinimumVersion=3.22`; development targets QGIS 4.0.3.

A raster job fails with ”disk quota exceeded” or ”No space left” even though I have free disk — why?

Raster steps write multi-gigabyte intermediates, and by default they land in the system temp dir — often a small RAM-backed `/tmp`. Point niva's scratch at a roomy disk-backed folder:

```
export NIVA_TMPDIR=$HOME/niva_scratch
```

(or set the **Raster scratch** field in the plugin's Setup tab). Scratch is purged on every run, even a failed one. See the User Guide.

How do I read and write a database?

Configure a PostGIS or SpatiaLite connection in QGIS, then reference it by name as `@conn`:

```
load @pg.public.roads | clip aoi.gpkg | save @pg.public.roads_clip mode=replace
sql @pg "SELECT id, ST_Buffer(geom,100) AS geom FROM homes WHERE has_cat" | save
 ↪  targets.gpkg
```

Only the connection **name** appears in a flow — credentials stay in QGIS. See the Reference.

How do I see what layers/tables are in a file or database?

Use `show` — it lists the loadable layers at one location with each one's kind, geometry/raster type, format, and a copy-pasteable `load` source:

```
show data.gpkg            # layers in a GeoPackage (or any container/shapefile/GeoTIFF)
show data/                # everything directly under a directory
show data/ deep           # recurse the whole tree
show @gisdb3              # all tables in a PostGIS connection
show @gisdb3.public       # just one schema
show "https://host/geoserver/wfs?service=WFS"   # remote WFS feature types
show "https://host/geoserver/ows?service=WMS"   # remote WMS layers
show "https://host/arcgis/rest/services/X/FeatureServer"  # ArcGIS REST layers
show "https://tile.osm.org/{z}/{x}/{y}.png"     # XYZ tile layer
```

The listing's footer shows two runnable examples built from the first row (e.g. `load "…|layername=roads" | buffer 100m | save out.gpkg`). Heads-up for the shell: the Source cells are wrapped in Markdown backticks — copy the value *inside* them, and quote the whole flow so your shell doesn't run the backticks or split on the `|`: `niva 'load "…|layername=roads"'`.

Directory listings are **shallow by default** — add the deep flag to recurse. Any QGIS-readable format is picked up (SpatiaLite, FileGDB, … — not a fixed extension list); dataset sidecars and non-geospatial files are skipped. It's the quick *"what can I load here?"* glance; for a deep per-dataset inventory (CRS, extent, fields, feature counts) use `catalog` instead (the two can't be piped together — both are terminal). For files you can take `show`'s source column straight to `ogrinfo <file> <layer>`.

How do I find my database connection names (the `@conn` values)?

Run `info`. From a shell — where the QGIS Browser isn't in front of you — this is the quickest way to see the registered PostGIS/SpatiaLite connections, plus your QGIS/GDAL/PROJ versions, the reachable algorithm count, and the niva environment variables (secrets masked):

```
niva info                 # prints the report; the "Database connections" section lists
↪  each @conn
niva info to=env.md       # or save it to a file
```

It's the CLI counterpart of the plugin's Setup-tab **Environment report**. See the Reference.

Standalone niva shows different connections than my QGIS — why?

Connections are **per QGIS user profile**. A standalone niva reads the **same profile your QGIS desktop last used**, so the @conn names match the GUI. (Earlier builds accidentally read a generic, empty Qt settings store and saw none of your connections — fixed in 0.28.0.) If you still see a mismatch, you're probably looking at a *different* profile than you think — run `niva info` and check the **QGIS profiles** section, which lists every profile and its connections.

Can niva use a connection from another QGIS profile?

Yes — niva uses **one** profile at a time. `niva info` lists all profiles and their connections; to use a connection that lives in another profile, point niva at it:

```
NIVA_QGIS_PROFILE=staging niva load @prod_db.public.roads | save roads.gpkg
```

(Inside QGIS, niva always uses the profile you have open.)

Can I use an algorithm that doesn't have a niva verb?

Yes. The ~45 alias verbs are conveniences; **every** QGIS algorithm (878 in QGIS 4.0.3) is reachable with `run <id> KEY=value …`. Discover one with `niva describe <id>`, or browse the algorithm appendix — each entry has a worked example.

How do I turn a niva flow into a normal PyQGIS script?

`niva export flow.niva -o flow.py` (or the plugin's Convert tab) writes a standalone PyQGIS script — one `processing.run(...)` per step. `niva import` reverses it for niva-shaped scripts.

Does niva send my data anywhere, or need an internet connection?

No. niva runs locally on QGIS Processing; it never transmits your data or credentials. The only outbound features are **opt-in**: `notify` (ntfy), `email` (SMTP), and any algorithm that is itself online (e.g. a geocoder you call via `run`). Their credentials come **only** from the environment, never from flow text.

How do I get the whole guide as one document?

Build a single PDF of this guide (and the full algorithm appendix) with:

```
python3 scripts/build_guide_pdf.py      # -> docs/guide/niva-guide.pdf
```

It's also attached to each release as `niva-guide.pdf`.

Editor / IDE integration

niva flows are plain `.niva` text files. This guide sets up **syntax highlighting** in whatever editor you use (the definitions live in `.vscode/niva/` — despite the folder name they cover far more than VS Code) — and, via the **language server**, real completion, diagnostics, and hover docs.

Language server — `niva lsp` (real completion, diagnostics, hover)

The syntax files below only *colour* a `.niva` file. **`niva lsp`** gives your editor the same intelligence the repl has, in any editor that speaks LSP:

- **completion** — verbs → their options/flags → enum values → **filesystem paths**;
- **diagnostics** — the offline validator's errors/warnings, live as you type (unknown verb, bad option/enum, a flow with no save), before you ever run;
- **hover** — `describe` docs for the verb under the cursor.

It's the *same* engine as the repl (`niva.intelligence`), so the two never disagree, and it stays correct as verbs evolve because it's generated from niva's manifest. It runs over stdio with **no extra dependencies** — you just need niva installed on the interpreter the command runs (`niva lsp` starts the server; it needs QGIS only for hover on live-only algorithm ids, everything else is offline).

Neovim (built-in LSP, `init.lua`):

```
vim.filetype.add({ extension = { niva = "niva" } })
vim.api.nvim_create_autocmd("FileType", {
  pattern = "niva",
  callback = function(args)
    vim.lsp.start({ name = "niva", cmd = { "niva", "lsp" }, root_dir = vim.fn.getcwd() })
  end,
})
```

Helix (`~/.config/helix/languages.toml`):

```
[[language]]
name = "niva"
scope = "source.niva"
file-types = ["niva"]
language-servers = ["niva-lsp"]

[language-server.niva-lsp]
command = "niva"
args = ["lsp"]
```

VS Code / VSCodium: the bundled extension already includes the language-server client — build and install it (needs Node/npm):

```
bash .vscode/niva/install.sh    # packages a .vsix and installs it via the `code` CLI
```

then reload the window. (Manual install: `cd .vscode/niva && npm install && npx @vscode/vsce package`, then Extensions ⇒ ⋯ ⇒ *Install from VSIX…*.) If the `niva` on VS Code's PATH isn't the one you want, set **`niva.lsp.command`** in Settings to an absolute path. See the extension's README. **Kate / KWrite**: `bash .vscode/niva/install.sh` writes the LSP server config for you (`~/.config/kate/lspclient/settings.json`); then enable it once via *Settings → Configure Kate → Plugins → "LSP Client"* and reopen a `.niva` file. (Manual equivalent: add `{"servers":{"niva":{"command":["niva","lsp"],"highlightingModeRegex":"^niva$"}}}` under *Settings → LSP Client → User Server Settings*.)

> Make sure the `niva` command on your PATH is the one you installed (see the Quick start); `niva lsp` uses that interpreter's niva.

One command (Linux / macOS)

```
bash .vscode/niva/install.sh
```

Idempotent, per-user, no root. It detects VS Code/VSCodium, Vim/Neovim, nano, the GtkSourceView family (Mousepad, gedit, …), and Kate, and installs into each. Restart your editor afterward. Windows and a few editors are manual — see below.

Coverage at a glance

niva ships syntax in **five formats**, which between them cover essentially every editor:

Editor	Format used	Highlighting	Snippets / completion
VS Code / VSCodium	TextMate + snippets	✓	✓ (see completion notes)
Sublime Text	TextMate (`.tmLanguage.json`)	✓	—
Zed	TextMate	✓	—
JetBrains (PyCharm/IDEA/…)	TextMate	✓	—
bat (pager)	TextMate	✓	—
Vim / Neovim	Vim syntax	✓	—
nano	`.nanorc`	✓	—
Mousepad / gedit / GNOME Builder / Pluma / xed	GtkSourceView `.lang`	✓	—
Kate / KWrite / KDevelop	KSyntaxHighlighting XML	✓	—
Notepad++ (Windows)	UDL XML	✓	—
GitHub	Linguist (`.gitattributes`)	✓	—
Emacs / Helix	— (not yet)	—	—

The principle: **most editors accept one of these five formats.** For any editor not listed, find which format it understands (TextMate, GtkSourceView, KSyntaxHighlighting, Vim, or a nano-style regex list) and point it at the matching file below.

VS Code / VSCodium

The richest integration — highlighting **plus** snippets and command-dropdown completion.

```
cp -r .vscode/niva ~/.vscode/extensions/niva      # VSCodium: ~/.vscodium/extensions
```

Then reload the window (**Ctrl/Cmd+Shift+P → Developer: Reload Window**). Type a verb prefix (`load`, `pipeline`, `pdalcli-dtm`, `chm`, ...) and Tab to expand. See the extension README for the snippet list and what "completion" covers.

Paths by OS: Linux/macOS `~/.vscode/extensions/` · Windows `%USERPROFILE%\.vscode\extensions\`.

Any TextMate-compatible editor (Sublime, Zed, JetBrains, bat)

They all consume `syntaxes/niva.tmLanguage.json`:

- **Sublime Text** — copy it into `Packages/User/` (Preferences → Browse Packages).
- **JetBrains** — Settings → Editor → TextMate Bundles → **+** → add the `.vscode/niva/syntaxes/` folder.
- **Zed** — add the grammar via a language extension, or reference the TextMate file in settings.
- **bat** — `mkdir -p ~/.config/bat/syntaxes && cp syntaxes/niva.tmLanguage.json ~/.config/bat/syntaxes/ && bat cache --build`.

Vim / Neovim

```
mkdir -p ~/.vim/syntax ~/.vim/ftdetect                # Neovim:
 ↪  ~/.config/nvim/{syntax,ftdetect}
cp .vscode/niva/vim/niva.vim ~/.vim/syntax/niva.vim
echo 'au BufRead,BufNewFile *.niva set filetype=niva' > ~/.vim/ftdetect/niva.vim
```

The `ftdetect` file is what makes Vim recognise `*.niva`; without it you'd `:set ft=niva` by hand.

nano

```
mkdir -p ~/.nano
cp .vscode/niva/nano/niva.nanorc ~/.nano/niva.nanorc
echo 'include "~/.nano/niva.nanorc"' >> ~/.nanorc
```

nano ≥ 2.9 required. System-wide alternative: copy to `/usr/share/nano/` (the stock `~/.nanorc` usually already `includes` that folder).

Mousepad / gedit / GNOME Builder / Pluma / xed (GtkSourceView)

```
mkdir -p ~/.local/share/gtksourceview-4/language-specs
```

```
cp .vscode/niva/gtksourceview/niva.lang
 ↪  ~/.local/share/gtksourceview-4/language-specs/niva.lang
```

Use `gtksourceview-3.0` for older apps (gedit 3, Pluma, xed) and `gtksourceview-5` for the newest. Restart the editor; it picks up `*.niva` from the `.lang` file's `globs`.

Kate / KWrite / KDevelop (KSyntaxHighlighting)

```
mkdir -p ~/.local/share/org.kde.syntax-highlighting/syntax
cp .vscode/niva/kate/niva.xml ~/.local/share/org.kde.syntax-highlighting/syntax/niva.xml
```

Restart Kate — it indexes that folder at startup. (Older KDE: `~/.local/share/katepart5/syntax/`.)

Notepad++ (Windows)

Language → User Defined Language → Define your language… → Import… and choose `.vscode\niva\npp\niva.udl.xml`. Restart Notepad++.

GitHub

`.gitattributes` maps `*.niva` to the `Niva` language via Linguist, so niva files render with highlighting in the GitHub web UI and diffs — nothing to install.

Emacs / Helix — not yet

No mode ships yet. Emacs would want a `niva-mode` (derive from `prog-mode`, reuse the verb lists here); Helix would want a Tree-sitter grammar. Contributions welcome — the verb/alias lists in any of the files above are the source of truth.

Keeping the definitions in sync

All formats encode the same three lists — **23 built-in verbs**, **45 alias verbs**, and the token rules (options key=, `@conn`, `EPSG:`, distances, pipe, `{template}`). When verbs change, update every file under `.vscode/niva/`. The lists mirror niva's registry (`niva/registry/definitions.py`) and the built-in dispatch in `niva/engine/engine.py`.

PDAL & LAStools in QGIS 4 on Ubuntu (native install)

How to unlock point cloud processing in QGIS 4 and make LiDAR algorithms available to niva's `run` escape hatch, the Processing Toolbox, and PyQGIS.

Verified on this host: **Ubuntu 26.04 LTS, QGIS 4.0.3-Norrköping**.

> Just want the point-cloud backend working on your OS? The plain, per-platform manual (Windows / macOS / Linux, no agent required) is **pdal-setup.md**. This page is the deep dive — every algorithm, LAStools/OTB/SAGA, and host specifics.

Quick status on this host (after the setup in this guide):

Component	State	What it means
PDAL processing provider (`pdal`)	**Working** — 24 algorithms	Provider is compiled into QGIS; `pdal_wrench` now installed and verified end-to-end
`pdal` **data** provider (loading raw LAS/LAZ)	**Not built in**	QGIS here reads only `copc` / `ept` / `vpc` point clouds — raw `.las`/`.laz` must be converted to COPC first
Orfeo ToolBox (`otb`)	**Working** — 109 algorithms	OTB 9.1.1 + provider plugin; verified loading in QGIS 4 and running via CLI
SAGA (`sagang`)	**CLI only**	`saga_cmd` 9.9.3 installed and works; the QGIS provider plugin is unavailable for QGIS 4 (see below)
LAStools plugin + binaries	**Extracted, plugin not installed**	Third-party; binaries at `~/Downloads/LAStools`, still needs the QGIS plugin

The most important correction versus older notes: **PDAL is not an `apt install pdal` step here.** QGIS 4.0.3 already ships the PDAL *processing* provider built in — the 24 algorithms appear in the Processing Toolbox out of the box. What was actually missing is (1) the `pdal_wrench` command-line helper the algorithms shell out to, and (2) the ability to open raw LAS/LAZ, because this build has **no `pdal` data provider** — only `copc`/`ept`/`vpc`. Both are addressed below and were verified working.

PDAL (native point cloud provider)

PDAL is the open-source point cloud library. QGIS 4 ships a **built-in PDAL processing provider** (`QgsPdalAlgorithms` in `qgis.analysis`). On this host it is **already loaded** — you can confirm 24 algorithms are registered under provider id `pdal` without installing anything.

How the provider actually works

The provider has two independent pieces:

1. **The provider itself** — `QgsPdalAlgorithms`, compiled into `libqgis_analysis` when QGIS is built `WITH_PDAL`. The Ubuntu 26.04 QGIS 4.0.3 package is built this way, so `from qgis.analysis import QgsPdalAlgorithms` succeeds and the 24 algorithms register at startup. **Nothing to install for this part.**
2. **The `pdal_wrench` executable** — every PDAL algorithm is a thin wrapper that shells out to the `pdal_wrench` command-line tool. QGIS finds it on `PATH`, or via the `QGIS_WRENCH_EXECUTABLE` environment variable. If it can't be found, running any PDAL algorithm fails with:

   ```
   wrench executable is not found. Either use QGIS build with PDAL support
   or provide correct path via QGIS_WRENCH_EXECUTABLE environment variable.
   ```

There is currently **no `pdal` package available via apt** on this host (checked: `apt-cache policy pdal` returns nothing, even with `universe` enabled), and Homebrew's `pdal` formula does **not** build the standalone `pdal_wrench`. The reliable source is conda-forge, which ships `pdal_wrench` as its own package.

Making the algorithms runnable — verified method

This is the exact sequence used and verified on this host. It installs everything under `$HOME` with no root.

1. Install micromamba (single static binary, self-contained):

```
cd ~ && curl -Ls https://micro.mamba.pm/api/micromamba/linux-64/latest | tar -xvj
 ↪  bin/micromamba
```

2. Create an env with `pdal_wrench` (the standalone tool is its own conda-forge package, *not* bundled in `pdal`):

```
export MAMBA_ROOT_PREFIX=$HOME/micromamba
~/bin/micromamba create -y -p $HOME/micromamba/envs/pdal -c conda-forge pdal pdal_wrench
# -> ~/micromamba/envs/pdal/bin/pdal_wrench   (v1.5.1, verified working; pdal 2.10.2)
```

3. Point QGIS at it. Add to `~/.bashrc` so both the QGIS desktop app (when launched from a shell) and the niva CLI inherit it:

```
export QGIS_WRENCH_EXECUTABLE="$HOME/micromamba/envs/pdal/bin/pdal_wrench"
```

No plugin, no Processing config — the provider is already there; it just needed the backend binary.

> Note: `untwine` is **not** required on this build. It only matters for the `pdal` *data* provider (raw-LAS indexing), which isn't compiled into this QGIS. Point clouds come in via COPC instead (next section).

Loading point clouds: convert raw LAS/LAZ to COPC first

This QGIS build has **no `pdal` data provider** — its point-cloud data providers are `copc`, `ept`, and `vpc` only. A raw `.las`/`.laz` therefore fails to load as a layer (`Could not load source layer for INPUT`), which blocks every PDAL algorithm. Convert once to Cloud-Optimized Point Cloud (COPC) using the `pdal` CLI from the same env:

```
~/micromamba/envs/pdal/bin/pdal translate input.las output.copc.laz
# ~9 s for a 159 MB / 5.3 M-point tile on this host; 159 MB -> 29 MB
```

`.copc.laz` loads natively (QGIS `copc` provider) and works as the `INPUT` to any PDAL algorithm. Verified end-to-end here: `pdal:info` and `pdal:exportraster` both ran against a COPC layer and `exportraster` produced a valid 1000×1000 Float32 DEM at 2 m resolution.

Verify the provider (this host's Python)

QGIS's Python isn't on the default `sys.path` here; the PDAL check needs the QGIS bindings **and** the bundled `processing` plugin on PYTHONPATH:

```
PYTHONPATH=/usr/share/qgis/python:/usr/share/qgis/python/plugins:/usr/lib/python3/dist-
 ↪  packages \
QT_QPA_PLATFORM=offscreen /usr/bin/python3 - <<'PY'
from qgis.core import QgsApplication
QgsApplication.setPrefixPath('/usr', True)
app = QgsApplication([], False); app.initQgis()
from processing.core.Processing import Processing
Processing.initialize()
reg = QgsApplication.processingRegistry()
p = reg.providerById('pdal')
print('PDAL provider loaded:', p is not None)
if p:
    print('algorithms:', len(list(p.algorithms())))
import os; os._exit(0)
PY
```

On this host this prints `PDAL provider loaded: True` and `algorithms: 24` — before installing anything. That confirms the *provider* is wired; it does **not** confirm `pdal_wrench` is present (that only shows up when you actually run an algorithm).

PDAL algorithms exposed by QGIS 4.0.3

QGIS 4.0.3 registers these **24** PDAL-based algorithms (enumerated live from the registry on this host). All are reachable via `run pdal:<id>`.

Point cloud conversion

Algorithm ID	Name
`pdal:convertformat`	Convert point cloud format
`pdal:exportraster`	Export point cloud to raster
`pdal:exportrastertin`	Export point cloud to raster (using triangulation)
`pdal:exportvector`	Export point cloud to vector

Point cloud data management

Algorithm ID	Name
`pdal:assignprojection`	Assign projection
`pdal:classifyground`	Classify ground points
`pdal:clip`	Clip point cloud
`pdal:compare`	Compare point clouds
`pdal:createcopc`	Create COPC
`pdal:filternoiseradius`	Filter noise (using radius)
`pdal:filternoisestatistical`	Filter noise
`pdal:heightabovegroundbynearestneighbor`	Height above ground
`pdal:heightabovegroundtriangulation`	Height above ground (using triangulation)
`pdal:info`	Point cloud information
`pdal:merge`	Merge point cloud
`pdal:reproject`	Reproject point cloud
`pdal:thinbydecimate`	Thin (by skipping points)
`pdal:thinbyradius`	Thin (by sampling radius)
`pdal:tile`	Create tiles from point cloud
`pdal:transformpointcloud`	Transform point cloud
`pdal:virtualpointcloud`	Build virtual point cloud (VPC)

Point cloud extraction

Algorithm ID	Name
pdal:boundary	Boundary
pdal:density	Point cloud density
pdal:filter	Filter point cloud

See the QGIS PDAL algorithm docs (conversion, data management, and extraction pages) for full parameter details.

Parameter names are UPPERCASE

PDAL algorithms follow the **same uppercase `INPUT`/`OUTPUT` convention as every other QGIS provider** — not a lowercase, provider-specific scheme. They add a few PDAL-specific parameters (`FILTER_EXPRESSION`, `FILTER_EXTENT`, `VPC_OUTPUT_FORMAT`, etc.). Verified signatures for the common ones:

Algorithm	Parameters
pdal:exportraster	INPUT, ATTRIBUTE, RESOLUTION, TILE_SIZE, FILTER_EXPRESSION, FILTER_EXTENT, ORIGIN_X, ORIGIN_Y, OUTPUT
pdal:clip	INPUT, OVERLAY, FILTER_EXPRESSION, FILTER_EXTENT, VPC_OUTPUT_FORMAT, OUTPUT
pdal:merge	LAYERS, FILTER_EXPRESSION, FILTER_EXTENT, OUTPUT
pdal:thinbydecimate	INPUT, POINTS_NUMBER, FILTER_EXPRESSION, FILTER_EXTENT, VPC_OUTPUT_FORMAT, OUTPUT
pdal:filter	INPUT, FILTER_EXPRESSION, FILTER_EXTENT, VPC_OUTPUT_FORMAT, OUTPUT
pdal:classifyground	INPUT, CELL_SIZE, SCALAR, SLOPE, THRESHOLD, WINDOW_SIZE, VPC_OUTPUT_FORMAT, OUTPUT
pdal:heightabovegroundtriangulation	INPUT, REPLACE_Z, COUNT, VPC_OUTPUT_FORMAT, OUTPUT
pdal:reproject	INPUT, CRS, OPERATION, VPC_OUTPUT_FORMAT, OUTPUT
pdal:info	INPUT, OUTPUT

Run `niva describe pdal:<id>` to see the exact signature for any algorithm before scripting it.

From niva

niva's `run <algorithm-id> KEY=value` escape hatch passes straight through to `processing.run(...)`. Once `pdal_wrench` is reachable, these work:

```
# raster/DEM from the Z attribute at 1 m resolution
run pdal:exportraster INPUT="~/data/lidar.laz" ATTRIBUTE=Z RESOLUTION=1
 ↪  OUTPUT="~/data/dem.tif"

# clip to a polygon overlay
run pdal:clip INPUT="~/data/lidar.laz" OVERLAY="~/data/aoi.gpkg"
 ↪  OUTPUT="~/data/lidar_clip.laz"

# merge several tiles
run pdal:merge LAYERS="~/data/a.laz;~/data/b.laz" OUTPUT="~/data/merged.laz"

# progressive-morphological ground classification (no CLASSIFY param — tune
 ↪  SLOPE/THRESHOLD/WINDOW_SIZE)
run pdal:classifyground INPUT="~/data/lidar.laz" OUTPUT="~/data/classified.laz"
```

```
# height above ground (normalised heights) via triangulation
run pdal:heightabovegroundtriangulation INPUT="~/data/lidar.laz" OUTPUT="~/data/hag.laz"

# statistical outlier / noise removal
run pdal:filternoisestatistical INPUT="~/data/lidar.laz" OUTPUT="~/data/denoised.laz"
```

> Until `pdal_wrench` is on PATH (or `QGIS_WRENCH_EXECUTABLE` is set), these fail with a *"wrench executable is not found"* error even though niva finds the algorithm — the algorithm exists, its backend doesn't.

LAStools (rapidlasso) — install & configure

LAStools is a **third-party** command-line LiDAR toolset (proprietary but free for most use; commercial license for unrestricted use). It has **no built-in provider** in QGIS — you need the **LAStools Processing Provider plugin**.

1. Extract LAStools

A copy is already at `/home/jcz/Downloads/LAStools.tar.gz` (~59 MB, confirmed present). Extract it:

```
mkdir -p ~/lastools
tar xzf /home/jcz/Downloads/LAStools.tar.gz -C ~/lastools
# creates ~/lastools/LAStools/ with bin/, lib/, etc.
```

Verify the tools:

```
ls ~/lastools/LAStools/bin/
```

Native 64-bit Linux binaries (no Wine needed): `laszip64`, `lasinfo64`, `las2dem64`, `lasground64`, `lasmerge64`, `blast2dem64`, etc.

2. Install the QGIS plugin

Plugins → Manage and Install Plugins → Search "LAStools" → Install

Use a version that supports QGIS 3.x/4. The plugin is maintained by rapidlasso at github.com/rapidlasso/LAStools

3. Configure the LAStools folder

In QGIS:

1. **Processing → Options**
2. Scroll to **Providers → LAStools**
3. Set **LAStools folder** to `/home/<you>/lastools/LAStools` (the directory containing `bin/`)
4. Leave **Wine folder** blank — Linux native binaries don't need it
5. Click OK

The Processing Toolbox will now show a **LAStools** section.

4. Verify

From the Processing Toolbox, try `lasinfo` on a `.las` file, or from niva:

```
run lastools:lasinfo INPUT="~/data/in.laz" OUTPUT="~/data/info.txt"
```

License note: some LAStools tools restrict output (adds noise / diagonal artifacts) without a commercial license. Place `lastoolslicense.txt` in the `bin/` directory or set the `LAStoolsLicenseFile` environment variable.

LAStools ↔ PDAL: when to use which

Both can now cover the basics natively — QGIS 4's PDAL provider does include ground classification, noise removal, and height-above-ground (contrary to older notes). Choose based on robustness and specialization:

Task	Suggested	Reason
Read / write / merge / convert	PDAL (`pdal:merge`, `pdal:convertformat`)	Built into QGIS, no extra plugin
Export raster (DEM/DSM)	PDAL (`pdal:exportraster`, `pdal:exportrastertin`)	Controlled resolution, TIN option
Ground classification	PDAL (`pdal:classifyground`) or LAStools (`lasground`)	PDAL for free/basic; LAStools more robust on complex terrain
Noise / outlier removal	PDAL (`pdal:filternoisestatistical` / `pdal:filternoiseradius`)	Built in, adequate
Height above ground	PDAL (`pdal:heightabovegroundtriangulation`)	Built in
Canopy metrics / forestry	LAStools (`lascanopy`)	Specialized, not in PDAL provider
Building/vegetation classification	LAStools (`lasclassify`)	Mature classifier
Tiling / batch	Either (`pdal:tile` or LAStools)	Both support batch workflows

---

Orfeo ToolBox & SAGA (extra providers)

QGIS 4 ships neither the **Orfeo ToolBox (OTB)** nor the **SAGA** processing provider in core (verified: `providerById('otb')` and `providerById('saga')`/`providerById('sagang')` all returned `False` before setup). **Providers registered out of the box:** `native` (339), `gdal` (59), `grass` (307), `qgis` (39), `pdal` (24), `3d` (1).

Orfeo ToolBox — installed & verified working in QGIS 4 ✓

OTB has no apt package here, but a self-contained build was already downloaded to `~/Downloads/OTB-9.1.1-Linux`. Even though the QGIS provider plugin is officially marked for QGIS 3.x, **it loads and runs under QGIS 4.0.3** — verified: 109 OTB algorithms registered, and `otbcli_Smoothing` produced a valid output raster.

1. **Confirm OTB works** (first launch runs one-time post-install steps):

```
~/Downloads/OTB-9.1.1-Linux/bin/otbcli_BandMath -help   # prints "version 9.1.1"
```

2. **Install the OTB provider plugin.** The plugin manager won't show it under QGIS 4 (it's tagged max 3.99), so install the zip manually:

```
curl -sSL -o /tmp/otbprov.zip \
  "https://plugins.qgis.org/plugins/orfeoToolbox_provider/version/3.0.3/download/"
unzip -o /tmp/otbprov.zip -d ~/.local/share/QGIS/QGIS3/profiles/default/python/plugins/
```

Then enable it once in **Plugins → Manage and Install Plugins → Installed → tick "OrfeoToolbox Provider"**.

3. **Point it at the OTB install.** In **Processing → Options → Providers → OTB**:
   - **OTB folder** → `/home/<you>/Downloads/OTB-9.1.1-Linux`

- **OTB application folder** → `/home/<you>/Downloads/OTB-9.1.1-Linux/lib/otb/applications`

Restart QGIS. An **OTB** section (~109 algorithms) appears, reachable via `run otb:<id>` — e.g.:

```
run otb:Smoothing -in "~/data/image.tif" -out "~/data/smooth.tif" -type mean
```

> The OTB *Python* bindings fail to recompile against system Python 3.14 (a harmless first-launch step). That does **not** affect the CLI apps QGIS calls, so the provider is unaffected.

SAGA — CLI works, but no QGIS-4 provider plugin ⚠

The SAGA binaries are installed and current: **`saga_cmd` 9.9.3** at `/usr/bin/saga_cmd` (via the `saga` / `saga-common` apt packages). SAGA computations work fine from the command line.

What does not work: getting SAGA into the QGIS 4 Processing Toolbox. QGIS dropped SAGA from core after 3.30, and the community replacement — **Processing Saga NextGen Provider** (`sagang`) — has been **withdrawn from every authoritative source**:

- Not in the QGIS plugin repository feed for QGIS 4 (the plugin-manager search "leads nowhere");
- Direct download from `plugins.qgis.org` returns **404**;
- The maintainer repo `north-road/qgis-processing-saga-nextgen` now **404s**;
- Alexander Bruy's original plugin is no longer in his repository either.

The only copies still circulating are **untrusted third-party redistributions** (random Google Drive folders, unofficial mirrors). A QGIS plugin is arbitrary Python that runs inside QGIS with your privileges, so installing one from an unvetted Drive link is a real supply-chain risk — **not recommended.**

Practical options for SAGA today:

1. **Use `saga_cmd` directly** (works now, no QGIS needed):
   `saga_cmd ta_morphometry 0 -ELEVATION dem.tif -SLOPE slope.tif -ASPECT aspect.tif`
2. **Reach SAGA via GRASS**, which *is* a working QGIS 4 provider here (307 algorithms) and covers much of the same terrain/hydrology ground.
3. **Wrap `saga_cmd` in a niva backend/harness** so niva calls it directly, bypassing the dead QGIS provider — see the note below.

Troubleshooting on this host (Ubuntu 26.04 — QGIS 4.0.3)

Symptom	Cause	Fix
PDAL algorithms visible but `run pdal:*` → *"wrench executable is not found"*	`pdal_wrench` binary missing or not pointed to	Install `pdal_wrench` (conda-forge `pdal_wrench` package) and set `QGIS_WRENCH_EXECUTABLE`
`run pdal:*` → *"Could not load source layer for INPUT: … not found"* (file exists)	No `pdal` data provider here — raw LAS/LAZ can't be opened	Convert once: `pdal translate in.las out.copc.laz`, then use the `.copc.laz` as INPUT
`run pdal:exportraster … FILTER_EXPRESSION="Classification==2"` → *"Argument 'filter' needs a value and none was provided"*	QGIS evaluates `FILTER_EXPRESSION` as a **QGIS** expression first; `==` is invalid QGIS syntax → it emits an empty `--filter=`. (Confirmed: QGIS logs `--filter=` for `==`, but a correct `--filter=…` for a `>`/`and` form.)	Use a `=`-free range form — `FILTER_EXPRESSION="Classification > 1.5 and Classification < 2.5"` — or use `run pdalcli:*` where `filter="Classification==2"` passes straight to wrench
`sudo apt install pdal` → *"Unable to locate package"*	No `pdal` apt package in the current cache	Use micromamba/conda-forge (see above); Homebrew `pdal` lacks `pdal_wrench`

Symptom	Cause	Fix
`run pdal:groundfilter` / `pdal:smrf` → ”algorithm not found”	Those IDs don't exist	Ground classification is `pdal:classifyground`; see the algorithm table for real IDs
`pdal:*` point-cloud output → *”Incorrect parameter value for VPC_OUTPUT_FORMAT”*	That enum is required for point-cloud-producing algorithms	Pass a valid `VPC_OUTPUT_FORMAT`; run `niva describe pdal:<id>` for allowed values
niva `run pdal:` params rejected	Wrong parameter names	Params are UPPERCASE (`INPUT`, `OUTPUT`, ...); run `niva describe pdal:<id>`
`run otb:*` → ”algorithm not found”	OTB folder not set / plugin not enabled	Install the OTB provider zip, enable it, set OTB folder + application folder
OTB provider shows 0 algorithms	`OTB_FOLDER` / `OTB_APP_FOLDER` unset	Set both in Processing → Options → Providers → OTB, then restart
`run lastools:*` → ”algorithm not found”	LAStools plugin not installed	Install via Plugins manager + set LAStools folder
”LAStools folder is not set”	Plugin needs path to `bin/` dir	Processing → Options → Providers → LAStools
`lasinfo: command not found` in shell	LAStools not on `PATH`	`export PATH=$PATH:~/Downloads/LAStools/bin` in `~/.bashrc`
`run sagang:*` → ”algorithm not found”	No SAGA provider available for QGIS 4	Provider plugin is withdrawn; use `saga_cmd` directly or GRASS instead

---

The native-CLI harness — `pdalcli:` and `saga:`

Some tools are best driven by their own command line rather than a QGIS provider: **PDAL** (the QGIS `pdal:` provider needs a COPC conversion first; `pdal_wrench` reads raw LAS directly), and **SAGA** (the QGIS provider is withdrawn — see above). niva ships a delegating adapter, **`NativeToolBackend`** (`niva/engine/native.py`), that wraps the real backend and intercepts two id families in `run`, shelling out to the CLI and wrapping the result as a niva layer so it still pipes. **Every other id — `native:*`, `gdal:*`, `grass:*`, `otb:*`, the QGIS `pdal:*` provider — passes straight through unchanged.** It is wired in transparently, so non-harness flows are unaffected.

Because the engine never imports QGIS (everything goes through the abstract Backend), this is just a third Backend composed over `PyqgisBackend`. It also teaches `load` to accept raw `.las`/`.laz`/`.copc.laz` as a path handle (they can't open as QGIS layers here).

`pdalcli:<command>` — PDAL on raw LAS/LAZ/COPC (no COPC step)

Shells to `pdal_wrench <command> --key=value ….` The upstream layer auto-wires to `--input`; the output goes to a scratch path (or an explicit `output=` you give — used to persist a `.laz`). `KEY=value` options become `--key=value`. Reads `.las`, `.laz`, and `.copc.laz` directly. Reuses `$QGIS_WRENCH_EXECUTABLE` (or `$NIVA_PDAL_WRENCH`).

Command	Output	Purpose
`pdalcli:to_raster`	raster	Grid an attribute (Z, intensity…) to a 2D raster — **DTM/DSM**
`pdalcli:to_raster_tin`	raster	Same, via triangulation (smoother surface)
`pdalcli:density`	raster	Point-count-per-cell raster (coverage QA)
`pdalcli:to_vector`	vector	Points → GeoPackage (3D points + attributes)
`pdalcli:boundary`	vector	Coverage polygon (concave hull)
`pdalcli:translate`	point cloud	Convert / reproject / **filter by class** to a new LAS/LAZ
`pdalcli:clip`	point cloud	Keep points inside `polygon=<vector>`
`pdalcli:thin`	point cloud	Downsample (every-Nth or by radius)
`pdalcli:classify_ground`	point cloud	Label ground points (SMRF)
`pdalcli:filter_noise`	point cloud	Flag statistical/radius outliers
`pdalcli:height_above_ground`	point cloud	Normalise heights to the ground surface

Command	Output	Purpose
`pdalcli:merge`	point cloud	Combine tiles — `files="a.las;b.las"` (positional)

All classification-aware via `filter="Classification==2"` (or any PDAL expression). **Verified end-to-end on the Youngstown tiles** (DTM from ground, DSM, CHM, class-extract, classify_ground, merge, clip, COPC input). Raster/vector outputs pipe into `|  save` and downstream `grass:`/`gdal:`; point-cloud outputs persist via `output=`.

```
# DTM from ground class, DSM from all returns, then CHM in GRASS
load "tile.las" | run pdalcli:to_raster attribute=Z filter="Classification==2"
 ↪  resolution=1 | save "dtm.tif"
load "tile.las" | run pdalcli:to_raster attribute=Z resolution=1 | save "dsm.tif"
run grass:r.mapcalc.simple expression="A-B" a="dsm.tif" b="dtm.tif" output="chm.tif"
```

See examples/lidar_pdal_grass.niva for the full, verified workflow set.

`saga:<library>:<tool>` — SAGA via `saga_cmd`

Shells to `saga_cmd <library> <tool> -FLAG value …` (reuses `$NIVA_SAGA_CMD` or `saga_cmd` on PATH). Three reserved keys wire niva's pipe onto the tool's own (per-tool) parameter names:

- `_in=<FLAG>` — feed the upstream layer path to that SAGA input flag (e.g. `_in=ELEVATION`)
- `_out=<FLAG>` — give that output flag a scratch path and pipe it on
- `_outext=<.ext>` — output extension (default `.tif`; a vector ext marks the result vector)

Other `KEY=value` → `-KEY value` (booleans → 1/0). Tool may be an index or a name — **prefer names**, since SAGA's indices shift between versions.

```
load "dtm.tif" | run saga:ta_morphometry:0 _in=ELEVATION _out=SLOPE UNIT_SLOPE=1 | save
 ↪  "slope.tif"
```

Graceful degradation

The harness fails **closed and clear**, and only for the specific call:

- **Tool missing** → an actionable `OpError` ("`saga_cmd` not found … install SAGA, or set `$NIVA_SAGA_CMD`"). Unrelated flows (`native:`/`gdal:`/`grass:`) are unaffected. Check programmatically with `backend.available("saga" | "pdal")`.
- **Wrong SAGA version** → SAGA's tool ids drift between releases, so a failed `saga:*` call appends the **detected version** and a nudge to verify the id for it (e.g. *"detected SAGA 9.9.3; tool ids differ between SAGA versions — verify `saga:ta_morphometry:0` …"*).
- **OTB unconfigured** → a `run otb:*` "not found" is rewritten to point at the OTB folder setup (Processing → Options → Providers → OTB).

Cross-platform (Windows / macOS / Linux)

The harness is pure Python (`subprocess(shell=False)`, `shutil.which`, `tempfile`, `os.path`) — it runs unchanged on all three. What varies is only how you install the CLIs and set the env var:

- **`pdal_wrench`** — `micromamba/conda install -c conda-forge pdal_wrench` on all three OSes. On Windows/macOS, QGIS often bundles it and includes the full PDAL **data provider**, so the raw-LAS limitation and even the COPC step may not apply there.
- **`grass:`** — ships with QGIS everywhere; nothing to install.
- **`saga_cmd`** — SAGA installer (Windows), Homebrew/conda (macOS), apt (Linux).
- **Env var** — set `QGIS_WRENCH_EXECUTABLE` / `NIVA_SAGA_CMD` in the shell rc on Unix, or as a system/`setx` variable on Windows.

Security posture (all branches): fixed executable (never from flow input), `shell=False` with an explicit argv list, allowlist-validated library/tool/command/flag names (so crafted tokens like `--config` are rejected), scratch-dir outputs.

Point-cloud (PDAL) backend — setup for Windows, macOS & Linux

niva runs LiDAR / point-cloud steps (`run pdal:*` and `run pdalcli:*` — DTM/DSM, ground classification, clip, merge, boundary, height-above-ground, ...) through **PDAL** via QGIS. Those verbs have an **extra runtime dependency** that the rest of niva does not: a working PDAL backend. Everything else (`native:`, `gdal:`, `qgis:`, `grass:`) needs only QGIS.

This page is the **plain, per-platform manual** for getting that backend working — no agent required. For the deep dive (every algorithm, LAStools/OTB/SAGA, this project's host specifics) see **pdal-lastools-qgis4.md**.

> **TL;DR** — you need two things: (1) the **`pdal_wrench`** executable, which the QGIS PDAL algorithms shell out to, pointed at by the **`QGIS_WRENCH_EXECUTABLE`** environment variable; and (2) point clouds in **COPC** form (`.copc.laz`), because many QGIS builds can't open a raw `.las`/`.laz` as a layer. On **Windows/macOS the official QGIS installer usually ships both**; on **Linux you install `pdal` + `pdal_wrench` yourself** (one conda-forge command, no compiling).

The dependency, stated plainly

Piece	Why it's needed	How QGIS/niva finds it
`pdal_wrench` executable (from PDAL/wrench)	Every QGIS PDAL algorithm (`pdal:exportraster`, `pdal:merge`, ...) is a thin wrapper that shells out to it. QGIS integrated these in **3.32 (June 2023)**.	PATH, or the **`QGIS_WRENCH_EXECUTABLE`** env var (niva's `pdalcli:` also honours `NIVA_PDAL_WRENCH`).
COPC point clouds (`.copc.laz`)	Many QGIS builds have only the copc/ept/vpc **data** providers — no raw-LAS reader. A raw `.las` then fails with *"Could not load source layer for INPUT"*.	Convert once with the `pdal` CLI: `pdal translate in.las out.copc.laz`. (niva's `pdalcli:` reads raw LAS directly and skips this — see below.)

Miss the first and you get: *"wrench executable is not found. Either use QGIS build with PDAL support or provide correct path via QGIS_WRENCH_EXECUTABLE environment variable."* Miss the second and you get: *"Could not load source layer for INPUT: ..."* even though the file is right there.

`pdal_wrench` is pre-built for every desktop OS — conda-forge ships `pdal_wrench` **1.5.1** for `linux-64`, `osx-64`, `osx-arm64`, and `win-64`. **You never have to compile it.** (Building from source is possible but needs PDAL's full toolchain — see the appendix — and is not the recommended path on any platform.)

Windows

Most users need to do nothing. The official QGIS standalone installer and the OSGeo4W installer from qgis.org/download ship the PDAL stack, including `pdal_wrench` and a raw-LAS reader. Point-cloud algorithms appear in Processing and work out of the box. If `run pdal:*` already works, stop here.

If your QGIS lacks it (older build, minimal install), add it via conda without touching QGIS:

1. Install **Miniforge** (conda-forge community installer): github.com/conda-forge/miniforge → run the `.exe`.
2. Open **"Miniforge Prompt"** from the Start menu and create the tools:
   ```
   conda create -y -n pdal -c conda-forge pdal pdal_wrench
   ```
   This installs `pdal_wrench.exe` under `%USERPROFILE%\miniforge3\envs\pdal\Library\bin\`.
3. Tell QGIS where it is — set a **permanent** user environment variable (so the QGIS app and the niva CLI both inherit it). In a normal Command Prompt:
   ```
   setx QGIS_WRENCH_EXECUTABLE
    ↪  "%USERPROFILE%\miniforge3\envs\pdal\Library\bin\pdal_wrench.exe"
   ```
   Then **restart QGIS** (and any terminal) so the variable takes effect. (GUI route: *Settings → System → About → Advanced system settings → Environment Variables → New…*)

---

macOS

Most users need to do nothing. The official QGIS `.dmg` from qgis.org/download ships the PDAL stack including `pdal_wrench`. If `run pdal:*` works, stop here.

If it's missing, install the tools with conda-forge (works on both Intel and Apple-silicon):

1. Install **Miniforge**: github.com/conda-forge/miniforge (or `brew install miniforge`). Note: Homebrew's own `pdal` formula does **not** include `pdal_wrench` — use conda-forge.
2. Create the tools:
   ```
   conda create -y -n pdal -c conda-forge pdal pdal_wrench
   # -> ~/miniforge3/envs/pdal/bin/pdal_wrench
   ```
3. Point QGIS at it — add to `~/.zshrc` (Apple's default shell):
   ```
   export QGIS_WRENCH_EXECUTABLE="$HOME/miniforge3/envs/pdal/bin/pdal_wrench"
   ```
   To make the **QGIS desktop app** (launched from Finder, not a shell) see it too, also run once:
   ```
   launchctl setenv QGIS_WRENCH_EXECUTABLE "$HOME/miniforge3/envs/pdal/bin/pdal_wrench"
   ```

Linux

Linux QGIS packages (distro repos and the qgis.org apt repo) ship the PDAL *processing provider* but **frequently omit the `pdal_wrench` binary and a raw-LAS data provider** — so you install both yourself. There is usually **no `pdal` apt package** (e.g. Ubuntu 26.04: `apt-cache policy pdal` returns nothing even with `universe` enabled). conda-forge is the reliable, no-root, no-compile route — this is the exact sequence **verified on this project's host (Ubuntu 26.04, QGIS 4.0.3)**:

1. Install micromamba (one self-contained static binary, no root):

```
cd ~ && curl -Ls https://micro.mamba.pm/api/micromamba/linux-64/latest | tar -xvj
 ↪  bin/micromamba
```

2. Create an env with `pdal` + `pdal_wrench` (the wrench tool is its *own* conda-forge package — installing `pdal` alone is **not** enough):

```
export MAMBA_ROOT_PREFIX=$HOME/micromamba
~/bin/micromamba create -y -p $HOME/micromamba/envs/pdal -c conda-forge pdal pdal_wrench
# -> ~/micromamba/envs/pdal/bin/{pdal,pdal_wrench}   (pdal 2.10, wrench 1.5.1 — verified)
```

3. Point QGIS/niva at it — add to `~/.bashrc`:

```
export QGIS_WRENCH_EXECUTABLE="$HOME/micromamba/envs/pdal/bin/pdal_wrench"
```

> **Do *not* set `LD_LIBRARY_PATH`.** The conda `pdal_wrench` has an RPATH (`$ORIGIN/../lib`) baked in, so it finds its own libraries — no loader-path help needed. Setting `LD_LIBRARY_PATH=…/envs/pdal/lib` **globally** is actively harmful: it shadows system libraries for QGIS's *own* `gdalwarp`/`gdal_translate`, producing warnings like `libcurl.so.4: no version information available`. Just set the one env var above.
>
> conda/Miniforge works identically to micromamba if you already have it — just `conda create -n pdal -c conda-forge pdal pdal_wrench` and use `~/miniforge3/envs/pdal/...`.

Verify it works (any platform)

Fastest — the built-in helper. niva ships a diagnostic that finds `pdal_wrench` (env vars, PATH, conda envs, QGIS bundles), tells you exactly what to set if it's missing, and runs an end-to-end grid test:

```
niva pdal check     # locate the backend + per-OS fix if anything's missing
niva pdal test      # execute wrench and grid a synthetic cloud to a raster (PASS/FAIL)
niva pdal setup     # print the install commands for your OS
niva pdal test my_tile.las   # grid your own tile instead of synthetic data
```

`niva pdal check` works even when QGIS's Python isn't wired yet (the usual broken state) — it probes the binary directly and only *optionally* reports the QGIS provider.

Manual check — convert one raw tile to COPC, then grid a DTM through niva:

```
# 1. index raw LAS -> COPC (the pdal CLI reads .las natively; ~9 s for a 160 MB tile)
pdal translate your_tile.las /tmp/tile.copc.laz

# 2. DTM from the ground class, through niva
printf 'load /tmp/tile.copc.laz | run pdal:exportraster ATTRIBUTE=Z
 ↪  FILTER_EXPRESSION="Classification > 1.5 and Classification < 2.5" RESOLUTION=1 | save
 ↪  /tmp/dtm.tif\n' > /tmp/dtm.niva
niva run /tmp/dtm.niva

# 3. confirm real elevations
gdalinfo -stats /tmp/dtm.tif | grep -E "Minimum|Maximum"
```

A valid DTM (sensible min/max elevations for your area) means the backend is wired. On this project's Youngstown NY tiles this yields a bare-earth DTM of **74.5–86.1 m** (Lake Ontario datum ≈ 74.2 m).

Gotchas

`FILTER_EXPRESSION` with `==` is silently dropped by QGIS `pdal:` algorithms

QGIS treats `pdal:exportraster`'s `FILTER_EXPRESSION` as a **QGIS expression** and evaluates it before handing it to wrench. `Classification==2` is **invalid QGIS syntax** (QGIS equality is a

single =), so it evaluates to empty and QGIS emits `--filter=` (nothing) — wrench then fails with *"Argument 'filter' needs a value and none was provided."* Two forms that survive the evaluator:

- **Range form (recommended, =-free):** `FILTER_EXPRESSION="Classification > 1.5 and Classification < 2.5"` — selects integer class 2 only; QGIS rewrites and→&& and passes it through correctly.
- **Double-quoted literal:** the value must reach QGIS wrapped in `"…"` so it's passed verbatim (awkward to quote through the niva grammar; prefer the range form).

`niva`'s own **`pdalcli:`** verbs shell straight to `pdal_wrench` and are **not** affected — there `filter="Classification==2"` works directly.

`run pdal:*` → "Could not load source layer for INPUT"

Your QGIS has no raw-LAS reader. Convert to COPC first (`pdal translate in.las out.copc.laz`), or use **`pdalcli:`** which reads raw LAS/LAZ directly and skips the COPC step entirely:

```
load "tile.las" | run pdalcli:to_raster attribute=Z filter="Classification==2"
 ↪  resolution=1 | save "dtm.tif"
```

Which to use — `pdal:` vs `pdalcli:`

	`run pdal:*`	**`run pdalcli:*`**
Backend	QGIS PDAL provider → `pdal_wrench`	niva harness → `pdal_wrench` directly
Raw `.las`/`.laz` input	needs COPC conversion first	**reads directly**
Filter syntax	QGIS-expression quirks (see above)	native PDAL `filter="Classification==2"`
Pipes into `| save`, `gdal:`, `grass:`	yes	yes
Needs `QGIS_WRENCH_EXECUTABLE`	yes	yes (or `NIVA_PDAL_WRENCH`)

Both need `pdal_wrench`. Prefer **`pdalcli:`** for raw-LAS-heavy work (no COPC step, cleaner filters); use **`pdal:`** when you already have COPC/EPT and want the standard QGIS provider. See `examples/lidar_pdal_grass.niva`.

Appendix — building from source (not recommended)

Only if you cannot use conda/QGIS-bundled binaries. `pdal_wrench` needs **PDAL ≥ 2.5** and **GDAL ≥ 3.0**, both with development files, plus a C++ toolchain (CMake, a C++17 compiler). Where PDAL itself isn't packaged (e.g. Ubuntu 26.04 has neither `pdal` nor `libpdal-dev`), you must build PDAL first and then wrench against it — `cmake -DPDAL_DIR=<prefix>/lib/cmake/PDAL .. && make`. This is a heavy, error-prone build; on Windows/macOS especially, **use the QGIS-bundled or conda-forge binaries instead.**

Publishing a QGIS plugin — a reusable playbook

A distilled, repo-agnostic guide to packaging and publishing a QGIS plugin to the official **QGIS Plugin Repository** (`plugins.qgis.org`), written from niva's own setup so it can serve as a **template for other plugins**. Where a step is niva-specific it is called out; everything else transfers.

> Ground truth: the QGIS docs win over this file. Verify against **Migrate to QGIS 4** and the **plugin metadata reference** before trusting anything here.

---

1. `metadata.txt` — the contract with the repository

The repo ingests a zip whose top folder contains a `metadata.txt`. Fields niva sets, and why:

Field	niva value	Notes
`name`	`niva`	Display name.
`qgisMinimumVersion`	`3.22`	Oldest QGIS you support.
`qgisMaximumVersion`	`4.99`	**This alone gates QGIS 4 readiness** (see §2).
`description`	one line	Short; shown in listings.
`about`	paragraph	Longer; indented continuation lines.
`version`	`0.42.4`	Must be **unique per upload** — bump every upload.
`author` / `email`	—	Required.
`repository`	GitHub URL	**Required** by the repo.
`tracker` / `homepage`	URLs	Recommended.
`category`	`Plugins`	Menu placement (Vector/Raster/Database/Web/Plugins).
`tags`	`processing,automation,…`	Comma-separated; drives discoverability.
`icon`	`icon.png`	Raster preferred for the website thumbnail (see §4).
`experimental`	`True`	Only shown to users who opt into experimental plugins. Flip to `False` when stable.
`deprecated`	`False`	—
`changelog`	latest 1–2 versions	Shown in the Plugin Manager; keep concise.

Do NOT set `plugin_dependencies` unless you truly need a pip package — a self-contained plugin installs with no extra steps (niva vendors its pure-Python package; see §4).

---

2. QGIS 4 / Qt6 readiness — the part that trips people up

The **`supportsQt6` flag is removed from QGIS core and no longer recognised.** Do not set it. QGIS-4 readiness is determined **solely** by:

```
qgisMaximumVersion=4.99
```

Setting that (with your existing `qgisMinimumVersion=3.xx`) lists the plugin for **both** QGIS 3.x and QGIS 4.x. (The niva repo learned this from a plugins.qgis.org reviewer message on v0.42.3.)

Code must actually work on Qt6. The safe, dual-version pattern — used throughout niva's `plugin/` — is to import Qt through the QGIS-provided compatibility layer **`qgis.PyQt`**, never raw PyQt5/PyQt6:

```
from qgis.PyQt.QtCore import Qt          # not `from PyQt5.QtCore import Qt`
from qgis.PyQt.QtWidgets import QDockWidget
```

Watch for classes **relocated between Qt5 and Qt6** — most notably `QAction` moved from `QtWidgets` (Qt5) to `QtGui` (Qt6). Handle both:

```
try:                                     # Qt6
    from qgis.PyQt.QtGui import QAction
except ImportError:                      # Qt5
    from qgis.PyQt.QtWidgets import QAction
```

Then **test on a real QGIS 4 build** before publishing.

---

3. License

- Ship an **OSI-approved** license (QGIS itself is GPL; GPL-compatible is expected).
- Keep the license **consistent** across `LICENSE`, `pyproject.toml`, README badges, and any deck — a mismatch is a red flag. (niva declares **GPL-3.0-or-later** and bundles the full GPL-3 text.)
- **Bundle the `LICENSE` inside the plugin zip**, not just in the git repo (see §4).

---

4. Packaging the zip (niva's `plugin/build_plugin.sh` pattern)

Requirements the repo enforces, and how niva meets them:

1. **Exactly one top-level folder**, and its name must be a **valid Python identifier** — it becomes the plugin's import package. niva uses `niva_qgis/` (not `niva/`, which would shadow the vendored library).
2. **Self-contained** — no pip step for users. niva vendors its zero-dependency pure-Python package under `niva_qgis/libs/niva/` and adds `libs/` to `sys.path` at load.
3. **Bundle `LICENSE`** and an **`icon.png`** (render from SVG with `rsvg-convert -w 256 -h 256 --background-color=none icon.svg -o icon.png`).
4. **Strip `__pycache__` / `*.pyc`** so the zip is clean.

Sketch (see `plugin/build_plugin.sh` for the full script):

```
STAGE="$WORK/niva_qgis"
cp plugin/{__init__,plugin,dock,runner,…}.py plugin/metadata.txt plugin/icon.{svg,png}
 ↪  "$STAGE/"
cp LICENSE "$STAGE/LICENSE"                # ship the license inside the package
cp -r niva "$STAGE/libs/niva"              # vendor the pure-Python lib
find "$STAGE" -name __pycache__ -prune -exec rm -rf {} +
( cd "$WORK" && zip -qr niva_qgis.zip niva_qgis )
```

Verify before shipping:

```
unzip -l niva_qgis.zip | grep -E 'niva_qgis/(metadata.txt|LICENSE|icon.png)$'
unzip -p niva_qgis.zip niva_qgis/metadata.txt | grep -E '^version|^qgisMaximumVersion'
```

Distribution: niva keeps the built zip **gitignored** and attaches it to a **GitHub Release** per version (`gh release create vX.Y.Z <zip>`), rather than committing the binary. The same zip is what you upload to plugins.qgis.org.

---

5. Publishing to plugins.qgis.org

1. **Register / log in** with an **OSGeo ID** (the plugins.qgis.org account).
2. **Share a plugin → upload** the zip.
3. **First upload of a new plugin goes to an approval queue** — a human on the QGIS plugin-approval team reviews it (allow a few days). After approval, subsequent version uploads publish automatically.
4. With `experimental=True`, it appears only to users who tick **"Show experimental plugins"**. Flip to `False` and re-upload (new version) once you've tested and are confident.

---

6. Security scanning — clear it BEFORE you upload

Every upload to plugins.qgis.org is **automatically scanned**. Two scanners are **BLOCKING** — a finding delists the version until you upload a fixed one — and two are informational:

Tool	Status	Catches
Bandit	□ blocking	Python security issues — shell/`eval`/`pickle`, SQL injection, weak crypto, unsafe URL/XML, …
detect-secrets	□ blocking	Hardcoded secrets: cloud creds, API keys/tokens, SSH keys, DB strings, high-entropy strings
Flake8	□ info	PEP 8 style, syntax, unused names
File analysis	□ info	Executable/hidden files, suspicious types, odd permissions

Only Bandit findings at MEDIUM or HIGH severity block — LOW findings (e.g. `try/except/pass` `B110`, `subprocess` `B404`/`B603`) are reported but do not block. Fix the blockers; document the rest.

Run the exact tools locally first

```
pip install bandit detect-secrets flake8            # or: uv tool install bandit
↪ detect-secrets
bandit -r path/to/plugin/            # scan the plugin tree; add --severity-level medium
↪ for blockers
detect-secrets scan path/to/plugin/
flake8 path/to/plugin/               # informational
```

Scan the **built** plugin tree (what the repo actually sees), not just your source — for niva that means unzipping `niva_qgis.zip` and running `bandit -r niva_qgis/`, because the scanner reads the vendored `libs/` too.

How niva cleared its 5 blockers (v0.42.5) — prefer real fixes over `# nosec`

The docs don't promise `# nosec` is honoured, so **change the code** wherever a genuine fix exists:

Bandit	Where	Real fix
B324 weak SHA-1	a geometry-dedup digest	add `usedforsecurity=False` — tells Bandit (and Python) it isn't a security hash. Needs Python ≥ 3.9.

Bandit	Where	Real fix
B608 SQL injection	string-built `INSERT`	write through `sqlite3` with **bound `?` parameters** (`executemany`), not an f-string. If your DB API has no bind params, validate identifiers against a strict allowlist.
B310 urlopen scheme ×2	`urllib` fetches	enforce an **http/https scheme allowlist**, and fetch via an `OpenerDirector` that carries **only** `HTTPHandler`/`HTTPSHandler` (+redirect/error) — no `FileHandler`/`FTPHandler`, so no request or redirect can read `file://`. Using `OpenerDirector.open` instead of `urllib.request.urlopen` also sidesteps Bandit's B310 blacklist at the AST level.
B314/B405 XML parse	`xml.etree` on remote XML	reject any `DOCTYPE` before parsing (kills XXE / billion-laughs), then `# nosec B314/B405`. **No stdlib XML parser passes Bandit** — the only Bandit-clean option is `defusedxml`; niva stays zero-dependency and relies on the DOCTYPE guard + a justified `# nosec`.

Rules of thumb learned here:

- **`usedforsecurity=False`** clears B324 for any non-security hash (cache keys, dedup, ETags).
- **Bound parameters** are the only real fix for B608 — never hand-escape SQL.
- For **B310**, the `OpenerDirector`-with-only-http/https-handlers pattern is both a real SSRF/`file://` mitigation *and* AST-clean, so you don't depend on `# nosec`.
- For **XML**, if you can take the dependency, `defusedxml` is the Bandit-blessed fix; otherwise refuse `DOCTYPE` and justify a `# nosec`.
- `# nosec B<id>` (space, then the ID) suppresses one check on that line; standard Bandit honours it, but treat it as a last resort and always pair it with a real mitigation + a comment saying why.

detect-secrets

Keep **all** credentials in the environment or a secret manager — never in code, config, or the example flows. niva reads tokens (e.g. `NIVA_NTFY_TOKEN`) only from the environment and never logs or echoes them, so the scan is clean.

---

7. Pre-publish checklist

- ☐ `qgisMaximumVersion=4.99`, **no `supportsQt6`** field
- ☐ Qt imports via `qgis.PyQt`; `QAction`-style relocations handled; tested on a QGIS 4 build
- ☐ `version` bumped (unique per upload); `changelog` line added
- ☐ License consistent everywhere and **bundled in the zip**
- ☐ `tags`, `description`, `about`, `repository`, `tracker`, `homepage` set
- ☐ Icon present (PNG referenced by `icon=`)
- ☐ **`bandit -r` on the built tree = 0 MEDIUM/HIGH**, and `detect-secrets scan` = 0 (see §6)
- ☐ Zip has exactly one identifier-named top folder, no `.pyc`, self-contained
- ☐ Installed from ZIP in a clean QGIS (3.x **and** 4.x) and it loads + runs
- ☐ Release cut (tag + GitHub Release with the zip attached)

---

niva-specific extras

- **`niva validate <flow.niva>`** lints flows offline; run it (and the test suite) before releasing.

- Versions are kept in lockstep across `pyproject.toml`, `niva/__init__.py`, and `plugin/metadata.txt`; `CHANGELOG.md` is the full history and `metadata.txt`'s `changelog=` mirrors the top 1–2 entries.

Quick start

Install and run niva — inside QGIS (no install needed) or as a CLI on QGIS's own Python.

> **Dependencies:** the only requirement is **QGIS 3.22+/4.x** (niva runs in its Python) — core geoprocessing needs nothing else. Point clouds (`pdalcli:`/`pdal:`) need a PDAL backend (`pdal_wrench`) — bundled in QGIS on Windows/macOS, a one-line conda install on Linux: see the **point-cloud setup guide**. SAGA, OTB, LAStools, `notify`/`email`, and editor highlighting are **optional** add-ons; see **Dependencies — required & optional**.

In QGIS — no install needed (easiest)

1. Get `niva_qgis.zip` — download it from the latest release, or build it with `plugin/build_plugin.sh`.
2. In QGIS: **Plugins ▸ Manage and Install Plugins ▸ Install from ZIP** → pick the zip. (Enable *"Show also experimental plugins"* if it doesn't appear.)
3. Click the **niva** toolbar button, type a flow, hit **Run** — results land on the map. The plugin bundles niva, so there's no pip step on Windows, macOS, or Linux.

As a package — CLI + Python

Install into **QGIS's own Python** — this is what makes `niva run` *execute*, because that interpreter is the one with PyQGIS:

```
<qgis-python> -m pip install qgis-niva                  # pip, into QGIS's Python
uv pip install --python <qgis-python> qgis-niva         # …or uv, targeting that same
 ↪ interpreter
# dev/source install:  <qgis-python> -m pip install
 ↪ git+https://github.com/johnzastrow/niva.git
```

The deciding factor is the **target interpreter**, not the tool: both commands above land niva in QGIS's Python, so both execute. By contrast `uv tool install` / `uvx` / `pipx` build their *own* isolated Python (no PyQGIS) — great for offline authoring, but they can't run flows. See the note below and the FAQ.

> **On PyPI** as **`qgis-niva`** (the import package and `niva` command stay `niva`, like `scikit-learn` → `sklearn`): `pip install qgis-niva`, or with uv: `uv tool install qgis-niva`, or run it with no install via `uvx --from qgis-niva niva …`. Add the rich REPL/TUI with the extra: `qgis-niva[cli]`.
>
> **One caveat with `uv tool` / `uvx` / `pipx`:** they install niva into their *own* isolated Python, which has **no PyQGIS** — so that install is the **offline** CLI (validate / explain / search / setup / plan / manifest / repl / export) only. To *execute* flows, niva has to run on QGIS's own Python: install the **plugin** (runs in QGIS) or `pip install qgis-niva` into <qgis-python> as above. Full breakdown in the FAQ.

Ubuntu (system QGIS) — install for all users or one user

On Ubuntu/Debian, QGIS uses the **system Python** (e.g. `/usr/bin/python3.14`; its PyQGIS bindings live at `/usr/share/qgis/python`, added to the path at runtime). Confirm the exact interpreter in the QGIS **Python Console**:

```
import sys; print(sys.executable)      # → /usr/bin/python3.14 on Ubuntu
```

That interpreter is **externally-managed** (PEP 668), so every install below passes `--break-system-packages` — safe here because niva is **pure-Python with zero runtime dependencies** (it adds nothing to the system but niva itself).

All users (system-wide, needs `sudo`) — installs into the interpreter's global `site-packages`, so the `niva` command and `import niva` work for every account and inside QGIS's Python Console:

```
# with uv (recommended): --system installs into the interpreter, not a venv
sudo uv pip install --python /usr/bin/python3.14 --system --break-system-packages qgis-niva
# …or with pip:
sudo /usr/bin/python3.14 -m pip install --break-system-packages qgis-niva
```

This lands the package in `/usr/local/lib/python3.14/dist-packages/niva` and the launcher at `/usr/local/bin/niva` (root-owned, on everyone's PATH). Verify from any directory:

```
niva --version            # or:  /usr/bin/python3.14 -c "import niva;
 ↪  print(niva.__version__)"
```

One user (no `sudo`) — installs into your personal user-site, which the same interpreter also reads (nothing system-wide changes):

```
/usr/bin/python3.14 -m pip install --user --break-system-packages qgis-niva
```

This lands niva in `~/.local/lib/python3.14/site-packages` and the launcher at `~/.local/bin/niva`. Make sure that bin dir is on your PATH (add to `~/.bashrc` if needed):

```
export PATH="$HOME/.local/bin:$PATH"
```

Either way, because niva is installed **into QGIS's own interpreter**, `niva run` *executes* flows (not just the offline commands) — and since **0.55.1** niva auto-discovers QGIS's bindings (`/usr/share/qgis/python` on Ubuntu), so **the `niva` command just works from any directory** with no PYTHONPATH needed. Verify: `niva info` should report the Processing providers (not `Backend: mock`). Upgrade later with the same command plus `--upgrade`; uninstall with `… pip uninstall qgis-niva`. Add the rich REPL/TUI with the `[cli]` extra: `… qgis-niva[cli]`.

> If your QGIS lives somewhere non-standard and `niva info` still shows `Backend: mock`, point niva at the bindings once via `export NIVA_QGIS_PYTHONPATH=/path/to/qgis/python` (the dir containing the `qgis/` package — find it in the QGIS Python Console with `import qgis, os; print(os.path.dirname(os.path.dirname(qgis.__file__)))`).

Make `niva` a terminal command

If you `pip installed qgis-niva` into QGIS's Python (above), you already have a `niva` command — skip this section. It's only for running niva **straight from a git clone** (no install), where you need to point `niva` at QGIS's Python and add the repo to PYTHONPATH. The portable way is a small alias/function in your shell profile. Substitute **`<qgis-python>`** — the full path to QGIS's Python; find it in the QGIS **Python Console** with `import sys; print(sys.executable)` — and **`/path/to/niva`** (your repo root). `QT_QPA_PLATFORM=offscreen` keeps it headless.

Linux — add to `~/.bashrc` (zsh: `~/.zshrc`), then `source ~/.bashrc`:

```
alias niva='QT_QPA_PLATFORM=offscreen
 ↪  PYTHONPATH=/path/to/niva:/usr/share/qgis/python:/usr/lib/python3/dist-packages
 ↪  <qgis-python> -m niva.cli.main'
# pip-installed (no clone)? drop the /path/to/niva: prefix from PYTHONPATH.
```

```
# Concrete: alias niva='QT_QPA_PLATFORM=offscreen
 ↪  PYTHONPATH=/home/me/Github/niva:/usr/share/qgis/python:/usr/lib/python3/dist-packages
 ↪  /usr/bin/python3.12 -m niva.cli.main'
```

macOS — add to `~/.zshrc`, then `source ~/.zshrc`:

```
# QGIS.app's bundled Python already imports the bindings — no PYTHONPATH needed when
 ↪  pip-installed.
alias niva='QT_QPA_PLATFORM=offscreen /Applications/QGIS.app/Contents/MacOS/bin/python3 -m
 ↪  niva.cli.main'
# running from a clone? add the repo: PYTHONPATH=/path/to/niva
 ↪  /Applications/QGIS.app/Contents/MacOS/bin/python3 -m niva.cli.main
```

Windows (Windows Terminal / PowerShell) — add to `$PROFILE` (open it with `notepad $PROFILE`), then reload with `. $PROFILE`:

```
function niva {
  $env:QT_QPA_PLATFORM = 'offscreen'
  & 'C:\OSGeo4W\bin\python-qgis.bat' -m niva.cli.main @args
}
# standalone installer instead of OSGeo4W? use e.g. 'C:\Program Files\QGIS
 ↪  3.40\bin\python-qgis.bat'.
```

`python-qgis.bat` sets up the QGIS environment itself, so no PYTHONPATH is needed on Windows. After reloading your profile, `niva run myflow.niva` works from any terminal. More detail (finding QGIS's Python, troubleshooting) is in the **User Guide**.

Then run flows from the shell or Python:

```
niva run myflow.niva                                # execute a .niva file
niva "load a.gpkg | buffer 100m | save b.gpkg"      # a one-liner
niva describe buffer                                # how a verb maps to a QGIS algorithm
niva "load a.gpkg | buffer 100m | save b.gpkg" --dry-run   # validate — no QGIS needed
```

```
import niva
niva.flow('load "data.gpkg|layername=roads" | buffer 100m dissolve | save out.gpkg')
```

> A GeoPackage holds many layers — name one with `|layername=…`. Databases: `load @conn.table`, `sql @conn "SELECT …"`, and write back with `save @conn.table` or a non-SELECT `sql @conn "…"` (credentials stay in QGIS).

QGIS algorithm appendix

Every QGIS Processing algorithm reachable from niva via `run <id> KEY=value …` — **878 algorithms** across 7 providers (QGIS 4.0.3). For each: its parameters (with types, defaults, and enum options), description, outputs, a worked **Example usage**, and which niva **alias verb** (if any) wraps it (★).

This is auto-generated — regenerate with `scripts/gen_algorithms.py` after a QGIS upgrade. Most users only need the 45 alias verbs; this appendix is for reaching everything else through `run`. Discover one live with `niva describe <id>`.

Provider	Algorithms	niva alias verbs	Reference
`3d:`	1	0	3d.md
`gdal:`	59	5	gdal.md
`grass:`	307	0	grass.md
`native:`	339	40	native.md
`otb:`	109	0	otb.md
`pdal:`	24	0	pdal.md
`qgis:`	39	0	qgis.md
Total	**878**	**45**	

> **Beyond QGIS providers:** niva's native-CLI harness adds `pdalcli:<command>` (PDAL on raw LAS/LAZ/COPC) and `saga:<library>:<tool>` (`saga_cmd`). These are not QGIS algorithms, so they are not listed above — see the harness reference.

native: algorithms

native — QGIS native (C++) algorithms — 339 algorithms (QGIS 4.0.3). Auto-generated by `scripts/gen_algorithms.py`; see the index.

Call any of these with `run <id> KEY=value …` (the **Name** column is the KEY); each entry includes a worked **Example usage**. A ★ marks an algorithm with a friendly niva alias verb.

`native:addautoincrementalfield` — Add autoincremental field

group: Vector table

This algorithm adds a new integer field to a vector layer, with a sequential value for each feature.

This field can be used as a unique ID for features in the layer. The new attribute is not added to the input layer but a new layer is generated instead.

The initial starting value for the incremental series can be specified.

Specifying an optional modulus value will restart the count to START whenever the field value reaches the modulus value.

Optionally, grouping fields can be specified. If group fields are present, then the field value will be reset for each combination of these group field values.

The sort order for features may be specified, if so, then the incremental field will respect this sort order.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`FIELD_NAME`	string	no	AUTO	Field name
`START`	number	no	0	Start values at
`MODULUS`	number	no	0	Modulus value
`GROUP_FIELDS`	field	no	—	Group values by
`SORT_EXPRESSION *(adv)*`	expression	no	—	Sort expression
`SORT_ASCENDING *(adv)*`	boolean	no	True	Sort ascending
`SORT_NULLS_FIRST *(adv)*`	boolean	no	False	Sort nulls first
`OUTPUT`	sink	yes	—	Incremented

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:addautoincrementalfield INPUT=input.gpkg FIELD_NAME=AUTO START=0 MODULUS=0
 ↪  GROUP_FIELDS=field1 OUTPUT=output.gpkg
```

This calls **Add autoincremental field**: `INPUT=input.gpkg` — Input layer; `FIELD_NAME=AUTO` — Field name; `START=0` — Start values at; `MODULUS=0` — Modulus value; `GROUP_FIELDS=field1` — Group values by; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:addfieldtoattributestable` — Add field to attributes table

group: Vector table

This algorithm adds a new attribute to a vector layer.

The name and characteristics of the attribute are defined as parameters. The new attribute is not added to the input layer but a new layer is generated instead.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD_NAME	string	yes	—	Field name
FIELD_TYPE	enum	no	0	Field type — options: 0=Integer (32 bit), 1=Decimal (double), 2=Text (string), 3=Boolean, 4=Date, 5=Time, 6=Date & Time, 7=Binary Object (BLOB), 8=String List, 9=Integer List, 10=Decimal (double) List
FIELD_LENGTH	number	no	10	Field length
FIELD_PRECISION	number	no	0	Field precision
FIELD_ALIAS	string	no	—	Field alias
FIELD_COMMENT	string	no	—	Field comment
OUTPUT	sink	yes	—	Added

Outputs: OUTPUT (outputVector)

Example usage

```
run native:addfieldtoattributestable INPUT=input.gpkg FIELD_NAME=value FIELD_TYPE=0
↪ FIELD_LENGTH=10 FIELD_PRECISION=0 FIELD_ALIAS=value FIELD_COMMENT=value
↪ OUTPUT=output.gpkg
```

This calls **Add field to attributes table**: INPUT=input.gpkg — Input layer; FIELD_NAME=value — Field name; FIELD_TYPE=0 selects *Integer (32 bit)*; FIELD_LENGTH=10 — Field length; FIELD_PRECISION=0 — Field precision; FIELD_ALIAS=value — Field alias; FIELD_COMMENT=value — Field comment; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:addhistorymetadata` — Add history metadata

group: Metadata tools

This algorithm adds a new history entry to the layer's metadata.

Parameter	Type	Required	Default	Description
INPUT	layer	yes	—	Layer
HISTORY	string	yes	—	History entry

Outputs: OUTPUT (outputLayer)

Example usage

```
run native:addhistorymetadata INPUT=input.gpkg HISTORY=value
```

This calls **Add history metadata**: INPUT=input.gpkg — Layer; HISTORY=value — History entry. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:adduniquevalueindexfield` — Add unique value index field

group: Vector table

This algorithm takes a vector layer and an attribute and adds a new numeric field. Values in this field correspond to values in the specified attribute, so features with the same value for the

attribute will have the same value in the new numeric field. This creates a numeric equivalent of the specified attribute, which defines the same classes.

The new attribute is not added to the input layer but a new layer is generated instead.

Optionally, a separate table can be output which contains a summary of the class field values mapped to the new unique numeric value.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	yes	—	Class field
FIELD_NAME	string	no	NUM_FIELD	Output field name
OUTPUT	sink	no	—	Layer with index field
SUMMARY_OUTPUT	sink	no	—	Class summary

Outputs: OUTPUT (outputVector), SUMMARY_OUTPUT (outputVector)

Example usage

```
run native:adduniquevalueindexfield INPUT=input.gpkg FIELD=field1 FIELD_NAME=NUM_FIELD
 ↪  OUTPUT=output.gpkg
```

This calls **Add unique value index field**: INPUT=input.gpkg — Input layer; FIELD=field1 — Class field; FIELD_NAME=NUM_FIELD — Output field name; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:addxyfields — Add X/Y fields to layer

group: Vector table

This algorithm adds X and Y (or latitude/longitude) fields to a point layer. The X/Y fields can be calculated in a different CRS to the layer (e.g. creating latitude/longitude fields for a layer in a project CRS).

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
CRS	crs	no	EPSG:4326	Coordinate system
PREFIX	string	no	—	Field prefix
OUTPUT	sink	yes	—	Added fields

Outputs: OUTPUT (outputVector)

Example usage

```
run native:addxyfields INPUT=input.gpkg CRS=EPSG:6346 PREFIX=value OUTPUT=output.gpkg
```

This calls **Add X/Y fields to layer**: INPUT=input.gpkg — Input layer; CRS=EPSG:6346 — Coordinate system; PREFIX=value — Field prefix; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:affinetransform — Affine transform

group: Vector geometry

This algorithm applies an affine transformation to the geometries from a layer. Affine transformations can include translation, scaling and rotation. The operations are performed in a scale, rotation, translation order.

Z and M values present in the geometry can also be translated and scaled independently.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
DELTA_X	distance	no	0.0	Translation (x-axis)
DELTA_Y	distance	no	0.0	Translation (y-axis)
DELTA_Z	number	no	0.0	Translation (z-axis)
DELTA_M	number	no	0.0	Translation (m values)
SCALE_X	number	no	1.0	Scale factor (x-axis)
SCALE_Y	number	no	1.0	Scale factor (y-axis)
SCALE_Z	number	no	1.0	Scale factor (z-axis)
SCALE_M	number	no	1.0	Scale factor (m values)
ROTATION_Z	number	no	0.0	Rotation around z-axis (degrees counter-clockwise)
OUTPUT	sink	yes	—	Transformed

Outputs: OUTPUT (outputVector)

Example usage

```
run native:affinetransform INPUT=input.gpkg DELTA_X=0 DELTA_Y=0 DELTA_Z=0 DELTA_M=0
↪  SCALE_X=1 SCALE_Y=1 OUTPUT=output.gpkg
```

This calls **Affine transform**: INPUT=input.gpkg — Input layer; DELTA_X=0 — Translation (x-axis); DELTA_Y=0 — Translation (y-axis); DELTA_Z=0 — Translation (z-axis); DELTA_M=0 — Translation (m values); SCALE_X=1 — Scale factor (x-axis); SCALE_Y=1 — Scale factor (y-axis); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:aggregate — Aggregate

group: Vector geometry

This algorithm takes a vector or table layer and aggregates features based on a group by expression. Features for which group by expression return the same value are grouped together.

It is possible to group all source features together using constant value in group by parameter, example: NULL.

It is also possible to group features using multiple fields using Array function, example: Array(”Field1”, ”Field2”).

Geometries (if present) are combined into one multipart geometry for each group.

Output attributes are computed depending on each given aggregate definition.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
GROUP_BY	expression	no	NULL	Group by expression (NULL to group all features)
AGGREGATES	aggregates	yes	—	Aggregates
OUTPUT	sink	yes	—	Aggregated

Outputs: OUTPUT (outputVector)

Example usage

```
run native:aggregate INPUT=input.gpkg AGGREGATES=… GROUP_BY="value > 0" OUTPUT=output.gpkg
```

This calls **Aggregate**: INPUT=input.gpkg — Input layer; AGGREGATES=… — Aggregates; GROUP_BY="value > 0" — Group by expression (NULL to group all features); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:alignrasters` — Align rasters

group: Raster tools

This algorithm aligns rasters by resampling them to the same cell size and reprojecting to the same CRS.

Parameter	Type	Required	Default	Description
LAYERS	alignrasterlayers	yes	—	Input layers
REFERENCE_LAYER	raster	yes	—	Reference layer
CRS	crs	no	—	Override reference CRS
CELL_SIZE_X	number	no	—	Override reference cell size X
CELL_SIZE_Y	number	no	—	Override reference cell size Y
GRID_OFFSET_X	number	no	—	Override reference grid offset X
GRID_OFFSET_Y	number	no	—	Override reference grid offset Y
EXTENT	extent	no	—	Clip to extent

Outputs: OUTPUT_LAYERS (outputMultilayer)

Example usage

```
run native:alignrasters LAYERS=… REFERENCE_LAYER=dem.tif CRS=EPSG:6346 CELL_SIZE_X=10
↪  CELL_SIZE_Y=10 GRID_OFFSET_X=10 GRID_OFFSET_Y=10
```

This calls **Align rasters**: LAYERS=… — Input layers; REFERENCE_LAYER=dem.tif — Reference layer; CRS=EPSG:6346 — Override reference CRS; CELL_SIZE_X=10 — Override reference cell size X; CELL_SIZE_Y=10 — Override reference cell size Y; GRID_OFFSET_X=10 — Override reference grid offset X; GRID_OFFSET_Y=10 — Override reference grid offset Y. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:alignsingleraster` — Align raster

group: Raster tools

This algorithm aligns raster by resampling it to the same cell size and reprojecting to the same CRS as a reference raster.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
RESAMPLING_METHOD	enum	no	0	Resampling method — options: 0=Nearest Neighbour, 1=Bilinear (2x2 Kernel), 2=Cubic (4x4 Kernel), 3=Cubic B-Spline (4x4 Kernel), 4=Lanczos (6x6 Kernel), 5=Average, 6=Mode, 7=Maximum, 8=Minimum, 9=Median, 10=First Quartile (Q1), 11=Third Quartile (Q3)
RESCALE	boolean	no	False	Rescale values according to the cell size
REFERENCE_LAYER	raster	yes	—	Reference layer
CRS	crs	no	—	Override reference CRS
CELL_SIZE_X	number	no	—	Override reference cell size X
CELL_SIZE_Y	number	no	—	Override reference cell size Y
GRID_OFFSET_X	number	no	—	Override reference grid offset X
GRID_OFFSET_Y	number	no	—	Override reference grid offset Y
EXTENT	extent	no	—	Clip to extent
OUTPUT	rasterDestination	yes	—	Aligned raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:alignsingleraster INPUT=dem.tif REFERENCE_LAYER=dem.tif RESAMPLING_METHOD=0
↪  RESCALE=false CRS=EPSG:6346 CELL_SIZE_X=10 CELL_SIZE_Y=10 OUTPUT=output.tif
```

This calls **Align raster**: `INPUT=dem.tif` — Input layer; `REFERENCE_LAYER=dem.tif` — Reference layer; `RESAMPLING_METHOD=0` selects *Nearest Neighbour*; `RESCALE=false` — Rescale values according to the cell size; `CRS=EPSG:6346` — Override reference CRS; `CELL_SIZE_X=10` — Override reference cell size X; `CELL_SIZE_Y=10` — Override reference cell size Y; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:angletonearest` — Align points to features

group: Cartography

This algorithm calculates the rotation required to align point features with their nearest feature from another reference layer. A new field is added to the output layer which is filled with the angle (in degrees, clockwise) to the nearest reference feature.

Optionally, the output layer's symbology can be set to automatically use the calculated rotation field to rotate marker symbols.

If desired, a maximum distance to use when aligning points can be set, to avoid aligning isolated points to distant features.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`REFERENCE_LAYER`	source	yes	—	Reference layer
`MAX_DISTANCE`	distance	no	—	Maximum distance to consider
`FIELD_NAME`	string	no	rotation	Angle field name
`APPLY_SYMBOLOGY`	boolean	no	True	Automatically apply symbology
`OUTPUT`	sink	yes	—	Aligned layer

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:angletonearest INPUT=input.gpkg REFERENCE_LAYER=input.gpkg MAX_DISTANCE=10
 ↪  FIELD_NAME=rotation APPLY_SYMBOLOGY=true OUTPUT=output.gpkg
```

This calls **Align points to features**: `INPUT=input.gpkg` — Input layer; `REFERENCE_LAYER=input.gpkg` — Reference layer; `MAX_DISTANCE=10` — Maximum distance to consider; `FIELD_NAME=rotation` — Angle field name; `APPLY_SYMBOLOGY=true` — Automatically apply symbology; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:antimeridiansplit` — Geodesic line split at antimeridian

group: Vector geometry

This algorithm splits a line into multiple geodesic segments, whenever the line crosses the antimeridian (±180 degrees longitude).

Splitting at the antimeridian helps the visual display of the lines in some projections. The returned geometry will always be a multi-part geometry.

Whenever line segments in the input geometry cross the antimeridian, they will be split into two segments, with the latitude of the breakpoint being determined using a geodesic line connecting the points either side of this segment. The current project ellipsoid setting will be used when calculating this breakpoint.

If the input geometry contains M or Z values, these will be linearly interpolated for the new vertices created at the antimeridian.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Split

Outputs: OUTPUT (outputVector)

Example usage

```
run native:antimeridiansplit INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Geodesic line split at antimeridian**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:approximatemedialaxis — Approximate medial axis

group: Vector geometry

The Approximate Medial Axis algorithm generates a simplified skeleton of a shape by approximating its medial axis.

The output is a collection of lines that follow the central structure of the shape. The result is a thin, stable set of curves that capture the main topology while ignoring noise.

This algorithm ignores the Z dimensions. If the geometry is 3D, the approximate medial axis will be calculated from its 2D projection.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Medial axis

Outputs: OUTPUT (outputVector)

Example usage

```
run native:approximatemedialaxis INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Approximate medial axis**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:arrayoffsetlines — Array of offset (parallel) lines

group: Vector creation

This algorithm creates copies of line features in a layer, by creating multiple offset versions of each feature. Each copy is offset by a preset distance.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
COUNT	number	no	10	Number of features to create
OFFSET	distance	no	1.0	Offset step distance
SEGMENTS *(adv)*	number	no	8	Segments
JOIN_STYLE *(adv)*	enum	no	0	Join style — options: 0=Round, 1=Miter, 2=Bevel
MITER_LIMIT *(adv)*	number	no	2	Miter limit
OUTPUT	sink	yes	—	Offset lines

Outputs: OUTPUT (outputVector)

Example usage

```
run native:arrayoffsetlines INPUT=input.gpkg COUNT=10 OFFSET=1 OUTPUT=output.gpkg
```

This calls **Array of offset (parallel) lines**: `INPUT=input.gpkg` — Input layer; `COUNT=10` — Number of features to create; `OFFSET=1` — Offset step distance; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:arraytranslatedfeatures` — Array of translated features

group: Vector creation

This algorithm creates copies of features in a layer, by creating multiple translated versions of each feature. Each copy is incrementally displaced by a preset amount in the x/y/z/m axis.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`COUNT`	number	no	10	Number of features to create
`DELTA_X`	distance	no	0.0	Step distance (x-axis)
`DELTA_Y`	distance	no	0.0	Step distance (y-axis)
`DELTA_Z`	number	no	0.0	Step distance (z-axis)
`DELTA_M`	number	no	0.0	Step distance (m values)
`OUTPUT`	sink	yes	—	Translated

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:arraytranslatedfeatures INPUT=input.gpkg COUNT=10 DELTA_X=0 DELTA_Y=0 DELTA_Z=0
↪  DELTA_M=0 OUTPUT=output.gpkg
```

This calls **Array of translated features**: `INPUT=input.gpkg` — Input layer; `COUNT=10` — Number of features to create; `DELTA_X=0` — Step distance (x-axis); `DELTA_Y=0` — Step distance (y-axis); `DELTA_Z=0` — Step distance (z-axis); `DELTA_M=0` — Step distance (m values); `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:aspect` — Aspect

group: Raster terrain analysis

This algorithm calculates the aspect of the Digital Terrain Model in input.

The final aspect raster layer contains values from 0 to 360 that express the slope direction: starting from North (0°) and continuing clockwise.

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Elevation layer
`Z_FACTOR`	number	no	1.0	Z factor
`NODATA *(adv)*`	number	no	-9999.0	Output NoData value
`CREATION_OPTIONS *(adv)*`	string	no	—	Creation options
`OUTPUT`	rasterDestination	yes	—	Aspect

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run native:aspect INPUT=dem.tif Z_FACTOR=1 OUTPUT=output.tif
```

This calls **Aspect**: `INPUT=dem.tif` — Elevation layer; `Z_FACTOR=1` — Z factor; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:assignprojection` — Assign projection

group: Vector general

This algorithm assigns a new projection to a vector layer. It creates a new layer with the exact same features and geometries as the input one, but assigned to a new CRS. E.g. the geometries are not reprojected, they are just assigned to a different CRS. This algorithm can be used to repair layers which have been assigned an incorrect projection.

Attributes are not modified by this algorithm.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
CRS	crs	no	EPSG:4326	Assigned CRS
OUTPUT	sink	yes	—	Assigned CRS

Outputs: OUTPUT (outputVector)

Example usage

```
run native:assignprojection INPUT=input.gpkg CRS=EPSG:6346 OUTPUT=output.gpkg
```

This calls **Assign projection**: INPUT=input.gpkg — Input layer; CRS=EPSG:6346 — Assigned CRS; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:atlaslayouttoimage` — Export atlas layout as image

group: Cartography

This algorithm outputs an atlas layout to a set of image files (e.g. PNG or JPEG images).

If a coverage layer is set, the selected layout's atlas settings exposed in this algorithm will be overwritten. In this case, an empty filter or sort by expression will turn those settings off.

Parameter	Type	Required	Default	Description
LAYOUT	layout	yes	—	Atlas layout
COVERAGE_LAYER	vector	no	—	Coverage layer
FILTER_EXPRESSION	expression	no	—	Filter expression
SORTBY_EXPRESSION	expression	no	—	Sort expression
SORTBY_REVERSE	boolean	no	False	Reverse sort order (used when a sort expression is provided)
FILENAME_EXPRESSION	expression	no	'output_'||@atlas_featurenumber	Output filename expression
FOLDER	file	yes	—	Output folder
LAYERS *(adv)*	multilayer	no	—	Map layers to assign to unlocked map item(s)
EXTENSION *(adv)*	enum	no	23	Image format — options: 0=avif, 1=bmp, 2=bw, 3=cur, 4=dds, 5=eps, 6=epsf, 7=epsi, 8=exr, 9=heic, 10=heif, 11=icns, 12=ico, 13=j2k, 14=jfif, 15=jp2, 16=jpeg, 17=jpg, 18=jxl, 19=pbm, 20=pcx, 21=pgm, 22=pic, 23=png, 24=ppm, 25=qoi, 26=rgb, 27=rgba, 28=sgi, 29=tga, 30=tif, 31=tiff, 32=wbmp, 33=webp, 34=xbm, 35=xpm
DPI *(adv)*	number	no	—	DPI (leave blank for default layout DPI)
GEOREFERENCE *(adv)*	boolean	no	True	Generate world file
INCLUDE_METADATA *(adv)*	boolean	no	True	Export RDF metadata (title, author, etc.)
ANTIALIAS *(adv)*	boolean	no	True	Enable antialiasing

Example usage

```
run native:atlaslayouttoimage LAYOUT=… FOLDER=input.dat COVERAGE_LAYER=input.gpkg
 ↪  FILTER_EXPRESSION="value > 0" SORTBY_EXPRESSION="value > 0" SORTBY_REVERSE=false
 ↪  FILENAME_EXPRESSION="value > 0"
```

This calls **Export atlas layout as image**: `LAYOUT=…` — Atlas layout; `FOLDER=input.dat` — Output folder; `COVERAGE_LAYER=input.gpkg` — Coverage layer; `FILTER_EXPRESSION="value > 0"` — Filter expression; `SORTBY_EXPRESSION="value > 0"` — Sort expression; `SORTBY_REVERSE=false` — Reverse sort order (used when a sort expression is provided); `FILENAME_EXPRESSION="value > 0"` — Output filename expression. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:atlaslayouttomultiplepdf` — Export atlas layout as PDF (multiple files)

group: Cartography

This algorithm outputs an atlas layout to multiple PDF files.

If a coverage layer is set, the selected layout's atlas settings exposed in this algorithm will be overwritten. In this case, an empty filter or sort by expression will turn those settings off.

Parameter	Type	Required	Default	Description
`LAYOUT`	layout	yes	—	Atlas layout
`COVERAGE_LAYER`	vector	no	—	Coverage layer
`FILTER_EXPRESSION`	expression	no	—	Filter expression
`SORTBY_EXPRESSION`	expression	no	—	Sort expression
`SORTBY_REVERSE`	boolean	no	False	Reverse sort order (used when a sort expression is provided)
`LAYERS *(adv)*`	multilayer	no	—	Map layers to assign to unlocked map item(s)
`DPI *(adv)*`	number	no	—	DPI (leave blank for default layout DPI)
`FORCE_VECTOR *(adv)*`	boolean	no	False	Always export as vectors
`FORCE_RASTER *(adv)*`	boolean	no	False	Always export as raster
`GEOREFERENCE *(adv)*`	boolean	no	True	Append georeference information
`INCLUDE_METADATA *(adv)*`	boolean	no	True	Export RDF metadata (title, author, etc.)
`DISABLE_TILED *(adv)*`	boolean	no	False	Disable tiled raster layer exports
`SIMPLIFY *(adv)*`	boolean	no	True	Simplify geometries to reduce output file size
`TEXT_FORMAT *(adv)*`	enum	no	0	Text export — options: 0=Always Export Text as Paths (Recommended), 1=Always Export Text as Text Objects, 2=Prefer Exporting Text as Text Objects
`IMAGE_COMPRESSION *(adv)*`	enum	no	0	Image compression — options: 0=Lossy (JPEG), 1=Lossless
`OUTPUT_FILENAME`	expression	no	—	Output filename
`OUTPUT_FOLDER`	file	yes	—	Output folder

Example usage

```
run native:atlaslayouttomultiplepdf LAYOUT=… OUTPUT_FOLDER=input.dat
 ↪  COVERAGE_LAYER=input.gpkg FILTER_EXPRESSION="value > 0" SORTBY_EXPRESSION="value > 0"
 ↪  SORTBY_REVERSE=false OUTPUT_FILENAME="value > 0"
```

This calls **Export atlas layout as PDF (multiple files)**: `LAYOUT=…` — Atlas layout; `OUTPUT_FOLDER=input.dat` — Output folder; `COVERAGE_LAYER=input.gpkg` — Coverage layer; `FILTER_EXPRESSION="value > 0"` — Filter expression; `SORTBY_EXPRESSION="value > 0"` — Sort expression; `SORTBY_REVERSE=false` — Reverse sort order (used when a sort expression is provided); `OUTPUT_FILENAME="value > 0"` — Output filename. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:atlaslayouttopdf — Export atlas layout as PDF (single file)

group: Cartography

This algorithm outputs an atlas layout as a single PDF file.

If a coverage layer is set, the selected layout's atlas settings exposed in this algorithm will be overwritten. In this case, an empty filter or sort by expression will turn those settings off.

Parameter	Type	Required	Default	Description
LAYOUT	layout	yes	—	Atlas layout
COVERAGE_LAYER	vector	no	—	Coverage layer
FILTER_EXPRESSION	expression	no	—	Filter expression
SORTBY_EXPRESSION	expression	no	—	Sort expression
SORTBY_REVERSE	boolean	no	False	Reverse sort order (used when a sort expression is provided)
LAYERS *(adv)*	multilayer	no	—	Map layers to assign to unlocked map item(s)
DPI *(adv)*	number	no	—	DPI (leave blank for default layout DPI)
FORCE_VECTOR *(adv)*	boolean	no	False	Always export as vectors
FORCE_RASTER *(adv)*	boolean	no	False	Always export as raster
GEOREFERENCE *(adv)*	boolean	no	True	Append georeference information
INCLUDE_METADATA *(adv)*	boolean	no	True	Export RDF metadata (title, author, etc.)
DISABLE_TILED *(adv)*	boolean	no	False	Disable tiled raster layer exports
SIMPLIFY *(adv)*	boolean	no	True	Simplify geometries to reduce output file size
TEXT_FORMAT *(adv)*	enum	no	0	Text export — options: 0=Always Export Text as Paths (Recommended), 1=Always Export Text as Text Objects, 2=Prefer Exporting Text as Text Objects
IMAGE_COMPRESSION *(adv)*	enum	no	0	Image compression — options: 0=Lossy (JPEG), 1=Lossless
OUTPUT	fileDestination	yes	—	PDF file

Outputs: OUTPUT (outputFile)

Example usage

```
run native:atlaslayouttopdf LAYOUT=… COVERAGE_LAYER=input.gpkg FILTER_EXPRESSION="value >
 ↪  0" SORTBY_EXPRESSION="value > 0" SORTBY_REVERSE=false OUTPUT=output.txt
```

This calls **Export atlas layout as PDF (single file)**: LAYOUT=… — Atlas layout; COVERAGE_LAYER=input.gpkg — Coverage layer; FILTER_EXPRESSION="value > 0" — Filter expression; SORTBY_EXPRESSION="value > 0" — Sort expression; SORTBY_REVERSE=false — Reverse sort order (used when a sort expression is provided); OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:b3dmtogltf — Convert B3DM to GLTF

group: 3D Tiles

This algorithm converts files from the legacy B3DM format to GLTF.

Parameter	Type	Required	Default	Description
INPUT	file	yes	—	Input B3DM
OUTPUT	fileDestination	yes	—	Output file

Outputs: OUTPUT (outputFile)

Example usage

```
run native:b3dmtogltf INPUT=input.dat OUTPUT=output.txt
```

This calls **Convert B3DM to GLTF**: `INPUT=input.dat` — Input B3DM; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:basicstatisticsforfields` — Basic statistics for fields

group: Vector analysis

This algorithm generates basic statistics from the analysis of values in a field in the attribute table of a vector layer. Numeric, date, time and string fields are supported. The statistics returned will depend on the field type.

Parameter	Type	Required	Default	Description
`INPUT_LAYER`	source	yes	—	Input layer
`FIELD_NAME`	field	yes	—	Field to calculate statistics on
`OUTPUT`	sink	no	—	Statistics
`OUTPUT_HTML_FILE`	fileDestination	no	—	Statistics report

Outputs: `OUTPUT` (outputVector), `OUTPUT_HTML_FILE` (outputHtml), `COUNT` (outputNumber), `UNIQUE` (outputNumber), `EMPTY` (outputNumber), `FILLED` (outputNumber), `MIN` (outputNumber), `MAX` (outputNumber), `MIN_LENGTH` (outputNumber), `MAX_LENGTH` (outputNumber), `MEAN_LENGTH` (outputNumber), `CV` (outputNumber), `SUM` (outputNumber), `MEAN` (outputNumber), `STD_DEV` (outputNumber), `RANGE` (outputNumber), `MEDIAN` (outputNumber), `MINORITY` (outputNumber), `MAJORITY` (outputNumber), `FIRSTQUARTILE` (outputNumber), `THIRDQUARTILE` (outputNumber), `IQR` (outputNumber)

Example usage

```
run native:basicstatisticsforfields INPUT_LAYER=input.gpkg FIELD_NAME=field1
 ↪  OUTPUT=output.gpkg
```

This calls **Basic statistics for fields**: `INPUT_LAYER=input.gpkg` — Input layer; `FIELD_NAME=field1` — Field to calculate statistics on; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:batchnominatimgeocoder` — Batch Nominatim geocoder

group: Vector general

This algorithm performs batch geocoding using the Nominatim service against an input layer string field.

The output layer will have a point geometry reflecting the geocoded location as well as a number of attributes associated to the geocoded location.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`FIELD`	field	yes	—	Address field
`OUTPUT`	sink	yes	—	Geocoded

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:batchnominatimgeocoder INPUT=input.gpkg FIELD=field1 OUTPUT=output.gpkg
```

This calls **Batch Nominatim geocoder**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — Address field; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:bookmarkstolayer` — Convert spatial bookmarks to layer

group: Vector general

This algorithm creates a new layer containing polygon features for stored spatial bookmarks.

The export can be filtered to only bookmarks belonging to the current project, to all user bookmarks, or a combination of both.

Parameter	Type	Required	Default	Description
SOURCE	enum	no	[0, 1]	Bookmark source — options: 0=Project bookmarks, 1=User bookmarks
CRS	crs	no	<QgsCoordinateReferenceSystem: EPSG:4326>	Output CRS
OUTPUT	sink	yes	—	Output

Outputs: OUTPUT (outputVector)

Example usage

```
run native:bookmarkstolayer SOURCE=0 CRS=EPSG:6346 OUTPUT=output.gpkg
```

This calls **Convert spatial bookmarks to layer**: `SOURCE=0` selects *Project bookmarks*; `CRS=EPSG:6346` — Output CRS; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:boundary` — Boundary

group: Vector geometry

This algorithm returns the closure of the combinatorial boundary of the input geometries (ie the topological boundary of the geometry). For instance, a polygon geometry will have a boundary consisting of the linestrings for each ring in the polygon. Only valid for polygon or line layers.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Boundary

Outputs: OUTPUT (outputVector)

Example usage

```
run native:boundary INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Boundary**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:boundingboxes` — Bounding boxes ★

group: Vector geometry · **niva verb:** `boundingbox`

This algorithm calculates the bounding box (envelope) for each feature in an input layer.

See the 'Minimum bounding geometry' algorithm for a bounding box calculation which covers the whole layer or grouped subsets of features.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Bounds

Outputs: OUTPUT (outputVector)

Example usage

```
run native:boundingboxes INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Bounding boxes**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:buffer` — Buffer ★

group: Vector geometry · **niva verb:** `buffer`

This algorithm computes a buffer area for all the features in an input layer, using a fixed or dynamic distance.

The segments parameter controls the number of line segments to use to approximate a quarter circle when creating rounded offsets.

The end cap style parameter controls how line endings are handled in the buffer.

The join style parameter specifies whether round, miter or beveled joins should be used when offsetting corners in a line.

The miter limit parameter is only applicable for miter join styles, and controls the maximum distance from the offset curve to use when creating a mitered join.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`DISTANCE`	distance	no	10	Distance
`SEGMENTS`	number	no	5	Segments
`END_CAP_STYLE`	enum	no	0	End cap style — options: 0=Round, 1=Flat, 2=Square
`JOIN_STYLE`	enum	no	0	Join style — options: 0=Round, 1=Miter, 2=Bevel
`MITER_LIMIT`	number	no	2	Miter limit
`DISSOLVE`	boolean	no	False	Dissolve result
`SEPARATE_DISJOINT *(adv)*`	boolean	no	False	Keep disjoint results separate
`OUTPUT`	sink	yes	—	Buffered

Outputs: OUTPUT (outputVector)

Example usage

```
run native:buffer INPUT=input.gpkg DISTANCE=10 SEGMENTS=5 END_CAP_STYLE=0 JOIN_STYLE=0
↪  MITER_LIMIT=2 DISSOLVE=false OUTPUT=output.gpkg
```

This calls **Buffer**: `INPUT=input.gpkg` — Input layer; `DISTANCE=10` — Distance; `SEGMENTS=5` — Segments; `END_CAP_STYLE=0` selects *Round*; `JOIN_STYLE=0` selects *Round*; `MITER_LIMIT=2` — Miter limit; `DISSOLVE=false` — Dissolve result; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:bufferbym` — Variable width buffer (by M value)

group: Vector geometry

This algorithm creates variable width buffers along lines, using the M value of the line geometries as the diameter of the buffer at each vertex.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
SEGMENTS	number	no	16	Segments
OUTPUT	sink	yes	—	Buffered

Outputs: OUTPUT (outputVector)

Example usage

```
run native:bufferbym INPUT=input.gpkg SEGMENTS=16 OUTPUT=output.gpkg
```

This calls **Variable width buffer (by M value)**: INPUT=input.gpkg — Input layer; SEGMENTS=16 — Segments; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:calculateexpression — Calculate expression

group: Modeler tools

This algorithm calculates the result of a QGIS expression and makes it available for use in other parts of the model.

Parameter	Type	Required	Default	Description
INPUT	string	yes	—	Input

Outputs: OUTPUT (outputVariant)

Example usage

```
run native:calculateexpression INPUT=value
```

This calls **Calculate expression**: INPUT=value — Input. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:calculatevectoroverlaps — Overlap analysis

group: Vector analysis

This algorithm calculates the area and percentage cover by which features from an input layer are overlapped by features from a selection of overlay layers.

New attributes are added to the output layer reporting the total area of overlap and percentage of the input feature overlapped by each of the selected overlay layers.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
LAYERS	multilayer	yes	—	Overlay layers
OUTPUT	sink	yes	—	Overlap
GRID_SIZE *(adv)*	number	no	—	Grid size

Outputs: OUTPUT (outputVector)

Example usage

```
run native:calculatevectoroverlaps INPUT=input.gpkg LAYERS="a.gpkg;b.gpkg"
↪ OUTPUT=output.gpkg
```

This calls **Overlap analysis**: INPUT=input.gpkg — Input layer; LAYERS="a.gpkg;b.gpkg" — Overlay layers; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:categorizeusingstyle — Create categorized renderer from styles

group: Cartography

This algorithm sets a vector layer's renderer to a categorized renderer using matching symbols from a style database. If no style file is specified, symbols from the user's current style library are used instead.

The specified expression (or field name) is used to create categories for the renderer. A category will be created for each unique value within the layer.

Each category is individually matched to the symbols which exist within the specified QGIS XML style database. Whenever a matching symbol name is found, the category's symbol will be set to this matched symbol.

The matching is case-insensitive by default, but can be made case-sensitive if required.

Optionally, non-alphanumeric characters in both the category value and symbol name can be ignored while performing the match. This allows for greater tolerance when matching categories to symbols.

If desired, tables can also be output containing lists of the categories which could not be matched to symbols, and symbols which were not matched to categories.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
FIELD	expression	yes	—	Categorize using expression
STYLE	file	no	—	Style database (leave blank to use saved symbols)
CASE_SENSITIVE	boolean	no	False	Use case-sensitive match to symbol names
TOLERANT	boolean	no	False	Ignore non-alphanumeric characters while matching
NON_MATCHING_CATEGORIES	sink	no	—	Non-matching categories
NON_MATCHING_SYMBOLS	sink	no	—	Non-matching symbol names

Outputs: OUTPUT (outputVector), NON_MATCHING_CATEGORIES (outputVector), NON_MATCHING_SYMBOLS (outputVector)

Example usage

```
run native:categorizeusingstyle INPUT=input.gpkg FIELD="value > 0" STYLE=input.dat
 ↪  CASE_SENSITIVE=false TOLERANT=false
 ↪  NON_MATCHING_CATEGORIES=non_matching_categories.gpkg
```

This calls **Create categorized renderer from styles**: INPUT=input.gpkg — Input layer; FIELD="value > 0" — Categorize using expression; STYLE=input.dat — Style database (leave blank to use saved symbols); CASE_SENSITIVE=false — Use case-sensitive match to symbol names; TOLERANT=false — Ignore non-alphanumeric characters while matching; NON_MATCHING_CATEGORIES=non_matching_categories.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:cellstackpercentile — Cell stack percentile

group: Raster analysis

This algorithm generates a raster containing the cell-wise percentile value of a stack of input rasters. The percentile to return is determined by the percentile input value (ranges between 0 and 1). At each cell location, the specified percentile is obtained using the respective value from the stack of all overlaid and sorted cell values of the input rasters.

There are three methods for percentile calculation: Nearest rank Inclusive linear interpolation (PERCENTILE.INC) Exclusive linear interpolation (PERCENTILE.EXC) While the output value can stay the same for the nearest rank method (obtains the value that is nearest to the specified percentile),

the linear interpolation method return unique values for different percentiles. Both interpolation methods follow their counterpart methods implemented by LibreOffice or Microsoft Excel.

The output raster's extent and resolution is defined by a reference raster. If the input raster layers that do not match the cell size of the reference raster layer will be resampled using nearest neighbor resampling. NoData values in any of the input layers will result in a NoData cell output if the Ignore NoData parameter is not set. The output raster data type will be set to the most complex data type present in the input datasets.

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input layers
METHOD	enum	no	0	Method — options: 0=Nearest rank, 1=Inclusive linear interpolation (PERCENTILE.INC), 2=Exclusive linear interpolation (PERCENTILE.EXC)
PERCENTILE	number	no	0.25	Percentile
IGNORE_NODATA	boolean	no	True	Ignore NoData values
REFERENCE_LAYER	raster	yes	—	Reference layer
OUTPUT_NODATA_VALUE *(adv)*	number	no	-9999	Output NoData value
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber)

Example usage

```
run native:cellstackpercentile INPUT="a.gpkg;b.gpkg" REFERENCE_LAYER=dem.tif METHOD=0
 ↪  PERCENTILE=0.25 IGNORE_NODATA=true CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Cell stack percentile**: INPUT="a.gpkg;b.gpkg" — Input layers; REFERENCE_LAYER=dem.tif — Reference layer; METHOD=0 selects *Nearest rank*; PERCENTILE=0.25 — Percentile; IGNORE_NODATA=true — Ignore NoData values; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:cellstackpercentrankfromrasterlayer — Cell stack percentrank from raster layer

group: Raster analysis

This algorithm generates a raster containing the cell-wise percent rank value of a stack of input rasters based on an input value raster.

At each cell location, the current value of the value raster is used ranked among the respective values in the stack of all overlaid and sorted cell values of the input rasters. For values outside of the the stack value distribution, the algorithm returns NoData because the value cannot be ranked among the cell values.

There are two methods for percentile calculation: Inclusive linearly interpolated percent rank (PERCENTRANK.INC) Exclusive linearly interpolated percent rank (PERCENTRANK.EXC) The linear interpolation method return the unique percent rank for different values. Both interpolation methods follow their counterpart methods implemented by LibreOffice or Microsoft Excel.

The output raster's extent and resolution is defined by a reference raster. If the input raster layers that do not match the cell size of the reference raster layer will be resampled using nearest neighbor resampling. NoData values in any of the input layers will result in a NoData cell output if the Ignore NoData parameter is not set. The output raster data type will always be Float32.

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input layers
INPUT_VALUE_RASTER	raster	yes	—	Value raster layer
VALUE_RASTER_BAND	band	no	1	Value raster band
METHOD	enum	no	0	Method — options: 0=Inclusive linear interpolation (PERCENTRANK.INC), 1=Exclusive linear interpolation (PERCENTRANK.EXC)
IGNORE_NODATA	boolean	no	True	Ignore NoData values
REFERENCE_LAYER	raster	yes	—	Reference layer
OUTPUT_NODATA_VALUE *(adv)*	number	no	-9999	Output NoData value
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber)

Example usage

```
run native:cellstackpercentrankfromrasterlayer INPUT="a.gpkg;b.gpkg"
 ↪  INPUT_VALUE_RASTER=dem.tif REFERENCE_LAYER=dem.tif VALUE_RASTER_BAND=1 METHOD=0
 ↪  IGNORE_NODATA=true CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Cell stack percentrank from raster layer**: INPUT="a.gpkg;b.gpkg" — Input layers; INPUT_VALUE_RASTER=dem.tif — Value raster layer; REFERENCE_LAYER=dem.tif — Reference layer; VALUE_RASTER_BAND=1 — Value raster band; METHOD=0 selects *Inclusive linear interpolation (PERCENTRANK.INC)*; IGNORE_NODATA=true — Ignore NoData values; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:cellstackpercentrankfromvalue — Cell stack percent rank from value

group: Raster analysis

This algorithm generates a raster containing the cell-wise percent rank value of a stack of input rasters based on a single input value.

At each cell location, the specified value is ranked among the respective values in the stack of all overlaid and sorted cell values from the input rasters. For values outside of the stack value distribution, the algorithm returns NoData because the value cannot be ranked among the cell values.

There are two methods for percentile calculation: Inclusive linearly interpolated percent rank (PERCENTRANK.INC) Exclusive linearly interpolated percent rank (PERCENTRANK.EXC) The linear interpolation method return the unique percent rank for different values. Both interpolation methods follow their counterpart methods implemented by LibreOffice or Microsoft Excel.

The output raster's extent and resolution is defined by a reference raster. If the input raster layers that do not match the cell size of the reference raster layer will be resampled using nearest neighbor resampling. NoData values in any of the input layers will result in a NoData cell output if the Ignore NoData parameter is not set. The output raster data type will always be Float32.

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input layers
METHOD	enum	no	0	Method — options: 0=Inclusive linear interpolation (PERCENTRANK.INC), 1=Exclusive linear interpolation (PERCENTRANK.EXC)

Parameter	Type	Required	Default	Description
VALUE	number	no	10	Value
IGNORE_NODATA	boolean	no	True	Ignore NoData values
REFERENCE_LAYER	raster	yes	—	Reference layer
OUTPUT_NODATA_VALUE *(adv)*	number	no	-9999	Output NoData value
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber)

Example usage

```
run native:cellstackpercentrankfromvalue INPUT="a.gpkg;b.gpkg" REFERENCE_LAYER=dem.tif
 ↪  METHOD=0 VALUE=10 IGNORE_NODATA=true CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Cell stack percent rank from value**: INPUT="a.gpkg;b.gpkg" — Input layers; REFERENCE_LAYER=dem.tif — Reference layer; METHOD=0 selects *Inclusive linear interpolation (PERCENTRANK.INC)*; VALUE=10 — Value; IGNORE_NODATA=true — Ignore NoData values; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:cellstatistics — Cell statistics

group: Raster analysis

The Cell statistics algorithm computes a value for each cell of the output raster. At each cell location, the output value is defined as a function of all overlaid cell values of the input rasters.

The output raster's extent and resolution is defined by a reference raster. The following functions can be applied on the input raster cells per output raster cell location: Sum Count Mean Median Standard deviation Variance Minimum Maximum Minority (least frequent value) Majority (most frequent value) Range (max-min) Variety (count of unique values) Input raster layers that do not match the cell size of the reference raster layer will be resampled using nearest neighbor resampling. The output raster data type will be set to the most complex data type present in the input datasets except when using the functions Mean, Standard deviation and Variance (data type is always Float32/Float64 depending on input float type) or Count and Variety (data type is always Int32). Calculation details - general: NoData values in any of the input layers will result in a NoData cell output if the Ignore NoData parameter is not set. Calculation details - Count: Count will always result in the number of cells without NoData values at the current cell location. Calculation details - Median: If the number of input layers is even, the median will be calculated as the arithmetic mean of the two middle values of the ordered cell input values. In this case the output data type is Float32. Calculation details - Minority/Majority: If no unique minority or majority could be found, the result is NoData, except all input cell values are equal.

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input layers
STATISTIC	enum	no	0	Statistic — options: 0=Sum, 1=Count, 2=Mean, 3=Median, 4=Standard deviation, 5=Variance, 6=Minimum, 7=Maximum, 8=Minority, 9=Majority, 10=Range, 11=Variety
IGNORE_NODATA	boolean	no	True	Ignore NoData values
REFERENCE_LAYER	raster	yes	—	Reference layer
OUTPUT_NODATA_VALUE *(adv)*	number	no	-9999	Output NoData value
CREATE_OPTIONS	string	no	—	Creation options

Parameter	Type	Required	Default	Description
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber)

Example usage

```
run native:cellstatistics INPUT="a.gpkg;b.gpkg" REFERENCE_LAYER=dem.tif STATISTIC=0
 ↪  IGNORE_NODATA=true CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Cell statistics**: INPUT="a.gpkg;b.gpkg" — Input layers; REFERENCE_LAYER=dem.tif — Reference layer; STATISTIC=0 selects *Sum*; IGNORE_NODATA=true — Ignore NoData values; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:centroids — Centroids ★

group: Vector geometry · **niva verb:** centroid

This algorithm creates a new point layer with points representing the centroid of the geometries in an input layer.

The attributes associated to each point in the output layer are the same ones associated to the original features.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ALL_PARTS	boolean	no	False	Create centroid for each part
OUTPUT	sink	yes	—	Centroids

Outputs: OUTPUT (outputVector)

Example usage

```
run native:centroids INPUT=input.gpkg ALL_PARTS=false OUTPUT=output.gpkg
```

This calls **Centroids**: INPUT=input.gpkg — Input layer; ALL_PARTS=false — Create centroid for each part; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:checkgeometryangle — Small angles

group: Check geometry

This algorithm checks the angles of line or polygon geometries. Angles below the minimum angle are errors.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
UNIQUE_ID	field	yes	—	Unique feature identifier
MIN_ANGLE	number	no	0	Minimum angle (in degrees)
ERRORS	sink	yes	—	Small angle errors
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: ERRORS (outputVector)

Example usage

```
run native:checkgeometryangle INPUT=input.gpkg UNIQUE_ID=field1 MIN_ANGLE=0
↪ ERRORS=errors.gpkg
```

This calls **Small angles**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `MIN_ANGLE=0` — Minimum angle (in degrees); `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometryarea` — Small polygons

group: Check geometry

This algorithm checks the areas of polygon geometries. Areas below the area threshold are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`AREATHRESHOLD`	area	no	0	Area threshold
`ERRORS`	sink	yes	—	Small polygons errors
`OUTPUT`	sink	no	—	Small polygons features
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometryarea INPUT=input.gpkg UNIQUE_ID=field1 AREATHRESHOLD=0
↪ ERRORS=errors.gpkg
```

This calls **Small polygons**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `AREATHRESHOLD=0` — Area threshold; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometrycontained` — Features inside polygon

group: Check geometry

This algorithm checks the input geometries contained in the polygons from the polygon layers list. A polygon layer can be checked against itself. Input features contained in the polygon layers features are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`POLYGONS`	multilayer	yes	—	Polygon layers
`ERRORS`	sink	yes	—	Errors from contained features
`OUTPUT`	sink	no	—	Contained features
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometrycontained INPUT=input.gpkg UNIQUE_ID=field1
↪ POLYGONS="a.gpkg;b.gpkg" ERRORS=errors.gpkg
```

This calls **Features inside polygon**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `POLYGONS="a.gpkg;b.gpkg"` — Polygon layers; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometrydangle` — Dangle-end lines

group: Check geometry

This algorithm checks lines with a dangle end. Lines with a dangle end are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`ERRORS`	sink	yes	—	Dangle-end errors
`OUTPUT`	sink	no	—	Dangle-end features
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometrydangle INPUT=input.gpkg UNIQUE_ID=field1 ERRORS=errors.gpkg
```

This calls **Dangle-end lines**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometrydegeneratepolygon` — Degenerate polygons

group: Check geometry

This algorithm checks the polygons with less than 3 points, which are degenerate polygons. Degenerate polygons are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`ERRORS`	sink	yes	—	Degenerate polygons errors
`OUTPUT`	sink	no	—	Degenerate polygons features
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometrydegeneratepolygon INPUT=input.gpkg UNIQUE_ID=field1
↪ ERRORS=errors.gpkg
```

This calls **Degenerate polygons**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometryduplicate` — Duplicated geometries

group: Check geometry

This algorithm checks the duplicate geometries. Duplicate geometries are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`ERRORS`	sink	yes	—	Duplicate geometries errors
`OUTPUT`	sink	no	—	Duplicate geometries

Parameter	Type	Required	Default	Description
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometryduplicate INPUT=input.gpkg UNIQUE_ID=field1 ERRORS=errors.gpkg
```

This calls **Duplicated geometries**: INPUT=input.gpkg — Input layer; UNIQUE_ID=field1 — Unique feature identifier; ERRORS=errors.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:checkgeometryduplicatenodes — Duplicated vertices

group: Check geometry

This algorithm checks the vertices of line and polygon geometries. Duplicated vertices are errors.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
UNIQUE_ID	field	yes	—	Unique feature identifier
ERRORS	sink	yes	—	Duplicated vertices errors
OUTPUT	sink	no	—	Duplicated vertices features
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometryduplicatenodes INPUT=input.gpkg UNIQUE_ID=field1
↪ ERRORS=errors.gpkg
```

This calls **Duplicated vertices**: INPUT=input.gpkg — Input layer; UNIQUE_ID=field1 — Unique feature identifier; ERRORS=errors.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:checkgeometryfollowboundaries — Polygons exceeding boundaries

group: Check geometry

This algorithm checks if the polygons follow the boundaries of the reference layer. Polygons not following reference boundaries are errors.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
UNIQUE_ID	field	yes	—	Unique feature identifier
ERRORS	sink	yes	—	Errors exceeding boundaries
OUTPUT	sink	no	—	Features exceeding boundaries
REF_LAYER	source	yes	—	Reference layer
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometryfollowboundaries INPUT=input.gpkg UNIQUE_ID=field1
↪ REF_LAYER=input.gpkg ERRORS=errors.gpkg
```

This calls **Polygons exceeding boundaries**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `REF_LAYER=input.gpkg` — Reference layer; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometrygap` — Small gaps

group: Check geometry

This algorithm checks the gaps between polygons. Gaps with an area smaller than the gap threshold are errors.

If an allowed gaps layer is given, the gaps contained in polygons from this layer will be ignored. An optional buffer can be applied to the allowed gaps.

The neighbors output layer is needed for the fix geometry (gaps) algorithm. It is a 1-N relational table for correspondence between a gap and the unique id of its neighbor features.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`GAP_THRESHOLD`	number	no	0	Gap threshold
`ALLOWED_GAPS_LAYER`	vector	no	—	Allowed gaps layer
`ALLOWED_GAPS_BUFFER`	distance	no	—	Allowed gaps buffer
`NEIGHBORS`	sink	yes	—	Neighbors layer
`ERRORS`	sink	no	—	Gap errors
`OUTPUT`	sink	yes	—	Gap features
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: `NEIGHBORS` (outputVector), `ERRORS` (outputVector), `OUTPUT` (outputVector)

Example usage

```
run native:checkgeometrygap INPUT=input.gpkg UNIQUE_ID=field1 GAP_THRESHOLD=0
 ↪  ALLOWED_GAPS_LAYER=input.gpkg ALLOWED_GAPS_BUFFER=10 NEIGHBORS=neighbors.gpkg
```

This calls **Small gaps**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `GAP_THRESHOLD=0` — Gap threshold; `ALLOWED_GAPS_LAYER=input.gpkg` — Allowed gaps layer; `ALLOWED_GAPS_BUFFER=10` — Allowed gaps buffer; `NEIGHBORS=neighbors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometryhole` — Holes

group: Check geometry

This algorithm checks the holes of polygon geometries. Holes are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`ERRORS`	sink	yes	—	Holes errors
`OUTPUT`	sink	no	—	Polygons with holes
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: `ERRORS` (outputVector), `OUTPUT` (outputVector)

Example usage

```
run native:checkgeometryhole INPUT=input.gpkg UNIQUE_ID=field1 ERRORS=errors.gpkg
```

This calls **Holes**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometrylineintersection` — Lines intersecting each other

group: Check geometry

This algorithm checks intersections between line geometries. Intersections between two different lines are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`ERRORS`	sink	yes	—	Intersection errors
`OUTPUT`	sink	no	—	Intersecting feature
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometrylineintersection INPUT=input.gpkg UNIQUE_ID=field1
 ↪  ERRORS=errors.gpkg
```

This calls **Lines intersecting each other**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometrylinelayerintersection` — Lines intersecting other layer

group: Check geometry

This algorithm checks if the input line layer features intersect with the check layer features. An input feature that intersects with a check layer feature is an error.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`CHECK_LAYER`	source	yes	—	Check layer
`ERRORS`	sink	yes	—	Line intersecting other layer errors
`OUTPUT`	sink	no	—	Line intersecting other layer features
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometrylinelayerintersection INPUT=input.gpkg UNIQUE_ID=field1
 ↪  CHECK_LAYER=input.gpkg ERRORS=errors.gpkg
```

This calls **Lines intersecting other layer**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `CHECK_LAYER=input.gpkg` — Check layer; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:checkgeometrymissingvertex — Missing vertices along borders

group: Check geometry

This algorithm checks for missing vertices along polygon borders. To be topologically correct, a vertex at the junction of two polygons must be present on both polygons. Missing vertices are errors.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
UNIQUE_ID	field	yes	—	Unique feature identifier
ERRORS	sink	yes	—	Missing vertices errors
OUTPUT	sink	no	—	Missing vertices features
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometrymissingvertex INPUT=input.gpkg UNIQUE_ID=field1 ERRORS=errors.gpkg
```

This calls **Missing vertices along borders**: INPUT=input.gpkg — Input layer; UNIQUE_ID=field1 — Unique feature identifier; ERRORS=errors.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:checkgeometrymultipart — Strictly multipart

group: Check geometry

This algorithm checks if multipart geometries have more than one part. Multipart geometries with only one part are errors.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
UNIQUE_ID	field	yes	—	Unique feature identifier
ERRORS	sink	yes	—	One-part geometry errors
OUTPUT	sink	no	—	One-part geometry features
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometrymultipart INPUT=input.gpkg UNIQUE_ID=field1 ERRORS=errors.gpkg
```

This calls **Strictly multipart**: INPUT=input.gpkg — Input layer; UNIQUE_ID=field1 — Unique feature identifier; ERRORS=errors.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:checkgeometryoverlap — Overlaps

group: Check geometry

This algorithm checks the overlapping areas. Overlapping areas smaller than the minimum overlapping area are errors.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
UNIQUE_ID	field	yes	—	Unique feature identifier

Parameter	Type	Required	Default	Description
ERRORS	sink	yes	—	Overlap errors
OUTPUT	sink	no	—	Overlap features
MIN_OVERLAP_AREA	number	no	0	Minimum overlap area
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometryoverlap INPUT=input.gpkg UNIQUE_ID=field1 MIN_OVERLAP_AREA=0
↪  ERRORS=errors.gpkg
```

This calls **Overlaps**: INPUT=input.gpkg — Input layer; UNIQUE_ID=field1 — Unique feature identifier; MIN_OVERLAP_AREA=0 — Minimum overlap area; ERRORS=errors.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:checkgeometrypointcoveredbyline — Points outside lines

group: Check geometry

This algorithm checks if the points in the input layer are covered by a line in the selected line layers. A point not covered by a line is an error.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
UNIQUE_ID	field	yes	—	Unique feature identifier
LINES	multilayer	yes	—	Line layers
ERRORS	sink	yes	—	Points not covered by a line
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: ERRORS (outputVector)

Example usage

```
run native:checkgeometrypointcoveredbyline INPUT=input.gpkg UNIQUE_ID=field1
↪  LINES="a.gpkg;b.gpkg" ERRORS=errors.gpkg
```

This calls **Points outside lines**: INPUT=input.gpkg — Input layer; UNIQUE_ID=field1 — Unique feature identifier; LINES="a.gpkg;b.gpkg" — Line layers; ERRORS=errors.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:checkgeometrypointinpolygon — Points outside polygons

group: Check geometry

This algorithm checks if points from the input layer are in polygons from the selected polygon layers. Points that are not fully inside polygons are errors.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
UNIQUE_ID	field	yes	—	Unique feature identifier
POLYGONS	multilayer	yes	—	Polygon layers
ERRORS	sink	yes	—	Points outside polygons errors
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: ERRORS (outputVector)

Example usage

```
run native:checkgeometrypointinpolygon INPUT=input.gpkg UNIQUE_ID=field1
 ↪  POLYGONS="a.gpkg;b.gpkg" ERRORS=errors.gpkg
```

This calls **Points outside polygons**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `POLYGONS="a.gpkg;b.gpkg"` — Polygon layers; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometrysegmentlength` — Small segments

group: Check geometry

This algorithm checks the minimum segment length of line or polygon geometries. Segments shorter than the minimum length are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`ERRORS`	sink	yes	—	Short segments errors
`OUTPUT`	sink	no	—	Short segments features
`MIN_SEGMENT_LENGTH`	number	no	0	Minimum segment length
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometrysegmentlength INPUT=input.gpkg UNIQUE_ID=field1
 ↪  MIN_SEGMENT_LENGTH=0 ERRORS=errors.gpkg
```

This calls **Small segments**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `MIN_SEGMENT_LENGTH=0` — Minimum segment length; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometryselfcontact` — Self-contacts

group: Check geometry

This algorithm checks if the geometry has self contact points (in line or polygon). Self contacts are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`ERRORS`	sink	yes	—	Self contact error points
`OUTPUT`	sink	no	—	Self contact features
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometryselfcontact INPUT=input.gpkg UNIQUE_ID=field1 ERRORS=errors.gpkg
```

This calls **Self-contacts**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometryselfintersection` — Self-intersections

group: Check geometry

This algorithm checks self-intersecting geometries. Self-intersecting geometries are errors.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`ERRORS`	sink	yes	—	Self-intersecting errors
`OUTPUT`	sink	no	—	Self-intersecting features
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometryselfintersection INPUT=input.gpkg UNIQUE_ID=field1
↪  ERRORS=errors.gpkg
```

This calls **Self-intersections**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkgeometrysliverpolygon` — Sliver polygons

group: Check geometry

This algorithm checks sliver polygons.

The thinness is the ratio between the area of the minimum square containing the polygon and the area of the polygon itself (a square has a thinness value of 1). The thinness value is between 1 and +infinity. If a polygon has an area higher than the maximum area, it is skipped (a maximum area value of 0 means no area check).

Polygons having a thinness higher than the maximum thinness are errors.

To fix sliver polygons, use the ”Fix small polygons” algorithm.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`UNIQUE_ID`	field	yes	—	Unique feature identifier
`ERRORS`	sink	yes	—	Sliver polygon errors
`OUTPUT`	sink	no	—	Sliver polygon features
`MAX_THINNESS`	number	no	20	Maximum thinness
`MAX_AREA`	number	no	0	Maximum area (map units squared)
`TOLERANCE *(adv)*`	number	no	8	Tolerance

Outputs: ERRORS (outputVector), OUTPUT (outputVector)

Example usage

```
run native:checkgeometrysliverpolygon INPUT=input.gpkg UNIQUE_ID=field1 MAX_THINNESS=20
↪  MAX_AREA=0 ERRORS=errors.gpkg
```

This calls **Sliver polygons**: `INPUT=input.gpkg` — Input layer; `UNIQUE_ID=field1` — Unique feature identifier; `MAX_THINNESS=20` — Maximum thinness; `MAX_AREA=0` — Maximum area (map units squared); `ERRORS=errors.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:checkvalidity` — Check validity

group: Vector geometry

This algorithm performs a validity check on the geometries of a vector layer.

The geometries are classified in three groups (valid, invalid and error), and a vector layer is generated with the features in each of these categories.

By default the algorithm uses the strict OGC definition of polygon validity, where a polygon is marked as invalid if a self-intersecting ring causes an interior hole. If the 'Ignore ring self intersections' option is checked, then this rule will be ignored and a more lenient validity check will be performed.

The GEOS method is faster and performs better on larger geometries, but is limited to only returning the first error encountered in a geometry. The QGIS method will be slower but reports all errors encountered in the geometry, not just the first.

Parameter	Type	Required	Default	Description
`INPUT_LAYER`	source	yes	—	Input layer
`METHOD`	enum	no	2	Method — options: 0=The one selected in digitizing settings, 1=QGIS, 2=GEOS
`IGNORE_RING_SELF_INTERSECTION`	boolean	no	False	Ignore ring self intersections
`VALID_OUTPUT`	sink	no	—	Valid output
`INVALID_OUTPUT`	sink	no	—	Invalid output
`ERROR_OUTPUT`	sink	no	—	Error output

Outputs: `VALID_OUTPUT` (outputVector), `INVALID_OUTPUT` (outputVector), `ERROR_OUTPUT` (outputVector), `VALID_COUNT` (outputNumber), `INVALID_COUNT` (outputNumber), `ERROR_COUNT` (outputNumber)

Example usage

```
run native:checkvalidity INPUT_LAYER=input.gpkg METHOD=2
↪  IGNORE_RING_SELF_INTERSECTION=false VALID_OUTPUT=valid_output.gpkg
```

This calls **Check validity**: `INPUT_LAYER=input.gpkg` — Input layer; `METHOD=2` selects *GEOS*; `IGNORE_RING_SELF_INTERSECTION=false` — Ignore ring self intersections; `VALID_OUTPUT=valid_output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:climbalongline` — Climb along line

group: Vector analysis

This algorithm calculates the total climb and descent along line geometries.

Input layer must have Z values present. If Z values are not available, the ”Drape” (set Z value from raster) algorithm may be used to add Z values from a DEM layer.

The output layer is a copy of the input layer with additional fields that contain the total climb, total descent, the minimum elevation and the maximum elevation for each line geometry.If the input layer contains fields with the same names as these added fields, they will be renamed (field names will be altered to ”name_2”, ”name_3”, etc, finding the first non-duplicate name).

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Line layer
OUTPUT	sink	yes	—	Climb layer

Outputs: OUTPUT (outputVector), TOTALCLIMB (outputNumber), TOTALDESCENT (outputNumber), MINELEVATION (outputNumber), MAXELEVATION (outputNumber)

Example usage

```
run native:climbalongline INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Climb along line**: INPUT=input.gpkg — Line layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:clip — Clip ★

group: Vector overlay · **niva verb:** clip

This algorithm clips a vector layer using the features of an additional polygon layer. Only the parts of the features in the Input layer that fall within the polygons of the Overlay layer will be added to the resulting layer.

The attributes of the features are not modified, although properties such as area or length of the features will be modified by the clipping operation. If such properties are stored as attributes, those attributes will have to be manually updated.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OVERLAY	source	yes	—	Overlay layer
OUTPUT	sink	yes	—	Clipped

Outputs: OUTPUT (outputVector)

Example usage

```
run native:clip INPUT=input.gpkg OVERLAY=input.gpkg OUTPUT=output.gpkg
```

This calls **Clip**: INPUT=input.gpkg — Input layer; OVERLAY=input.gpkg — Overlay layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:collect — Collect geometries ★

group: Vector geometry · **niva verb:** collect

This algorithm takes a vector layer and collects its geometries into new multipart geometries. One or more attributes can be specified to collect only geometries belonging to the same class (having the same value for the specified attributes), alternatively all geometries can be collected.

All output geometries will be converted to multi geometries, even those with just a single part. This algorithm does not dissolve overlapping geometries - they will be collected together without modifying the shape of each geometry part.

See the 'Promote to multipart' or 'Aggregate' algorithms for alternative options.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	no	—	Unique ID fields
OUTPUT	sink	yes	—	Collected

Outputs: OUTPUT (outputVector)

Example usage

```
run native:collect INPUT=input.gpkg FIELD=field1 OUTPUT=output.gpkg
```

This calls **Collect geometries**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — Unique ID fields; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:combinestyles` — Combine style databases

group: Cartography

This algorithm combines multiple QGIS style databases into a single style database. If any symbols exist with duplicate names between the different source databases these will be renamed to have unique names in the output combined database.

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input databases
OUTPUT	fileDestination	yes	—	Output style database
OBJECTS	enum	no	[0, 1, 2, 3]	Objects to combine — options: 0=Symbols, 1=Color ramps, 2=Text formats, 3=Label settings

Outputs: OUTPUT (outputFile), SYMBOLS (outputNumber), COLORRAMPS (outputNumber), TEXTFORMATS (outputNumber), LABELSETTINGS (outputNumber)

Example usage

```
run native:combinestyles INPUT="a.gpkg;b.gpkg" OBJECTS=0 OUTPUT=output.txt
```

This calls **Combine style databases**: `INPUT="a.gpkg;b.gpkg"` — Input databases; `OBJECTS=0` selects *Symbols*; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:concavehull` — Concave hull (by layer)

group: Vector geometry

This algorithm computes the concave hull covering all features from an input point layer.

See the 'Concave hull (by feature)' algorithm for a concave hull calculation which covers individual features from a layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ALPHA	number	no	0.3	Threshold (0-1, where 1 is equivalent with Convex Hull)
HOLES	boolean	no	True	Allow holes
NO_MULTIGEOMETRY	boolean	no	False	Split multipart geometry into singleparts
OUTPUT	sink	yes	—	Concave hull

Outputs: OUTPUT (outputVector)

Example usage

```
run native:concavehull INPUT=input.gpkg ALPHA=0.3 HOLES=true NO_MULTIGEOMETRY=false
↪ OUTPUT=output.gpkg
```

This calls **Concave hull (by layer)**: `INPUT=input.gpkg` — Input layer; `ALPHA=0.3` — Threshold (0-1, where 1 is equivalent with Convex Hull); `HOLES=true` — Allow holes; `NO_MULTIGEOMETRY=false` — Split multipart geometry into singleparts; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:concavehullbyfeature` — Concave hull (by feature)

group: Vector geometry

This algorithm calculates the concave hull for each feature in an input layer.

A concave hull is a polygon which contains all the points of the input geometries, but is a better approximation than the convex hull to the area occupied by the input.

It is frequently used to convert a multi-point into a polygonal area which contains all the points from the input geometry.

See the 'Concave hull (by layer)' algorithm for a concave hull calculation which covers the whole layer or grouped subsets of features.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`ALPHA`	number	no	0.3	Threshold (0-1, where 1 is equivalent with Convex Hull)
`HOLES`	boolean	no	True	Allow holes
`OUTPUT`	sink	yes	—	Concave hulls

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:concavehullbyfeature INPUT=input.gpkg ALPHA=0.3 HOLES=true OUTPUT=output.gpkg
```

This calls **Concave hull (by feature)**: `INPUT=input.gpkg` — Input layer; `ALPHA=0.3` — Threshold (0-1, where 1 is equivalent with Convex Hull); `HOLES=true` — Allow holes; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:condition` — Conditional branch

group: Modeler tools

This algorithm adds a conditional branch into a model, allowing parts of the model to be executed based on the result of an expression evaluation.

Example usage

```
run native:condition
```

This calls **Conditional branch** (no required parameters).

`native:convertgeometrytype` — Convert geometry type

group: Vector geometry

This algorithm generates a new layer based on an existing one, with a different type of geometry.

Not all conversions are possible. For instance, a line layer can be converted to a point layer, but a point layer cannot be converted to a line layer.

See the ”Polygonize” or ”Lines to polygons” algorithms for alternative options.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
TYPE	enum	yes	—	New geometry type — options: 0=Centroids, 1=Nodes, 2=Linestrings, 3=Multilinestrings, 4=Polygons
OUTPUT	sink	yes	—	Converted

Outputs: OUTPUT (outputVector)

Example usage

```
run native:convertgeometrytype INPUT=input.gpkg TYPE=0 OUTPUT=output.gpkg
```

This calls **Convert geometry type**: INPUT=input.gpkg — Input layer; TYPE=0 selects *Centroids*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:convertgpsdata — Convert GPS data

group: GPS

This algorithm uses the GPSBabel tool to convert a GPS data file from a range of formats to the GPX standard format.

Parameter	Type	Required	Default	Description
INPUT	file	yes	—	Input file
FORMAT	string	yes	—	Format
FEATURE_TYPE	enum	no	0	Feature type — options: 0=Waypoints, 1=Routes, 2=Tracks
OUTPUT	fileDestination	yes	—	Output

Outputs: OUTPUT (outputFile), OUTPUT_LAYER (outputVector)

Example usage

```
run native:convertgpsdata INPUT=input.dat FORMAT=value FEATURE_TYPE=0 OUTPUT=output.txt
```

This calls **Convert GPS data**: INPUT=input.dat — Input file; FORMAT=value — Format; FEATURE_TYPE=0 selects *Waypoints*; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:convertgpxfeaturetype — Convert GPX feature type

group: GPS

This algorithm uses the GPSBabel tool to convert GPX features from one type to another (e.g. converting all waypoint features to a route feature).

Parameter	Type	Required	Default	Description
INPUT	file	yes	—	Input file
CONVERSION	enum	no	0	Conversion — options: 0=Waypoints from a Route, 1=Waypoints from a Track, 2=Route from Waypoints, 3=Track from Waypoints
OUTPUT	fileDestination	yes	—	Output

Outputs: OUTPUT (outputFile), OUTPUT_LAYER (outputVector)

Example usage

```
run native:convertgpxfeaturetype INPUT=input.dat CONVERSION=0 OUTPUT=output.txt
```

This calls **Convert GPX feature type**: INPUT=input.dat — Input file; CONVERSION=0 selects *Waypoints from a Route*; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:converttocurves` — Convert to curved geometries

group: Vector geometry

This algorithm converts a geometry into its curved geometry equivalent.

Already curved geometries will be retained without change.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`DISTANCE`	number	no	1e-06	Maximum distance tolerance
`ANGLE`	number	no	1e-06	Maximum angle tolerance
`OUTPUT`	sink	yes	—	Curves

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:converttocurves INPUT=input.gpkg DISTANCE=1e-06 ANGLE=1e-06 OUTPUT=output.gpkg
```

This calls **Convert to curved geometries**: `INPUT=input.gpkg` — Input layer; `DISTANCE=1e-06` — Maximum distance tolerance; `ANGLE=1e-06` — Maximum angle tolerance; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:convexhull` — Convex hull ★

group: Vector geometry · **niva verb:** `convexhull`

This algorithm calculates the convex hull for each feature in an input layer.

See the 'Minimum bounding geometry' algorithm for a convex hull calculation which covers the whole layer or grouped subsets of features.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`OUTPUT`	sink	yes	—	Convex hulls

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:convexhull INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Convex hull**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:copylayermetadata` — Copy layer metadata

group: Metadata tools

This algorithm copies metadata from a source layer to a target layer.

Any existing metadata in the target layer will be replaced.

Parameter	Type	Required	Default	Description
`SOURCE`	layer	yes	—	Source layer
`TARGET`	layer	yes	—	Target layer
`DEFAULT`	boolean	no	False	Save metadata as default

Outputs: OUTPUT (outputLayer)

Example usage

```
run native:copylayermetadata SOURCE=input.gpkg TARGET=input.gpkg DEFAULT=false
```

This calls **Copy layer metadata**: SOURCE=input.gpkg — Source layer; TARGET=input.gpkg — Target layer; DEFAULT=false — Save metadata as default. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:countpointsinpolygon — Count points in polygon ★

group: Vector analysis · **niva verb:** countpoints

This algorithm takes a points layer and a polygon layer and counts the number of points from the first one in each polygons of the second one.

A new polygons layer is generated, with the exact same content as the input polygons layer, but containing an additional field with the points count corresponding to each polygon.

An optional weight field can be used to assign weights to each point. If set, the count generated will be the sum of the weight field for each point contained by the polygon.

Alternatively, a unique class field can be specified. If set, points are classified based on the selected attribute, and if several points with the same attribute value are within the polygon, only one of them is counted. The final count of the point in a polygon is, therefore, the count of different classes that are found in it.

Both the weight field and unique class field cannot be specified. If they are, the weight field will take precedence and the unique class field will be ignored.

Parameter	Type	Required	Default	Description
POLYGONS	source	yes	—	Polygons
POINTS	source	yes	—	Points
WEIGHT	field	no	—	Weight field
CLASSFIELD	field	no	—	Class field
FIELD	string	no	NUMPOINTS	Count field name
OUTPUT	sink	yes	—	Count

Outputs: OUTPUT (outputVector)

Example usage

```
run native:countpointsinpolygon POLYGONS=input.gpkg POINTS=input.gpkg WEIGHT=field1
↪ CLASSFIELD=field1 FIELD=NUMPOINTS OUTPUT=output.gpkg
```

This calls **Count points in polygon**: POLYGONS=input.gpkg — Polygons; POINTS=input.gpkg — Points; WEIGHT=field1 — Weight field; CLASSFIELD=field1 — Class field; FIELD=NUMPOINTS — Count field name; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:coveragesimplify — Simplify coverage

group: Vector coverage

This algorithm operates on a coverage (represented as a set of polygon features with exactly matching edge geometry) to apply a Visvalingam–Whyatt simplification to the edges, reducing complexity in proportion with the provided tolerance, while retaining a valid coverage (i.e. no edges will cross or touch after the simplification).

Geometries will never be removed, but they may be simplified down to just a triangle. Also, some geometries (such as polygons which have too few non-repeated points) will be returned unchanged.

If the input dataset is not a valid coverage due to overlaps, it will still be simplified, but invalid topology such as crossing edges will still be invalid.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
TOLERANCE	distance	no	1.0	Tolerance
PRESERVE_BOUNDARY	boolean	no	False	Preserve boundary
OUTPUT	sink	yes	—	Simplified

Outputs: OUTPUT (outputVector)

Example usage

```
run native:coveragesimplify INPUT=input.gpkg TOLERANCE=1 PRESERVE_BOUNDARY=false
↪  OUTPUT=output.gpkg
```

This calls **Simplify coverage**: INPUT=input.gpkg — Input layer; TOLERANCE=1 — Tolerance; PRESERVE_BOUNDARY=false — Preserve boundary; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:coverageunion — Dissolve coverage

group: Vector coverage

This algorithm operates on a coverage (represented as a set of polygon features with exactly matching edge geometry) to dissolve (union) the geometries.

It provides a heavily optimized approach for unioning these features compared with the standard Dissolve tools.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Dissolved

Outputs: OUTPUT (outputVector)

Example usage

```
run native:coverageunion INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Dissolve coverage**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:coveragevalidate — Validate coverage

group: Vector coverage

This algorithm analyzes a coverage (represented as a set of polygon features with exactly matching edge geometry) to find places where the assumption of exactly matching edges is not met.

Invalidity includes polygons that overlap or that have gaps smaller than the specified gap width.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
GAP_WIDTH	distance	no	0.0	Gap width
INVALID_EDGES	sink	no	—	Invalid edges

Outputs: INVALID_EDGES (outputVector), IS_VALID (outputBoolean)

Example usage

```
run native:coveragevalidate INPUT=input.gpkg GAP_WIDTH=0 INVALID_EDGES=invalid_edges.gpkg
```

This calls **Validate coverage**: INPUT=input.gpkg — Input layer; GAP_WIDTH=0 — Gap width; INVALID_EDGES=invalid_edges.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createattributeindex — Create attribute index

group: Vector general

This algorithm creates an index to speed up queries made against a field in a table. Support for index creation is dependent on the layer's data provider and the field type.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
FIELD	field	yes	—	Attribute to index

Outputs: OUTPUT (outputVector)

Example usage

```
run native:createattributeindex INPUT=input.gpkg FIELD=field1
```

This calls **Create attribute index**: INPUT=input.gpkg — Input layer; FIELD=field1 — Attribute to index. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createconstantrasterlayer — Create constant raster layer

group: Raster creation

This algorithm generates a raster layer for a given extent and cell size filled with a single constant value. Additionally an output data type can be specified. The algorithm will abort if a value has been entered that cannot be represented by the selected output raster data type.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Desired extent
TARGET_CRS	crs	no	ProjectCrs	Target CRS
PIXEL_SIZE	number	no	1	Pixel size
NUMBER	number	no	1	Constant value
OUTPUT_TYPE *(adv)*	enum	no	5	Output raster data type — options: 0=Byte, 1=Integer16, 2=Unsigned Integer16, 3=Integer32, 4=Unsigned Integer32, 5=Float32, 6=Float64
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Constant

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:createconstantrasterlayer EXTENT=0,1000,0,1000 TARGET_CRS=EPSG:6346
↪  PIXEL_SIZE=1 NUMBER=1 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Create constant raster layer**: EXTENT=0,1000,0,1000 — Desired extent; TARGET_CRS=EPSG:6346 — Target CRS; PIXEL_SIZE=1 — Pixel size; NUMBER=1 — Constant value; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createdirectory — Create directory

group: Modeler tools

This algorithm creates a new directory on a file system. Directories will be created recursively, creating all required parent directories in order to construct the full specified directory path.

No errors will be raised if the directory already exists.

Parameter	Type	Required	Default	Description
PATH	layer	yes	—	Directory path

Outputs: OUTPUT (outputFolder)

Example usage

```
run native:createdirectory PATH=input.gpkg
```

This calls **Create directory**: PATH=input.gpkg — Directory path. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:creategrid — Create grid

group: Vector creation

This algorithm creates a vector layer with a grid covering a given extent. Elements in the grid can be points, lines or polygons. The size and/or placement of each element in the grid is defined using a horizontal and vertical spacing. The CRS of the output layer must be defined. The grid extent and the spacing values must be expressed in the coordinates and units of this CRS. The top-left point (minX, maxY) is used as the reference point. That means that, at that point, an element is guaranteed to be placed. Unless the width and height of the selected extent is a multiple of the selected spacing, that is not true for the other points that define that extent.

Parameter	Type	Required	Default	Description
TYPE	enum	no	0	Grid type — options: 0=Point, 1=Line, 2=Rectangle (Polygon), 3=Diamond (Polygon), 4=Hexagon (Polygon)
EXTENT	extent	yes	—	Grid extent
HSPACING	distance	no	1	Horizontal spacing
VSPACING	distance	no	1	Vertical spacing
HOVERLAY	distance	no	0	Horizontal overlay
VOVERLAY	distance	no	0	Vertical overlay
CRS	crs	no	ProjectCrs	Grid CRS
OUTPUT	sink	yes	—	Grid

Outputs: OUTPUT (outputVector)

Example usage

```
run native:creategrid EXTENT=0,1000,0,1000 TYPE=0 HSPACING=1 VSPACING=1 HOVERLAY=0
↪  VOVERLAY=0 CRS=EPSG:6346 OUTPUT=output.gpkg
```

This calls **Create grid**: EXTENT=0,1000,0,1000 — Grid extent; TYPE=0 selects *Point*; HSPACING=1 — Horizontal spacing; VSPACING=1 — Vertical spacing; HOVERLAY=0 — Horizontal overlay; VOVERLAY=0 — Vertical overlay; CRS=EPSG:6346 — Grid CRS; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createpointslayerfromtable — Create points layer from table

group: Vector creation

This algorithm generates a point layer based on the coordinates from an input table.

The table must contain a field with the X coordinate of each point and another one with the Y coordinate, as well as optional fields with Z and M values. A CRS for the output layer has to be specified, and the coordinates in the table are assumed to be expressed in the units used by that CRS. The attributes table of the resulting layer will be the input table.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
XFIELD	field	yes	—	X field
YFIELD	field	yes	—	Y field
ZFIELD	field	no	—	Z field
MFIELD	field	no	—	M field
TARGET_CRS	crs	no	EPSG:4326	Target CRS
OUTPUT	sink	yes	—	Points from table

Outputs: OUTPUT (outputVector)

Example usage

```
run native:createpointslayerfromtable INPUT=input.gpkg XFIELD=field1 YFIELD=field1
 ↪  ZFIELD=field1 MFIELD=field1 TARGET_CRS=EPSG:6346 OUTPUT=output.gpkg
```

This calls **Create points layer from table**: INPUT=input.gpkg — Input layer; XFIELD=field1 — X field; YFIELD=field1 — Y field; ZFIELD=field1 — Z field; MFIELD=field1 — M field; TARGET_CRS=EPSG:6346 — Target CRS; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createrandombinomialrasterlayer — Create random raster layer (binomial distribution)

group: Raster creation

This algorithm generates a raster layer for a given extent and cell size filled with binomially distributed random values. By default, the values will be chosen given an N of 10 and a probability of 0.5. This can be overridden by using the advanced parameter for N and probability. The raster data type is set to Integer types (Integer16 by default). The binomial distribution random values are defined as positive integer numbers. A floating point raster will represent a cast of integer values to floating point.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Desired extent
TARGET_CRS	crs	no	ProjectCrs	Target CRS
PIXEL_SIZE	number	no	1	Pixel size
OUTPUT_TYPE *(adv)*	enum	no	0	Output raster data type — options: 0=Integer16, 1=Unsigned Integer16, 2=Integer32, 3=Unsigned Integer32, 4=Float32, 5=Float64
N *(adv)*	number	no	10	N
PROBABILITY *(adv)*	number	no	0.5	Probability
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:createrandombinomialrasterlayer EXTENT=0,1000,0,1000 TARGET_CRS=EPSG:6346
↪  PIXEL_SIZE=1 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Create random raster layer (binomial distribution)**: `EXTENT=0,1000,0,1000` — Desired extent; `TARGET_CRS=EPSG:6346` — Target CRS; `PIXEL_SIZE=1` — Pixel size; `CREATE_OPTIONS=value` — Creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:createrandomexponentialrasterlayer` — Create random raster layer (exponential distribution)

group: Raster creation

This algorithm generates a raster layer for a given extent and cell size filled with exponentially distributed random values. By default, the values will be chosen given a lambda of 1.0. This can be overridden by using the advanced parameter for lambda. The raster data type is set to Float32 by default as the exponential distribution random values are floating point numbers.

Parameter	Type	Required	Default	Description
`EXTENT`	extent	yes	—	Desired extent
`TARGET_CRS`	crs	no	ProjectCrs	Target CRS
`PIXEL_SIZE`	number	no	1	Pixel size
`OUTPUT_TYPE *(adv)*`	enum	no	0	Output raster data type — options: 0=Float32, 1=Float64
`LAMBDA *(adv)*`	number	no	1.0	Lambda
`CREATE_OPTIONS`	string	no	—	Creation options
`CREATION_OPTIONS *(adv)*`	string	no	—	Creation options
`OUTPUT`	rasterDestination	yes	—	Output raster

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run native:createrandomexponentialrasterlayer EXTENT=0,1000,0,1000 TARGET_CRS=EPSG:6346
↪  PIXEL_SIZE=1 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Create random raster layer (exponential distribution)**: `EXTENT=0,1000,0,1000` — Desired extent; `TARGET_CRS=EPSG:6346` — Target CRS; `PIXEL_SIZE=1` — Pixel size; `CREATE_OPTIONS=value` — Creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:createrandomgammarasterlayer` — Create random raster layer (gamma distribution)

group: Raster creation

This algorithm generates a raster layer for a given extent and cell size filled with gamma distributed random values. By default, the values will be chosen given an alpha and beta value of 1.0. This can be overridden by using the advanced parameter for alpha and beta. The raster data type is set to Float32 by default as the gamma distribution random values are floating point numbers.

Parameter	Type	Required	Default	Description
`EXTENT`	extent	yes	—	Desired extent
`TARGET_CRS`	crs	no	ProjectCrs	Target CRS
`PIXEL_SIZE`	number	no	1	Pixel size

Parameter	Type	Required	Default	Description
OUTPUT_TYPE *(adv)*	enum	no	0	Output raster data type — options: 0=Float32, 1=Float64
ALPHA *(adv)*	number	no	1.0	Alpha
BETA *(adv)*	number	no	1.0	Beta
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:createrandomgammarasterlayer EXTENT=0,1000,0,1000 TARGET_CRS=EPSG:6346
↪  PIXEL_SIZE=1 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Create random raster layer (gamma distribution)**: EXTENT=0,1000,0,1000 — Desired extent; TARGET_CRS=EPSG:6346 — Target CRS; PIXEL_SIZE=1 — Pixel size; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createrandomgeometricrasterlayer — Create random raster layer (geometric distribution)

group: Raster creation

This algorithm generates a raster layer for a given extent and cell size filled with geometrically distributed random values. By default, the values will be chosen given a probability of 0.5. This can be overridden by using the advanced parameter for mean value. The raster data type is set to Integer types (Integer16 by default). The geometric distribution random values are defined as positive integer numbers. A floating point raster will represent a cast of integer values to floating point.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Desired extent
TARGET_CRS	crs	no	ProjectCrs	Target CRS
PIXEL_SIZE	number	no	1	Pixel size
OUTPUT_TYPE *(adv)*	enum	no	0	Output raster data type — options: 0=Integer16, 1=Unsigned Integer16, 2=Integer32, 3=Unsigned Integer32, 4=Float32, 5=Float64
PROBABILITY *(adv)*	number	no	0.5	Probability
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:createrandomgeometricrasterlayer EXTENT=0,1000,0,1000 TARGET_CRS=EPSG:6346
↪  PIXEL_SIZE=1 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Create random raster layer (geometric distribution)**: EXTENT=0,1000,0,1000 — Desired extent; TARGET_CRS=EPSG:6346 — Target CRS; PIXEL_SIZE=1 — Pixel size; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createrandomnegativebinomialrasterlayer — Create random raster layer (negative binomial distribution)

group: Raster creation

This algorithm generates a raster layer for a given extent and cell size filled with negative binomially distributed random values. By default, the values will be chosen given a distribution parameter k of 10.0 and a probability of 0.5. This can be overridden by using the advanced parameters for k and probability. The raster data type is set to Integer types (Integer 16 by default). The negative binomial distribution random values are defined as positive integer numbers. A floating point raster will represent a cast of integer values to floating point.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Desired extent
TARGET_CRS	crs	no	ProjectCrs	Target CRS
PIXEL_SIZE	number	no	1	Pixel size
OUTPUT_TYPE *(adv)*	enum	no	0	Output raster data type — options: 0=Integer16, 1=Unsigned Integer16, 2=Integer32, 3=Unsigned Integer32, 4=Float32, 5=Float64
K_PARAMETER *(adv)*	number	no	10	Distribution parameter k
PROBABILITY *(adv)*	number	no	0.5	Probability
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:createrandomnegativebinomialrasterlayer EXTENT=0,1000,0,1000
 ↪  TARGET_CRS=EPSG:6346 PIXEL_SIZE=1 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Create random raster layer (negative binomial distribution)**: EXTENT=0,1000,0,1000 — Desired extent; TARGET_CRS=EPSG:6346 — Target CRS; PIXEL_SIZE=1 — Pixel size; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createrandomnormalrasterlayer — Create random raster layer (normal distribution)

group: Raster creation

This algorithm generates a raster layer for a given extent and cell size filled with normally distributed random values. By default, the values will be chosen given a mean of 0.0 and a standard deviation of 1.0. This can be overridden by using the advanced parameters for mean and standard deviation value. The raster data type is set to Float32 by default as the normal distribution random values are floating point numbers.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Desired extent
TARGET_CRS	crs	no	ProjectCrs	Target CRS
PIXEL_SIZE	number	no	1	Pixel size
OUTPUT_TYPE *(adv)*	enum	no	0	Output raster data type — options: 0=Float32, 1=Float64
MEAN *(adv)*	number	no	0	Mean of normal distribution
STDDEV *(adv)*	number	no	1	Standard deviation of normal distribution
CREATE_OPTIONS	string	no	—	Creation options

Parameter	Type	Required	Default	Description
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:createrandomnormalrasterlayer EXTENT=0,1000,0,1000 TARGET_CRS=EPSG:6346
↪  PIXEL_SIZE=1 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Create random raster layer (normal distribution)**: EXTENT=0,1000,0,1000 — Desired extent; TARGET_CRS=EPSG:6346 — Target CRS; PIXEL_SIZE=1 — Pixel size; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createrandompoissonrasterlayer — Create random raster layer (poisson distribution)

group: Raster creation

This algorithm generates a raster layer for a given extent and cell size filled with poisson distributed random values. By default, the values will be chosen given a mean of 1.0. This can be overridden by using the advanced parameter for mean value. The raster data type is set to Integer types (Integer16 by default). The poisson distribution random values are positive integer numbers. A floating point raster will represent a cast of integer values to floating point.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Desired extent
TARGET_CRS	crs	no	ProjectCrs	Target CRS
PIXEL_SIZE	number	no	1	Pixel size
OUTPUT_TYPE *(adv)*	enum	no	0	Output raster data type — options: 0=Integer16, 1=Unsigned Integer16, 2=Integer32, 3=Unsigned Integer32, 4=Float32, 5=Float64
MEAN *(adv)*	number	no	1.0	Mean
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:createrandompoissonrasterlayer EXTENT=0,1000,0,1000 TARGET_CRS=EPSG:6346
↪  PIXEL_SIZE=1 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Create random raster layer (poisson distribution)**: EXTENT=0,1000,0,1000 — Desired extent; TARGET_CRS=EPSG:6346 — Target CRS; PIXEL_SIZE=1 — Pixel size; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createrandomuniformrasterlayer — Create random raster layer (uniform distribution)

group: Raster creation

This algorithm generates a raster layer for a given extent and cell size filled with random values. By default, the values will range between the minimum and maximum value of the specified output

raster type. This can be overridden by using the advanced parameters for lower and upper bound value. If the bounds have the same value or both are zero (default) the algorithm will create random values in the full value range of the chosen raster data type. Choosing bounds outside the acceptable range of the output raster type will abort the algorithm.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Desired extent
TARGET_CRS	crs	no	ProjectCrs	Target CRS
PIXEL_SIZE	number	no	1	Pixel size
OUTPUT_TYPE *(adv)*	enum	no	5	Output raster data type — options: 0=Byte, 1=Integer16, 2=Unsigned Integer16, 3=Integer32, 4=Unsigned Integer32, 5=Float32, 6=Float64
LOWER_BOUND *(adv)*	number	yes	—	Lower bound for random number range
UPPER_BOUND *(adv)*	number	yes	—	Upper bound for random number range
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:createrandomuniformrasterlayer EXTENT=0,1000,0,1000 LOWER_BOUND=10
↪ UPPER_BOUND=10 TARGET_CRS=EPSG:6346 PIXEL_SIZE=1 CREATE_OPTIONS=value
↪ OUTPUT=output.tif
```

This calls **Create random raster layer (uniform distribution)**: EXTENT=0,1000,0,1000 — Desired extent; LOWER_BOUND=10 — Lower bound for random number range; UPPER_BOUND=10 — Upper bound for random number range; TARGET_CRS=EPSG:6346 — Target CRS; PIXEL_SIZE=1 — Pixel size; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:createspatialindex — Create spatial index

group: Vector general

Creates an index to speed up access to the features in a layer based on their spatial location. Support for spatial index creation is dependent on the layer's data provider.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer

Outputs: OUTPUT (outputVector)

Example usage

```
run native:createspatialindex INPUT=input.gpkg
```

This calls **Create spatial index**: INPUT=input.gpkg — Input layer. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:dbscanclustering — DBSCAN clustering

group: Vector analysis

This algorithm clusters point features based on a 2D implementation of Density-based spatial clustering of applications with noise (DBSCAN) algorithm.

The algorithm requires two parameters, a minimum cluster size (“minPts”), and the maximum distance allowed between clustered points (“eps”).

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
MIN_SIZE	number	no	5	Minimum cluster size
EPS	distance	no	1	Maximum distance between clustered points
DBSCAN* *(adv)*	boolean	no	False	Treat border points as noise (DBSCAN*)
FIELD_NAME *(adv)*	string	no	CLUSTER_ID	Cluster field name
SIZE_FIELD_NAME *(adv)*	string	no	CLUSTER_SIZE	Cluster size field name
OUTPUT	sink	yes	—	Clusters

Outputs: OUTPUT (outputVector), NUM_CLUSTERS (outputNumber)

Example usage

```
run native:dbscanclustering INPUT=input.gpkg MIN_SIZE=5 EPS=1 OUTPUT=output.gpkg
```

This calls **DBSCAN clustering**: INPUT=input.gpkg — Input layer; MIN_SIZE=5 — Minimum cluster size; EPS=1 — Maximum distance between clustered points; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:definecurrentprojection — Define projection

group: Vector general

This algorithm sets an existing layer's projection to the provided CRS without reprojecting features. Contrary to the ”Assign projection” algorithm, it will not output a new layer.

If the input layer is a shapefile, the .prj file will be overwritten — or created if missing — to match the provided CRS.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input shapefile
CRS	crs	no	<QgsCoordinateReferenceSystem: EPSG:4326>	CRS

Outputs: OUTPUT (outputVector)

Example usage

```
run native:definecurrentprojection INPUT=input.gpkg CRS=EPSG:6346
```

This calls **Define projection**: INPUT=input.gpkg — Input shapefile; CRS=EPSG:6346 — CRS. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:delaunaytriangulation — Delaunay triangulation ★

group: Vector geometry · **niva verb:** delaunay

This algorithm creates a polygon layer with the Delaunay triangulation corresponding to a points layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
TOLERANCE	number	no	0	Tolerance
ADD_ATTRIBUTES	boolean	no	True	Add point IDs to output
OUTPUT	sink	yes	—	Delaunay triangulation

Outputs: OUTPUT (outputVector)

Example usage

```
run native:delaunaytriangulation INPUT=input.gpkg TOLERANCE=0 ADD_ATTRIBUTES=true
↪  OUTPUT=output.gpkg
```

This calls **Delaunay triangulation**: INPUT=input.gpkg — Input layer; TOLERANCE=0 — Tolerance; ADD_ATTRIBUTES=true — Add point IDs to output; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:deletecolumn — Drop field(s) ★

group: Vector table · **niva verb:** dropfields

This algorithm takes a vector layer and generates a new one that has the exact same content but without the selected columns.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
COLUMN	field	yes	—	Fields to drop
OUTPUT	sink	yes	—	Remaining fields

Outputs: OUTPUT (outputVector)

Example usage

```
run native:deletecolumn INPUT=input.gpkg COLUMN=field1 OUTPUT=output.gpkg
```

This calls **Drop field(s)**: INPUT=input.gpkg — Input layer; COLUMN=field1 — Fields to drop; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:deleteduplicategeometries — Delete duplicate geometries

group: Vector general

This algorithm finds duplicated geometries and removes them.

Attributes are not checked, so in case two features have identical geometries but different attributes, only one of them will be added to the result layer.

Optionally, these duplicate features can be saved to a separate output for analysis.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Cleaned
DUPLICATES	sink	no	—	Duplicates

Outputs: OUTPUT (outputVector), DUPLICATES (outputVector), RETAINED_COUNT (outputNumber), DUPLICATE_COUNT (outputNumber)

Example usage

```
run native:deleteduplicategeometries INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Delete duplicate geometries**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:deleteholes — Delete holes

group: Vector geometry

This algorithm takes a polygon layer and removes holes in polygons. It creates a new vector layer in which polygons with holes have been replaced by polygons with only their external ring. Attributes are not modified.

An optional minimum area parameter allows removing only holes which are smaller than a specified area threshold. Leaving this parameter as 0.0 results in all holes being removed.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
MIN_AREA	area	no	0.0	Remove holes with area less than
OUTPUT	sink	yes	—	Cleaned

Outputs: OUTPUT (outputVector)

Example usage

```
run native:deleteholes INPUT=input.gpkg MIN_AREA=0 OUTPUT=output.gpkg
```

This calls **Delete holes**: INPUT=input.gpkg — Input layer; MIN_AREA=0 — Remove holes with area less than; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:densifygeometries — Densify by count

group: Vector geometry

This algorithm takes a polygon or line layer and generates a new one in which the geometries have a larger number of vertices than the original one.

If the geometries have z or m values present then these will be linearly interpolated at the added nodes.

The number of new vertices to add to each feature geometry is specified as an input parameter.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
VERTICES	number	no	1	Number of vertices to add
OUTPUT	sink	yes	—	Densified

Outputs: OUTPUT (outputVector)

Example usage

```
run native:densifygeometries INPUT=input.gpkg VERTICES=1 OUTPUT=output.gpkg
```

This calls **Densify by count**: INPUT=input.gpkg — Input layer; VERTICES=1 — Number of vertices to add; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:densifygeometriesgivenaninterval — Densify by interval ★

group: Vector geometry · **niva verb:** densify

This algorithm takes a polygon or line layer and generates a new one in which the geometries have a larger number of vertices than the original one.

Geometries are densified by adding additional vertices on edges that have a maximum distance of the interval parameter in map units.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
INTERVAL	distance	no	1	Interval between vertices to add
OUTPUT	sink	yes	—	Densified

Outputs: OUTPUT (outputVector)

Example usage

```
run native:densifygeometriesgivenaninterval INPUT=input.gpkg INTERVAL=1 OUTPUT=output.gpkg
```

This calls **Densify by interval**: `INPUT=input.gpkg` — Input layer; `INTERVAL=1` — Interval between vertices to add; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:detectvectorchanges` — Detect dataset changes

group: Vector general

This algorithm compares two vector layers, and determines which features are unchanged, added or deleted between the two. It is designed for comparing two different versions of the same dataset.

When comparing features, the original and revised feature geometries will be compared against each other. Depending on the Geometry Comparison Behavior setting, the comparison will either be made using an exact comparison (where geometries must be an exact match for each other, including the order and count of vertices) or a topological comparison only (where geometries are considered equal if all of their component edges overlap. E.g. lines with the same vertex locations but opposite direction will be considered equal by this method). If the topological comparison is selected then any z or m values present in the geometries will not be compared.

By default, the algorithm compares all attributes from the original and revised features. If the Attributes to Consider for Match parameter is changed, then only the selected attributes will be compared (e.g. allowing users to ignore a timestamp or ID field which is expected to change between the revisions).

If any features in the original or revised layers do not have an associated geometry, then care must be taken to ensure that these features have a unique set of attributes selected for comparison. If this condition is not met, warnings will be raised and the resultant outputs may be misleading.

The algorithm outputs three layers, one containing all features which are considered to be unchanged between the revisions, one containing features deleted from the original layer which are not present in the revised layer, and one containing features added to the revised layer which are not present in the original layer.

Parameter	Type	Required	Default	Description
ORIGINAL	source	yes	—	Original layer
REVISED	source	yes	—	Revised layer
COMPARE_ATTRIBUTES	field	no	—	Attributes to consider for match (or none to compare geometry only)
MATCH_TYPE *(adv)*	enum	no	1	Geometry comparison behavior — options: 0=Exact Match, 1=Tolerant Match (Topological Equality)
UNCHANGED	sink	no	—	Unchanged features
ADDED	sink	no	—	Added features
DELETED	sink	no	—	Deleted features

Outputs: UNCHANGED (outputVector), ADDED (outputVector), DELETED (outputVector), UNCHANGED_COUNT (outputNumber), ADDED_COUNT (outputNumber), DELETED_COUNT (outputNumber)

Example usage

```
run native:detectvectorchanges ORIGINAL=input.gpkg REVISED=input.gpkg
↪  COMPARE_ATTRIBUTES=field1 UNCHANGED=unchanged.gpkg
```

This calls **Detect dataset changes**: ORIGINAL=input.gpkg — Original layer; REVISED=input.gpkg — Revised layer; COMPARE_ATTRIBUTES=field1 — Attributes to consider for match (or none to compare geometry only); UNCHANGED=unchanged.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:difference — Difference ★

group: Vector overlay · **niva verb:** difference

This algorithm extracts features from the Input layer that fall outside, or partially overlap, features in the Overlay layer. Input layer features that partially overlap feature(s) in the Overlay layer are split along those features' boundary and only the portions outside the Overlay layer features are retained.

Attributes are not modified, although properties such as area or length of the features will be modified by the difference operation. If such properties are stored as attributes, those attributes will have to be manually updated.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OVERLAY	source	yes	—	Overlay layer
OUTPUT	sink	yes	—	Difference
GRID_SIZE *(adv)*	number	no	—	Grid size

Outputs: OUTPUT (outputVector)

Example usage

```
run native:difference INPUT=input.gpkg OVERLAY=input.gpkg OUTPUT=output.gpkg
```

This calls **Difference**: INPUT=input.gpkg — Input layer; OVERLAY=input.gpkg — Overlay layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:dissolve — Dissolve ★

group: Vector geometry · **niva verb:** dissolve

This algorithm takes a vector layer and combines their features into new features. One or more attributes can be specified to dissolve features belonging to the same class (having the same value for the specified attributes), alternatively all features can be dissolved in a single one.

All output geometries will be converted to multi geometries. In case the input is a polygon layer, common boundaries of adjacent polygons being dissolved will get erased.

If enabled, the optional ”Keep disjoint features separate” setting will cause features and parts that do not overlap or touch to be exported as separate features (instead of parts of a single multipart feature).

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	no	—	Dissolve field(s)
SEPARATE_DISJOINT *(adv)*	boolean	no	False	Keep disjoint features separate
OUTPUT	sink	yes	—	Dissolved

Outputs: OUTPUT (outputVector)

Example usage

```
run native:dissolve INPUT=input.gpkg FIELD=field1 OUTPUT=output.gpkg
```

This calls **Dissolve**: INPUT=input.gpkg — Input layer; FIELD=field1 — Dissolve field(s); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:distancetonearesthub — Distance to nearest hub

group: Vector analysis

This algorithm computes the distance between features from the source layer to the closest feature from the destination layer.

Distance calculations are based on the feature's bounding box center.

The resulting line layer contains lines linking each origin point with its nearest destination feature.

The resulting point layer contains each origin feature's center point with additional fields indicating the identifier of the nearest destination feature and the distance to it.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Source layer (spokes)
HUBS	source	yes	—	Destination layer (hubs)
FIELD	field	yes	—	Hub layer name attribute
UNIT	enum	no	0	Measurement unit — options: 0=Meters, 1=Feet, 2=Miles, 3=Kilometers, 4=Layer Units
OUTPUT_LINES	sink	no	—	Hub lines
OUTPUT_POINTS	sink	no	—	Hub points

Outputs: OUTPUT_LINES (outputVector), OUTPUT_POINTS (outputVector)

Example usage

```
run native:distancetonearesthub INPUT=input.gpkg HUBS=input.gpkg FIELD=field1 UNIT=0
↪ OUTPUT_LINES=output_lines.gpkg
```

This calls **Distance to nearest hub**: INPUT=input.gpkg — Source layer (spokes); HUBS=input.gpkg — Destination layer (hubs); FIELD=field1 — Hub layer name attribute; UNIT=0 selects *Meters*; OUTPUT_LINES=output_lines.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:downloadgpsdata — Download GPS data from device

group: GPS

This algorithm uses the GPSBabel tool to download data from a GPS device into the GPX standard format.

Parameter	Type	Required	Default	Description
DEVICE	string	yes	—	Device

Parameter	Type	Required	Default	Description
PORT	string	yes	—	Port
FEATURE_TYPE	enum	no	0	Feature type — options: 0=Waypoints, 1=Routes, 2=Tracks
OUTPUT	fileDestination	yes	—	Output

Outputs: OUTPUT (outputFile), OUTPUT_LAYER (outputVector)

Example usage

```
run native:downloadgpsdata DEVICE=value PORT=value FEATURE_TYPE=0 OUTPUT=output.txt
```

This calls **Download GPS data from device**: DEVICE=value — Device; PORT=value — Port; FEATURE_TYPE=0 selects *Waypoints*; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:downloadvectortiles — Download vector tiles

group: Vector tiles

This algorithm downloads vector tiles of the input vector tile layer and saves them in the local vector tile file.

Parameter	Type	Required	Default	Description
INPUT	layer	yes	—	Input layer
EXTENT	extent	yes	—	Extent
MAX_ZOOM	number	no	10	Maximum zoom level to download
TILE_LIMIT	number	no	100	Tile limit
OUTPUT	vectorTileDestination	yes	—	Output

Outputs: OUTPUT (outputVectorTile)

Example usage

```
run native:downloadvectortiles INPUT=input.gpkg EXTENT=0,1000,0,1000 MAX_ZOOM=10
↪  TILE_LIMIT=100 OUTPUT=output.mbtiles
```

This calls **Download vector tiles**: INPUT=input.gpkg — Input layer; EXTENT=0,1000,0,1000 — Extent; MAX_ZOOM=10 — Maximum zoom level to download; TILE_LIMIT=100 — Tile limit; OUTPUT=output.mbtiles is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:dropgeometries — Drop geometries

group: Vector general

This algorithm removes any geometries from an input layer and returns a layer containing only the feature attributes.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Dropped geometries

Outputs: OUTPUT (outputVector)

Example usage

```
run native:dropgeometries INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Drop geometries**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:dropmzvalues — Drop M/Z values

group: Vector geometry

This algorithm can remove any measure (M) or Z values from input geometries.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
DROP_M_VALUES	boolean	no	False	Drop M Values
DROP_Z_VALUES	boolean	no	False	Drop Z Values
OUTPUT	sink	yes	—	Z/M Dropped

Outputs: OUTPUT (outputVector)

Example usage

```
run native:dropmzvalues INPUT=input.gpkg DROP_M_VALUES=false DROP_Z_VALUES=false
↪  OUTPUT=output.gpkg
```

This calls **Drop M/Z values**: INPUT=input.gpkg — Input layer; DROP_M_VALUES=false — Drop M Values; DROP_Z_VALUES=false — Drop Z Values; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:dtmslopebasedfilter — DTM filter (slope-based)

group: Raster terrain analysis

This algorithm can be used to filter a Digital Elevation Model in order to classify its cells into ground and object (non-ground) cells.

The tool uses concepts as described by Vosselman (2000) and is based on the assumption that a large height difference between two nearby cells is unlikely to be caused by a steep slope in the terrain. The probability that the higher cell might be non-ground increases when the distance between the two cells decreases. Therefore the filter defines a maximum height difference (dz_max) between two cells as a function of the distance (d) between the cells (dz_max(d) = d).

A cell is classified as terrain if there is no cell within the kernel radius to which the height difference is larger than the allowed maximum height difference at the distance between these two cells.

The approximate terrain slope (s) parameter is used to modify the filter function to match the overall slope in the study area (dz_max(d) = d * s).

A 5 % confidence interval (ci = 1.65 * sqrt(2 * stddev)) may be used to modify the filter function even further by either relaxing (dz_max(d) = d * s + ci) or amplifying (dz_max(d) = d * s - ci) the filter criterium.

References: Vosselman, G. (2000): Slope based filtering of laser altimetry data. IAPRS, Vol. XXXIII, Part B3, Amsterdam, The Netherlands, 935-942

This algorithm is a port of the SAGA 'DTM Filter (slope-based)' tool.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
RADIUS	number	no	5	Kernel radius (pixels)
TERRAIN_SLOPE	number	no	30	Terrain slope (%, pixel size/vertical units)
FILTER_MODIFICATION	enum	no	0	Filter modification — options: 0=None, 1=Relax filter, 2=Amplify
STANDARD_DEVIATION	number	no	0.1	Standard deviation
CREATE_OPTIONS	string	no	—	Creation options

Parameter	Type	Required	Default	Description
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT_GROUND	rasterDestination	no	—	Output layer (ground)
OUTPUT_NONGROUND	rasterDestination	no	—	Output layer (non-ground objects)

Outputs: OUTPUT_GROUND (outputRaster), OUTPUT_NONGROUND (outputRaster)

Example usage

```
run native:dtmslopebasedfilter INPUT=dem.tif BAND=1 RADIUS=5 TERRAIN_SLOPE=30
 ↪ FILTER_MODIFICATION=0 STANDARD_DEVIATION=0.1 CREATE_OPTIONS=value
 ↪ OUTPUT_GROUND=output_ground.tif
```

This calls **DTM filter (slope-based)**: INPUT=dem.tif — Input layer; BAND=1 — Band number; RADIUS=5 — Kernel radius (pixels); TERRAIN_SLOPE=30 — Terrain slope (%, pixel size/vertical units); FILTER_MODIFICATION=0 selects *None*; STANDARD_DEVIATION=0.1 — Standard deviation; CREATE_OPTIONS=value — Creation options; OUTPUT_GROUND=output_ground.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:dxfexport — Export layers to DXF

group: Vector general

Exports layers to a DXF file. For each layer, you can choose a field whose values are used to split features in generated destination layers in the DXF output.

If no field is chosen, you can still override the output layer name by directly entering a new output layer name in the Configure Layer panel or by preferring layer title (set in layer properties) to layer name.

Parameter	Type	Required	Default	Description
LAYERS	dxflayers	yes	—	Input layers
SYMBOLOGY_MODE	enum	no	0	Symbology mode — options: 0=No Symbology, 1=Feature Symbology, 2=Symbol Layer Symbology
SYMBOLOGY_SCALE	scale	no	1000000	Symbology scale
MAP_THEME	maptheme	no	—	Map theme
ENCODING	enum	no	ISO-8859-1	Encoding — options: 0=ISO-8859-1, 1=ISO-8859-2, 2=ISO-8859-3, 3=ISO-8859-4, 4=ISO-8859-5, 5=ISO-8859-6, 6=ISO-8859-7, 7=ISO-8859-8, 8=ISO-8859-9, 9=Shift_JIS, 10=Big5, 11=CP936, 12=GB2312, 13=ms949, 14=cp850, 15=cp866, 16=cp1250, 17=cp1251, 18=cp1252, 19=cp1253, 20=cp1254, 21=cp1255, 22=cp1256, 23=cp1257, 24=cp1258, 25=macroman
CRS	crs	no	EPSG:4326	CRS
EXTENT	extent	no	—	Extent
SELECTED_FEATURES_ONLY	boolean	no	False	Use only selected features
USE_LAYER_TITLE	boolean	no	False	Use layer title as name
FORCE_2D	boolean	no	False	Force 2D output
MTEXT	boolean	no	True	Export labels as MTEXT elements
EXPORT_LINES_WITH_ZERO_WIDTH	boolean	yes	—	Export lines with zero width
OUTPUT	fileDestination	yes	—	DXF

Outputs: OUTPUT (outputFile)

Example usage

```
run native:dxfexport LAYERS=… EXPORT_LINES_WITH_ZERO_WIDTH=false SYMBOLOGY_MODE=0
↪ SYMBOLOGY_SCALE=1000000 ENCODING=0 CRS=EPSG:6346 OUTPUT=output.txt
```

This calls **Export layers to DXF**: LAYERS=… — Input layers; EXPORT_LINES_WITH_ZERO_WIDTH=false — Export lines with zero width; SYMBOLOGY_MODE=0 selects *No Symbology*; SYMBOLOGY_SCALE=1000000 — Symbology scale; ENCODING=0 selects *ISO-8859-1*; CRS=EPSG:6346 — CRS; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:equaltofrequency — Equal to frequency

group: Raster analysis

This algorithm evaluates on a cell-by-cell basis the frequency (number of times) the values of an input stack of rasters are equal to the value of a value raster. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. The input value layer serves as reference layer for the sample layers. Any NoData cells in the value raster or the data layer stack will result in a NoData cell in the output raster if the ignore NoData parameter is not checked. The output NoData value can be set manually. The output rasters extent and resolution is defined by the input raster layer and is always of int32 type.

Parameter	Type	Required	Default	Description
INPUT_VALUE_RASTER	raster	yes	—	Input value raster
INPUT_VALUE_RASTER_BAND	band	no	1	Value raster band
INPUT_RASTERS	multilayer	yes	—	Input raster layers
IGNORE_NODATA	boolean	no	False	Ignore NoData values
OUTPUT_NODATA_VALUE *(adv)*	number	no	-9999	Output NoData value
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), OCCURRENCE_COUNT (outputNumber), FOUND_LOCATIONS_COUNT (outputNumber), MEAN_FREQUENCY_PER_LOCATION (outputNumber), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber)

Example usage

```
run native:equaltofrequency INPUT_VALUE_RASTER=dem.tif INPUT_RASTERS="a.gpkg;b.gpkg"
↪ INPUT_VALUE_RASTER_BAND=1 IGNORE_NODATA=false CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Equal to frequency**: INPUT_VALUE_RASTER=dem.tif — Input value raster; INPUT_RASTERS="a.gpkg;b.gpkg" — Input raster layers; INPUT_VALUE_RASTER_BAND=1 — Value raster band; IGNORE_NODATA=false — Ignore NoData values; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:explodehstorefield — Explode HStore Field

group: Vector table

This algorithm creates a copy of the input layer and adds a new field for every unique key in the HStore field. The expected field list is an optional comma separated list. By default, all unique keys are added. If this list is specified, only these fields are added and the HStore field is updated.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer

Parameter	Type	Required	Default	Description
FIELD	field	yes	—	HStore field
EXPECTED_FIELDS	string	no	—	Expected list of fields separated by a comma
OUTPUT	sink	yes	—	Exploded

Outputs: OUTPUT (outputVector)

Example usage

```
run native:explodehstorefield INPUT=input.gpkg FIELD=field1 EXPECTED_FIELDS=value
↪  OUTPUT=output.gpkg
```

This calls **Explode HStore Field**: INPUT=input.gpkg — Input layer; FIELD=field1 — HStore field; EXPECTED_FIELDS=value — Expected list of fields separated by a comma; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:explodelines — Explode lines

group: Vector geometry

This algorithm takes a lines layer and creates a new one in which each line is replaced by a set of lines representing the segments in the original line. Each line in the resulting layer contains only a start and an end point, with no intermediate nodes between them.

If the input layer consists of CircularStrings or CompoundCurves, the output layer will be of the same type and contain only single curve segments.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Exploded

Outputs: OUTPUT (outputVector)

Example usage

```
run native:explodelines INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Explode lines**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:exportaddgeometrycolumns — Add geometry attributes

group: Vector geometry

This algorithm computes geometric properties of the features in a vector layer. The algorithm generates a new vector layer with the same content as the input one, but with additional attributes in its attributes table, containing geometric measurements.

Depending on the geometry type of the vector layer, the attributes added to the table will be different.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
METHOD	enum	no	0	Calculate using — options: 0=Cartesian Calculations in Layer's CRS, 1=Cartesian Calculations in Project's CRS, 2=Ellipsoidal Calculations
OUTPUT	sink	yes	—	Added geometry info

Outputs: OUTPUT (outputVector)

Example usage

```
run native:exportaddgeometrycolumns INPUT=input.gpkg METHOD=0 OUTPUT=output.gpkg
```

This calls **Add geometry attributes**: `INPUT=input.gpkg` — Input layer; `METHOD=0` selects *Cartesian Calculations in Layer's CRS*; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:exportlayermetadata` — Export layer metadata

group: Metadata tools

This algorithm exports layer's metadata to a QMD file.

Parameter	Type	Required	Default	Description
INPUT	layer	yes	—	Layer
OUTPUT	fileDestination	yes	—	Output

Outputs: OUTPUT (outputFile)

Example usage

```
run native:exportlayermetadata INPUT=input.gpkg OUTPUT=output.txt
```

This calls **Export layer metadata**: `INPUT=input.gpkg` — Layer; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:exportlayersinformation` — Export layer(s) information

group: Layer tools

This algorithm creates a polygon layer with features corresponding to the extent of selected layer(s).

Additional layer details - CRS, provider name, file path, layer name, subset filter, abstract and attribution - are attached as attributes to each feature.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layer(s)
OUTPUT	sink	yes	—	Output

Outputs: OUTPUT (outputVector)

Example usage

```
run native:exportlayersinformation LAYERS="a.gpkg;b.gpkg" OUTPUT=output.gpkg
```

This calls **Export layer(s) information**: `LAYERS="a.gpkg;b.gpkg"` — Input layer(s); `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:exportmeshedges — Export mesh edges

group: Mesh

This algorithm exports a mesh layer's edges to a line vector layer, with the dataset values on edges as attribute values.

Parameter	Type	Required	Default	Description
INPUT	mesh	yes	—	Input mesh layer
DATASET_GROUPS	meshdatasetgroups	no	[]	Dataset groups
DATASET_TIME	meshdatasettime	yes	—	Dataset time
CRS_OUTPUT	crs	no	—	Output coordinate system
VECTOR_OPTION	enum	no	0	Export vector option — options: 0=Cartesian (x,y), 1=Polar (magnitude,degree), 2=Cartesian and Polar
OUTPUT	sink	yes	—	Output vector layer

Outputs: OUTPUT (outputVector)

Example usage

```
run native:exportmeshedges INPUT=mesh.nc DATASET_TIME=… CRS_OUTPUT=EPSG:6346
↪  VECTOR_OPTION=0 OUTPUT=output.gpkg
```

This calls **Export mesh edges**: INPUT=mesh.nc — Input mesh layer; DATASET_TIME=… — Dataset time; CRS_OUTPUT=EPSG:6346 — Output coordinate system; VECTOR_OPTION=0 selects *Cartesian (x,y)*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:exportmeshfaces — Export mesh faces

group: Mesh

This algorithm exports a mesh layer's faces to a polygon vector layer, with the dataset values on faces as attribute values.

Parameter	Type	Required	Default	Description
INPUT	mesh	yes	—	Input mesh layer
DATASET_GROUPS	meshdatasetgroups	no	[]	Dataset groups
DATASET_TIME	meshdatasettime	yes	—	Dataset time
CRS_OUTPUT	crs	no	—	Output coordinate system
VECTOR_OPTION	enum	no	0	Export vector option — options: 0=Cartesian (x,y), 1=Polar (magnitude,degree), 2=Cartesian and Polar
OUTPUT	sink	yes	—	Output vector layer

Outputs: OUTPUT (outputVector)

Example usage

```
run native:exportmeshfaces INPUT=mesh.nc DATASET_TIME=… CRS_OUTPUT=EPSG:6346
↪  VECTOR_OPTION=0 OUTPUT=output.gpkg
```

This calls **Export mesh faces**: INPUT=mesh.nc — Input mesh layer; DATASET_TIME=… — Dataset time; CRS_OUTPUT=EPSG:6346 — Output coordinate system; VECTOR_OPTION=0 selects *Cartesian (x,y)*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:exportmeshongrid — Export mesh on grid

group: Mesh

This algorithm exports a mesh layer's dataset values to a gridded point vector layer, with the dataset values on each point as attribute values. For data on volume (3D stacked dataset values), the exported dataset values are averaged on faces using the method defined in the mesh layer properties (default is Multi level averaging method). 1D meshes are not supported.

Parameter	Type	Required	Default	Description
INPUT	mesh	yes	—	Input mesh layer
DATASET_GROUPS	meshdatasetgroups	no	[]	Dataset groups
DATASET_TIME	meshdatasettime	yes	—	Dataset time
EXTENT	extent	no	—	Extent
GRID_SPACING	distance	no	10	Grid spacing
CRS_OUTPUT	crs	no	—	Output coordinate system
VECTOR_OPTION	enum	no	0	Export vector option — options: 0=Cartesian (x,y), 1=Polar (magnitude,degree), 2=Cartesian and Polar
OUTPUT	sink	yes	—	Output vector layer

Outputs: OUTPUT (outputVector)

Example usage

```
run native:exportmeshongrid INPUT=mesh.nc DATASET_TIME=… EXTENT=0,1000,0,1000
↪  GRID_SPACING=10 CRS_OUTPUT=EPSG:6346 VECTOR_OPTION=0 OUTPUT=output.gpkg
```

This calls **Export mesh on grid**: INPUT=mesh.nc — Input mesh layer; DATASET_TIME=… — Dataset time; EXTENT=0,1000,0,1000 — Extent; GRID_SPACING=10 — Grid spacing; CRS_OUTPUT=EPSG:6346 — Output coordinate system; VECTOR_OPTION=0 selects *Cartesian (x,y)*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:exportmeshvertices — Export mesh vertices

group: Mesh

This algorithm exports a mesh layer's vertices to a point vector layer, with the dataset values on vertices as attribute values.

Parameter	Type	Required	Default	Description
INPUT	mesh	yes	—	Input mesh layer
DATASET_GROUPS	meshdatasetgroups	no	[]	Dataset groups
DATASET_TIME	meshdatasettime	yes	—	Dataset time
CRS_OUTPUT	crs	no	—	Output coordinate system
VECTOR_OPTION	enum	no	0	Export vector option — options: 0=Cartesian (x,y), 1=Polar (magnitude,degree), 2=Cartesian and Polar
OUTPUT	sink	yes	—	Output vector layer

Outputs: OUTPUT (outputVector)

Example usage

```
run native:exportmeshvertices INPUT=mesh.nc DATASET_TIME=… CRS_OUTPUT=EPSG:6346
↪  VECTOR_OPTION=0 OUTPUT=output.gpkg
```

This calls **Export mesh vertices**: INPUT=mesh.nc — Input mesh layer; DATASET_TIME=… — Dataset time; CRS_OUTPUT=EPSG:6346 — Output coordinate system; VECTOR_OPTION=0 selects *Cartesian (x,y)*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:exporttospreadsheet — Export to spreadsheet

group: Layer tools

This algorithm collects a number of existing layers and exports them to a spreadsheet document.

Optionally the layers can be appended to an existing spreadsheet as additional sheets.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layers
USE_ALIAS	boolean	no	False	Use field aliases as column headings
FORMATTED_VALUES	boolean	no	False	Export formatted values instead of raw values
OUTPUT	fileDestination	yes	—	Destination spreadsheet
OVERWRITE	boolean	no	True	Overwrite existing spreadsheet

Outputs: OUTPUT (outputFile), OUTPUT_LAYERS (outputMultilayer)

Example usage

```
run native:exporttospreadsheet LAYERS="a.gpkg;b.gpkg" USE_ALIAS=false
 ↪  FORMATTED_VALUES=false OVERWRITE=true OUTPUT=output.txt
```

This calls **Export to spreadsheet**: LAYERS="a.gpkg;b.gpkg" — Input layers; USE_ALIAS=false — Use field aliases as column headings; FORMATTED_VALUES=false — Export formatted values instead of raw values; OVERWRITE=true — Overwrite existing spreadsheet; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:extendlines — Extend lines

group: Vector geometry

This algorithm extends line geometries by a specified amount at the start and end of the line. Lines are extended using the bearing of the first and last segment in the line.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
START_DISTANCE	distance	no	0.0	Start distance
END_DISTANCE	distance	no	0.0	End distance
OUTPUT	sink	yes	—	Extended

Outputs: OUTPUT (outputVector)

Example usage

```
run native:extendlines INPUT=input.gpkg START_DISTANCE=0 END_DISTANCE=0 OUTPUT=output.gpkg
```

This calls **Extend lines**: INPUT=input.gpkg — Input layer; START_DISTANCE=0 — Start distance; END_DISTANCE=0 — End distance; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:extenttolayer — Create layer from extent

group: Vector creation

This algorithm creates a new vector layer that contains a single feature with geometry matching an extent parameter.

It can be used in models to convert an extent into a layer which can be used for other algorithms which require a layer based input.

Parameter	Type	Required	Default	Description
INPUT	extent	yes	—	Extent
OUTPUT	sink	yes	—	Extent

Outputs: OUTPUT (outputVector)

Example usage

```
run native:extenttolayer INPUT=0,1000,0,1000 OUTPUT=output.gpkg
```

This calls **Create layer from extent**: INPUT=0,1000,0,1000 — Extent; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:extractbinary — Extract binary field

group: Vector table

This algorithm extracts contents from a binary field, saving them to individual files.

Filenames can be generated using values taken from an attribute in the source table or based on a more complex expression.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	yes	—	Binary field
FILENAME	expression	yes	—	File name
FOLDER	folderDestination	yes	—	Destination folder

Outputs: FOLDER (outputFolder)

Example usage

```
run native:extractbinary INPUT=input.gpkg FIELD=field1 FILENAME="value > 0" FOLDER=folder/
```

This calls **Extract binary field**: INPUT=input.gpkg — Input layer; FIELD=field1 — Binary field; FILENAME="value > 0" — File name; FOLDER=folder/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:extractbyattribute — Extract by attribute

group: Vector selection

This algorithm creates a new vector layer that only contains matching features from an input layer. The criteria for adding features to the resulting layer is defined based on the values of an attribute from the input layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	yes	—	Selection attribute
OPERATOR	enum	no	0	Operator — options: 0==, 1=≠, 2=>, 3=≥, 4=<, 5=≤, 6=begins with, 7=contains, 8=is null, 9=is not null, 10=does not contain
VALUE	string	no	—	Value
OUTPUT	sink	yes	—	Extracted (attribute)
FAIL_OUTPUT	sink	no	—	Extracted (non-matching)

Outputs: OUTPUT (outputVector), FAIL_OUTPUT (outputVector)

Example usage

```
run native:extractbyattribute INPUT=input.gpkg FIELD=field1 OPERATOR=0 VALUE=value
↪ OUTPUT=output.gpkg
```

This calls **Extract by attribute**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — Selection attribute; `OPERATOR=0` selects =; `VALUE=value` — Value; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:extractbyexpression` — Extract by expression ★

group: Vector selection · **niva verb:** `filter`

This algorithm creates a new vector layer that only contains matching features from an input layer. The criteria for adding features to the resulting layer is based on a QGIS expression.

For help with QGIS expression functions, see the inbuilt help for specific functions which is available in the expression builder.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`EXPRESSION`	expression	yes	—	Expression
`OUTPUT`	sink	yes	—	Matching features
`FAIL_OUTPUT`	sink	no	—	Non-matching

Outputs: `OUTPUT` (outputVector), `FAIL_OUTPUT` (outputVector)

Example usage

```
run native:extractbyexpression INPUT=input.gpkg EXPRESSION="value > 0" OUTPUT=output.gpkg
```

This calls **Extract by expression**: `INPUT=input.gpkg` — Input layer; `EXPRESSION="value > 0"` — Expression; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:extractbyextent` — Extract/clip by extent

group: Vector overlay

This algorithm creates a new vector layer that only contains features which fall within a specified extent. Any features which intersect the extent will be included.

Optionally, feature geometries can also be clipped to the extent. If this option is selected, then the output geometries will automatically be converted to multi geometries to ensure uniform output geometry types.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`EXTENT`	extent	yes	—	Extent
`CLIP`	boolean	no	False	Clip features to extent
`OUTPUT`	sink	yes	—	Extracted

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:extractbyextent INPUT=input.gpkg EXTENT=0,1000,0,1000 CLIP=false
↪ OUTPUT=output.gpkg
```

This calls **Extract/clip by extent**: `INPUT=input.gpkg` — Input layer; `EXTENT=0,1000,0,1000` — Extent; `CLIP=false` — Clip features to extent; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:extractbylocation` — Extract by location ★

group: Vector selection · **niva verb:** `selectloc`

This algorithm creates a new vector layer that only contains matching features from an input layer. The criteria for adding features to the resulting layer is defined based on the spatial relationship between each feature and the features in an additional layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Extract features from
PREDICATE	enum	no	[0]	Where the features (geometric predicate) — options: 0=intersect, 1=contain, 2=disjoint, 3=equal, 4=touch, 5=overlap, 6=are within, 7=cross
INTERSECT	source	yes	—	By comparing to the features from
OUTPUT	sink	yes	—	Extracted (location)

Outputs: OUTPUT (outputVector)

Example usage

```
run native:extractbylocation INPUT=input.gpkg INTERSECT=input.gpkg PREDICATE=0
↪  OUTPUT=output.gpkg
```

This calls **Extract by location**: INPUT=input.gpkg — Extract features from; INTERSECT=input.gpkg — By comparing to the features from; PREDICATE=0 selects *intersect*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:extractlabels` — Extract labels

group: Cartography

This algorithm extracts label information from a rendered map at a given extent and scale.

If a map theme is provided, the rendered map will match the visibility and symbology of that theme. If left blank, all visible layers from the project will be used.

Extracted label information include: position (served as point geometries), the associated layer name and feature ID, label text, rotation (in degree, clockwise), multiline alignment, and font details.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Map extent
SCALE	scale	yes	—	Map scale
MAP_THEME	maptheme	no	—	Map theme
INCLUDE_UNPLACED	boolean	no	True	Include unplaced labels
DPI *(adv)*	number	no	96.0	Map resolution (in DPI)
OUTPUT	sink	yes	—	Extracted labels

Outputs: OUTPUT (outputVector)

Example usage

```
run native:extractlabels EXTENT=0,1000,0,1000 SCALE=10 INCLUDE_UNPLACED=true
↪  OUTPUT=output.gpkg
```

This calls **Extract labels**: EXTENT=0,1000,0,1000 — Map extent; SCALE=10 — Map scale; INCLUDE_UNPLACED=true — Include unplaced labels; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:extractmvalues — Extract M values

group: Vector geometry

Extracts m values from geometries into feature attributes.

By default only the m value from the first vertex of each feature is extracted, however the algorithm can optionally calculate statistics on all of the geometry's m values, including sums, means, and minimums and maximums

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
SUMMARIES	enum	no	[0]	Summaries to calculate — options: 0=First, 1=Last, 2=Count, 3=Sum, 4=Mean, 5=Median, 6=St dev (pop), 7=Minimum, 8=Maximum, 9=Range, 10=Minority, 11=Majority, 12=Variety, 13=Q1, 14=Q3, 15=IQR
COLUMN_PREFIX	string	no	m_	Output column prefix
OUTPUT	sink	yes	—	Extracted

Outputs: OUTPUT (outputVector)

Example usage

```
run native:extractmvalues INPUT=input.gpkg SUMMARIES=0 COLUMN_PREFIX=m_ OUTPUT=output.gpkg
```

This calls **Extract M values**: INPUT=input.gpkg — Input layer; SUMMARIES=0 selects *First*; COLUMN_PREFIX=m_ — Output column prefix; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:extractnetworkendpoints — Extract network end points

group: Network analysis

This algorithm extracts the end points (nodes) from a network line layer.

Two definitions are available for identifying end points:

1. Nodes with only all incoming or all outgoing edges: Identifies 'Source' or 'Sink' nodes based on the direction of flow. These are nodes where flow can start (only outgoing) or stop (only incoming).
2. Nodes connected to a single edge: Identifies topological 'dead-ends' or 'dangles', regardless of directionality. This checks if the node is connected to only one other distinct node.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Vector layer representing network
DIRECTION_FIELD *(adv)*	field	no	—	Direction field
VALUE_FORWARD *(adv)*	string	no	—	Value for forward direction
VALUE_BACKWARD *(adv)*	string	no	—	Value for backward direction
VALUE_BOTH *(adv)*	string	no	—	Value for both directions
DEFAULT_DIRECTION *(adv)*	enum	no	2	Default direction — options: 0=Forward direction, 1=Backward direction, 2=Both directions
TOLERANCE *(adv)*	distance	no	0	Topology tolerance
ENDPOINT_DEFINITION	enum	no	1	End point definition — options: 0=Extract Nodes with only All Incoming or All Outgoing Edges, 1=Extract Nodes Connected to a Single Edge
OUTPUT	sink	yes	—	Network endpoints

Outputs: OUTPUT (outputVector)

Example usage

```
run native:extractnetworkendpoints INPUT=input.gpkg ENDPOINT_DEFINITION=1
↪  OUTPUT=output.gpkg
```

This calls **Extract network end points**: `INPUT=input.gpkg` — Vector layer representing network; `ENDPOINT_DEFINITION=1` selects *Extract Nodes Connected to a Single Edge*; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:extractspecificvertices` — Extract specific vertices

group: Vector geometry

This algorithm takes a vector layer and generates a point layer with points representing specific vertices in the input geometries. For instance, this algorithm can be used to extract the first or last vertices in the geometry. The attributes associated to each point are the same ones associated to the feature that the point belongs to.

The vertex indices parameter accepts a comma separated string specifying the indices of the vertices to extract. The first vertex corresponds to an index of 0, the second vertex has an index of 1, etc. Negative indices can be used to find vertices at the end of the geometry, e.g., an index of -1 corresponds to the last vertex, -2 corresponds to the second last vertex, etc.

Additional fields are added to the points indicating the specific vertex position (e.g., 0, -1, etc), the original vertex index, the vertex's part and its index within the part (as well as its ring for polygons), distance along the original geometry and bisector angle of vertex for the original geometry.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`VERTICES`	string	no	0	Vertex indices
`OUTPUT`	sink	yes	—	Vertices

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:extractspecificvertices INPUT=input.gpkg VERTICES=0 OUTPUT=output.gpkg
```

This calls **Extract specific vertices**: `INPUT=input.gpkg` — Input layer; `VERTICES=0` — Vertex indices; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:extractvertices` — Extract vertices ★

group: Vector geometry · **niva verb:** `vertices`

This algorithm takes a vector layer and generates a point layer with points representing the vertices of the input geometries. The attributes associated to each point are the same ones associated to the feature that the point belongs to.

Additional fields are added to the point indicating the vertex index (beginning at 0), the vertex's part and its index within the part (as well as its ring for polygons), distance along original geometry and bisector angle of vertex for original geometry.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`OUTPUT`	sink	yes	—	Vertices

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:extractvertices INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Extract vertices**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:extractwithindistance` — Extract within distance

group: Vector selection

This algorithm creates a new vector layer that only contains matching features from an input layer. Features are copied wherever they are within the specified maximum distance from the features in an additional reference layer.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Extract features from
`REFERENCE`	source	yes	—	By comparing to the features from
`DISTANCE`	distance	no	100	Where the features are within
`OUTPUT`	sink	yes	—	Extracted (location)

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:extractwithindistance INPUT=input.gpkg REFERENCE=input.gpkg DISTANCE=100
↪  OUTPUT=output.gpkg
```

This calls **Extract within distance**: `INPUT=input.gpkg` — Extract features from; `REFERENCE=input.gpkg` — By comparing to the features from; `DISTANCE=100` — Where the features are within; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:extractzvalues` — Extract Z values

group: Vector geometry

Extracts z values from geometries into feature attributes.

By default only the z value from the first vertex of each feature is extracted, however the algorithm can optionally calculate statistics on all of the geometry's z values, including sums, means, and minimums and maximums

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`SUMMARIES`	enum	no	[0]	Summaries to calculate — options: 0=First, 1=Last, 2=Count, 3=Sum, 4=Mean, 5=Median, 6=St dev (pop), 7=Minimum, 8=Maximum, 9=Range, 10=Minority, 11=Majority, 12=Variety, 13=Q1, 14=Q3, 15=IQR
`COLUMN_PREFIX`	string	no	z_	Output column prefix
`OUTPUT`	sink	yes	—	Extracted

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:extractzvalues INPUT=input.gpkg SUMMARIES=0 COLUMN_PREFIX=z_ OUTPUT=output.gpkg
```

This calls **Extract Z values**: `INPUT=input.gpkg` — Input layer; `SUMMARIES=0` selects *First*; `COLUMN_PREFIX=z_` — Output column prefix; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:fieldcalculator` — Field calculator

group: Vector table

This algorithm computes a new vector layer with the same features of the input layer, but either overwriting an existing attribute or adding an additional attribute. The values of this field are computed from each feature using an expression, based on the properties and attributes of the feature. Note that if ”Field name” is an existing field in the layer then all the rest of the field settings are ignored.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`FIELD_NAME`	string	yes	—	Field name
`FIELD_TYPE`	enum	no	0	Result field type — options: 0=Decimal (double), 1=Integer (32 bit), 2=Text (string), 3=Date, 4=Time, 5=Date & Time, 6=Boolean, 7=Binary Object (BLOB), 8=String List, 9=Integer List, 10=Decimal (double) List
`FIELD_LENGTH`	number	no	0	Result field length
`FIELD_PRECISION`	number	no	0	Result field precision
`FORMULA`	expression	yes	—	Formula
`OUTPUT`	sink	yes	—	Calculated

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:fieldcalculator INPUT=input.gpkg FIELD_NAME=value FORMULA="value > 0"
 ↪  FIELD_TYPE=0 FIELD_LENGTH=0 FIELD_PRECISION=0 OUTPUT=output.gpkg
```

This calls **Field calculator**: `INPUT=input.gpkg` — Input layer; `FIELD_NAME=value` — Field name; `FORMULA="value > 0"` — Formula; `FIELD_TYPE=0` selects *Decimal (double)*; `FIELD_LENGTH=0` — Result field length; `FIELD_PRECISION=0` — Result field precision; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:filedownloader` — Download file via HTTP(S)

group: File tools

This algorithm downloads a URL to the file system with an HTTP(S) GET or POST request

Parameter	Type	Required	Default	Description
`URL`	string	yes	—	URL
`METHOD *(adv)*`	enum	no	0	Method — options: 0=GET, 1=POST
`DATA *(adv)*`	string	no	—	Data
`OUTPUT`	fileDestination	yes	—	File destination

Outputs: `OUTPUT` (outputFile)

Example usage

```
run native:filedownloader URL=value OUTPUT=output.txt
```

This calls **Download file via HTTP(S)**: `URL=value` — URL; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fillnodata — Fill NoData cells

group: Raster tools

This algorithm resets the NoData values in the input raster to a chosen value, resulting in a raster dataset with no NoData pixels. This value can be set by the user using the Fill value parameter. The algorithm respects the input raster data type (eg. a floating point fill value will be truncated when applied to an integer raster).

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Raster input
BAND	band	no	1	Band Number
FILL_VALUE	number	no	1	Fill value
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:fillnodata INPUT=dem.tif BAND=1 FILL_VALUE=1 CREATE_OPTIONS=value
↪  OUTPUT=output.tif
```

This calls **Fill NoData cells**: INPUT=dem.tif — Raster input; BAND=1 — Band Number; FILL_VALUE=1 — Fill value; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fillsinkswangliu — Fill sinks (Wang & Liu)

group: Raster terrain analysis

This algorithm uses a method proposed by Wang & Liu to identify and fill surface depressions in digital elevation models.

The method was enhanced to allow the creation of hydrologically sound elevation models, i.e. not only to fill the depression(s) but also to preserve a downward slope along the flow path. If desired, this is accomplished by preserving a minimum slope gradient (and thus elevation difference) between cells.

References: Wang, L. & H. Liu (2006): An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling. International Journal of Geographical Information Science, Vol. 20, No. 2: 193-213.

This algorithm is a port of the SAGA 'Fill Sinks (Wang & Liu)' tool.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
MIN_SLOPE	number	no	0.1	Minimum slope (degrees)
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT_FILLED_DEM	rasterDestination	no	—	Output layer (filled DEM)
OUTPUT_FLOW_DIRECTIONS	rasterDestination	no	—	Output layer (flow directions)
OUTPUT_WATERSHED_BASINS	rasterDestination	no	—	Output layer (watershed basins)

Outputs: OUTPUT_FILLED_DEM (outputRaster), OUTPUT_FLOW_DIRECTIONS (outputRaster), OUTPUT_WATERSHED_BASINS (outputRaster)

Example usage

```
run native:fillsinkswangliu INPUT=dem.tif BAND=1 MIN_SLOPE=0.1
↪ OUTPUT_FILLED_DEM=output_filled_dem.tif
```

This calls **Fill sinks (Wang & Liu)**: `INPUT=dem.tif` — Input layer; `BAND=1` — Band number; `MIN_SLOPE=0.1` — Minimum slope (degrees); `OUTPUT_FILLED_DEM=output_filled_dem.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:filter` — Feature filter

group: Modeler tools

This algorithm filters features from the input layer and redirects them to one or more outputs.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer

Example usage

```
run native:filter INPUT=input.gpkg
```

This calls **Feature filter**: `INPUT=input.gpkg` — Input layer. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:filterbygeometry` — Filter by geometry type

group: Vector selection

This algorithm filters features by their geometry type. Incoming features will be directed to different outputs based on whether they have a point, line or polygon geometry.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`POINTS`	sink	no	—	Point features
`LINES`	sink	no	—	Line features
`POLYGONS`	sink	no	—	Polygon features
`NO_GEOMETRY`	sink	no	—	Features with no geometry

Outputs: `POINTS` (outputVector), `LINES` (outputVector), `POLYGONS` (outputVector), `NO_GEOMETRY` (outputVector), `POINT_COUNT` (outputNumber), `LINE_COUNT` (outputNumber), `POLYGON_COUNT` (outputNumber), `NO_GEOMETRY_COUNT` (outputNumber)

Example usage

```
run native:filterbygeometry INPUT=input.gpkg POINTS=points.gpkg
```

This calls **Filter by geometry type**: `INPUT=input.gpkg` — Input layer; `POINTS=points.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:filterlayersbytype` — Filter layers by type

group: Modeler tools

This algorithm filters layer by their type. Incoming layers will be directed to different outputs based on whether they are a vector or raster layer.

Parameter	Type	Required	Default	Description
INPUT	layer	yes	—	Input layer
VECTOR	vectorDestination	no	—	Vector features
RASTER	rasterDestination	no	—	Raster layer

Outputs: VECTOR (outputVector), RASTER (outputRaster)

Example usage

```
run native:filterlayersbytype INPUT=input.gpkg VECTOR=vector.gpkg
```

This calls **Filter layers by type**: INPUT=input.gpkg — Input layer; VECTOR=vector.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:filterverticesbym — Filter vertices by M value

group: Vector geometry

Filters away vertices based on their m-value, returning geometries with only vertex points that have a m-value ≥ the specified minimum value and ≤ the maximum value.

If the minimum value is not specified then only the maximum value is tested, and similarly if the maximum value is not specified then only the minimum value is tested.

Depending on the input geometry attributes and the filters used, the resultant geometries created by this algorithm may no longer be valid.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
MIN	number	no	—	Minimum
MAX	number	no	—	Maximum
OUTPUT	sink	yes	—	Filtered

Outputs: OUTPUT (outputVector)

Example usage

```
run native:filterverticesbym INPUT=input.gpkg MIN=10 MAX=10 OUTPUT=output.gpkg
```

This calls **Filter vertices by M value**: INPUT=input.gpkg — Input layer; MIN=10 — Minimum; MAX=10 — Maximum; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:filterverticesbyz — Filter vertices by Z value

group: Vector geometry

Filters away vertices based on their z-value, returning geometries with only vertex points that have a z-value ≥ the specified minimum value and ≤ the maximum value.

If the minimum value is not specified then only the maximum value is tested, and similarly if the maximum value is not specified then only the minimum value is tested.

Depending on the input geometry attributes and the filters used, the resultant geometries created by this algorithm may no longer be valid.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
MIN	number	no	—	Minimum

Parameter	Type	Required	Default	Description
MAX	number	no	—	Maximum
OUTPUT	sink	yes	—	Filtered

Outputs: OUTPUT (outputVector)

Example usage

```
run native:filterverticesbyz INPUT=input.gpkg MIN=10 MAX=10 OUTPUT=output.gpkg
```

This calls **Filter vertices by Z value**: INPUT=input.gpkg — Input layer; MIN=10 — Minimum; MAX=10 — Maximum; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:findprojection — Find projection

group: Vector general

Creates a list of possible candidate coordinate reference systems for a layer with an unknown projection.

The expected area which the layer should reside in must be specified via the target area parameter.

The algorithm operates by testing the layer's extent in every known reference system and listing any in which the bounds would fall within the target area if the layer was in this projection.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
TARGET_AREA	extent	yes	—	Target area for layer
TARGET_AREA_CRS	crs	no	—	Target area CRS
OUTPUT	sink	yes	—	CRS candidates

Outputs: OUTPUT (outputVector)

Example usage

```
run native:findprojection INPUT=input.gpkg TARGET_AREA=0,1000,0,1000
↪  TARGET_AREA_CRS=EPSG:6346 OUTPUT=output.gpkg
```

This calls **Find projection**: INPUT=input.gpkg — Input layer; TARGET_AREA=0,1000,0,1000 — Target area for layer; TARGET_AREA_CRS=EPSG:6346 — Target area CRS; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fixgeometries — Fix geometries ★

group: Vector geometry · **niva verb:** fixgeom

This algorithm attempts to create a valid representation of a given invalid geometry without losing any of the input vertices. Already-valid geometries are returned without further intervention. Always outputs multi-geometry layer.

NOTE: M values will be dropped from the output.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
METHOD	enum	no	1	Repair method — options: 0=Linework, 1=Structure
OUTPUT	sink	yes	—	Fixed geometries

Outputs: OUTPUT (outputVector)

Example usage

```
run native:fixgeometries INPUT=input.gpkg METHOD=1 OUTPUT=output.gpkg
```

This calls **Fix geometries**: INPUT=input.gpkg — Input layer; METHOD=1 selects *Structure*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fixgeometryangle — Delete small angles

group: Fix geometry

This algorithm deletes vertices based on an error layer from the ”Small angles” algorithm in the ”Check geometry” section. When deletion of a vertex results in a duplicate vertex (when a spike vertex is deleted), the duplicate vertex is deleted to keep a single vertex and preserve topology.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ERRORS	source	yes	—	Error layer
UNIQUE_ID	field	no	id	Field of original feature unique identifier
PART_IDX	field	no	gc_partidx	Field of part index
RING_IDX	field	no	gc_ringidx	Field of ring index
VERTEX_IDX	field	no	gc_vertidx	Field of vertex index
OUTPUT	sink	yes	—	Small angle fixed layer
REPORT	sink	yes	—	Report layer from fixing small angles
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometryangle INPUT=input.gpkg ERRORS=input.gpkg UNIQUE_ID=id
↪  PART_IDX=gc_partidx RING_IDX=gc_ringidx VERTEX_IDX=gc_vertidx OUTPUT=output.gpkg
```

This calls **Delete small angles**: INPUT=input.gpkg — Input layer; ERRORS=input.gpkg — Error layer; UNIQUE_ID=id — Field of original feature unique identifier; PART_IDX=gc_partidx — Field of part index; RING_IDX=gc_ringidx — Field of ring index; VERTEX_IDX=gc_vertidx — Field of vertex index; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fixgeometryarea — Fix small polygons

group: Fix geometry

This algorithm merges neighboring polygons according to the chosen method, based on an error layer from the ”Small polygons” algorithm in the ”Check geometry” section.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ERRORS	source	yes	—	Error layer
METHOD	enum	yes	—	Method — options: 0=Merge with neighboring polygon with longest shared edge, 1=Merge with neighboring polygon with largest area, 2=Merge with neighboring polygon with identical attribute value, if any, or leave as is

Parameter	Type	Required	Default	Description
MERGE_ATTRIBUTE	field	no	—	Field to consider when merging polygons with the identical attribute method
UNIQUE_ID	field	no	id	Field of original feature unique identifier
PART_IDX	field	no	gc_partidx	Field of part index
RING_IDX	field	no	gc_ringidx	Field of ring index
VERTEX_IDX	field	no	gc_vertidx	Field of vertex index
OUTPUT	sink	yes	—	Small polygons merged layer
REPORT	sink	yes	—	Report layer from merging small polygons
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometryarea INPUT=input.gpkg ERRORS=input.gpkg METHOD=0
↪ MERGE_ATTRIBUTE=field1 UNIQUE_ID=id PART_IDX=gc_partidx RING_IDX=gc_ringidx
↪ OUTPUT=output.gpkg
```

This calls **Fix small polygons**: INPUT=input.gpkg — Input layer; ERRORS=input.gpkg — Error layer; METHOD=0 selects *Merge with neighboring polygon with longest shared edge*; MERGE_ATTRIBUTE=field1 — Field to consider when merging polygons with the identical attribute method; UNIQUE_ID=id — Field of original feature unique identifier; PART_IDX=gc_partidx — Field of part index; RING_IDX=gc_ringidx — Field of ring index; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fixgeometrydeletefeatures — Delete features

group: Fix geometry

This algorithm deletes error features listed in the errors layer from an algorithm in the ”Check geometry” section. The required inputs are the original layer used in the check algorithm, its unique id field, and its corresponding errors layer.

For instance, it can be used after the following check algorithms to delete error features:Feature inside polygonDegenerate polygonsSmall segmentsDuplicated geometriesetc.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ERRORS	source	yes	—	Error layer
UNIQUE_ID	field	yes	—	Field of original feature unique identifier
OUTPUT	sink	yes	—	Cleaned layer
REPORT	sink	yes	—	Report layer from deleting features

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometrydeletefeatures INPUT=input.gpkg ERRORS=input.gpkg UNIQUE_ID=field1
↪ OUTPUT=output.gpkg
```

This calls **Delete features**: INPUT=input.gpkg — Input layer; ERRORS=input.gpkg — Error layer; UNIQUE_ID=field1 — Field of original feature unique identifier; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:fixgeometryduplicatenodes` — Delete duplicated vertices

group: Fix geometry

This algorithm delete duplicate vertices based on an error layer from the ”Duplicated vertices” algorithm in the ”Check geometry” section.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ERRORS	source	yes	—	Error layer
UNIQUE_ID	field	no	id	Field of original feature unique identifier
PART_IDX	field	no	gc_partidx	Field of part index
RING_IDX	field	no	gc_ringidx	Field of ring index
VERTEX_IDX	field	no	gc_vertidx	Field of vertex index
OUTPUT	sink	yes	—	Fixed duplicate vertices layer
REPORT	sink	yes	—	Report layer from fixing duplicate vertices
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometryduplicatenodes INPUT=input.gpkg ERRORS=input.gpkg UNIQUE_ID=id
 ↪  PART_IDX=gc_partidx RING_IDX=gc_ringidx VERTEX_IDX=gc_vertidx OUTPUT=output.gpkg
```

This calls **Delete duplicated vertices**: INPUT=input.gpkg — Input layer; ERRORS=input.gpkg — Error layer; UNIQUE_ID=id — Field of original feature unique identifier; PART_IDX=gc_partidx — Field of part index; RING_IDX=gc_ringidx — Field of ring index; VERTEX_IDX=gc_vertidx — Field of vertex index; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:fixgeometrygap` — Fill gaps

group: Fix geometry

This algorithm fills the gaps based on a gap and neighbors layer from the ”Small gaps” algorithm in the ”Check geometry” section.

3 different fixing methods are available, which will give different results.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
NEIGHBORS	source	yes	—	Neighbors layer
GAPS	source	yes	—	Gaps layer
METHOD	enum	yes	—	Method — options: 0=Add to longest shared edge, 1=Create new feature, 2=Add to largest neighbouring area
UNIQUE_ID	field	yes	—	Field of original feature unique identifier
ERROR_ID_IDX	field	no	gc_errorid	Field of error id
OUTPUT	sink	yes	—	Gaps-filled layer
REPORT	sink	yes	—	Report layer from fixing gaps
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometrygap INPUT=input.gpkg NEIGHBORS=input.gpkg GAPS=input.gpkg METHOD=0
 ↪  UNIQUE_ID=field1 ERROR_ID_IDX=gc_errorid OUTPUT=output.gpkg
```

This calls **Fill gaps**: INPUT=input.gpkg — Input layer; NEIGHBORS=input.gpkg — Neighbors layer; GAPS=input.gpkg — Gaps layer; METHOD=0 selects *Add to longest shared edge*; UNIQUE_ID=field1 — Field of original feature unique identifier; ERROR_ID_IDX=gc_errorid — Field of error id; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fixgeometryhole — Fill holes

group: Fix geometry

This algorithm deletes holes based on an error layer from the ”Holes” algorithm in the ”Check geometry” section.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ERRORS	source	yes	—	Error layer
UNIQUE_ID	field	no	id	Field of original feature unique identifier
PART_IDX	field	no	gc_partidx	Field of part index
RING_IDX	field	no	gc_ringidx	Field of ring index
VERTEX_IDX	field	no	gc_vertidx	Field of vertex index
OUTPUT	sink	yes	—	Holes-filled layer
REPORT	sink	yes	—	Report layer from fixing holes
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometryhole INPUT=input.gpkg ERRORS=input.gpkg UNIQUE_ID=id
 ↪  PART_IDX=gc_partidx RING_IDX=gc_ringidx VERTEX_IDX=gc_vertidx OUTPUT=output.gpkg
```

This calls **Fill holes**: INPUT=input.gpkg — Input layer; ERRORS=input.gpkg — Error layer; UNIQUE_ID=id — Field of original feature unique identifier; PART_IDX=gc_partidx — Field of part index; RING_IDX=gc_ringidx — Field of ring index; VERTEX_IDX=gc_vertidx — Field of vertex index; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fixgeometrymissingvertex — Add missing vertices along borders

group: Fix geometry

This algorithm adds the missing vertices along polygons borders, based on an error layer from the ”Missing vertices along borders” algorithm in the ”Check geometry” section.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ERRORS	source	yes	—	Error layer
UNIQUE_ID	field	no	id	Field of original feature unique identifier
PART_IDX	field	no	gc_partidx	Field of part index
RING_IDX	field	no	gc_ringidx	Field of ring index
VERTEX_IDX	field	no	gc_vertidx	Field of vertex index
OUTPUT	sink	yes	—	Border vertices fixed layer
REPORT	sink	yes	—	Report layer from fixing border vertices
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometrymissingvertex INPUT=input.gpkg ERRORS=input.gpkg UNIQUE_ID=id
 ↪  PART_IDX=gc_partidx RING_IDX=gc_ringidx VERTEX_IDX=gc_vertidx OUTPUT=output.gpkg
```

This calls **Add missing vertices along borders**: INPUT=input.gpkg — Input layer; ERRORS=input.gpkg — Error layer; UNIQUE_ID=id — Field of original feature unique identifier; PART_IDX=gc_partidx — Field of part index; RING_IDX=gc_ringidx — Field of ring index; VERTEX_IDX=gc_vertidx — Field of vertex index; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fixgeometrymultipart — Convert to strictly multipart

group: Fix geometry

This algorithm converts multipart geometries that consists of only one geometry into singlepart geometries, based on an error layer from the ”Strict multipart” algorithm in the ”Check geometry” section.

This algorithm does not change the layer geometry type, which will remain multipart.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ERRORS	source	yes	—	Error layer
UNIQUE_ID	field	no	id	Field of original feature unique identifier
OUTPUT	sink	yes	—	Strictly-multipart layer
REPORT	sink	yes	—	Report layer from fixing multiparts
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometrymultipart INPUT=input.gpkg ERRORS=input.gpkg UNIQUE_ID=id
 ↪  OUTPUT=output.gpkg
```

This calls **Convert to strictly multipart**: INPUT=input.gpkg — Input layer; ERRORS=input.gpkg — Error layer; UNIQUE_ID=id — Field of original feature unique identifier; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fixgeometryoverlap — Delete overlaps

group: Fix geometry

This algorithm deletes overlap sections based on an error layer from the ”Overlap” algorithm in the ”Check geometry” section.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ERRORS	source	yes	—	Error layer
UNIQUE_ID	field	yes	—	Field of original feature unique identifier
OVERLAP_FEATURE_UNIQUE_IDX	field	yes	—	Field of overlap feature unique identifier
ERROR_VALUE_ID	field	no	gc_error	Field of error value
OUTPUT	sink	yes	—	No-overlap layer
REPORT	sink	yes	—	Report layer from fixing overlaps
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometryoverlap INPUT=input.gpkg ERRORS=input.gpkg UNIQUE_ID=field1
↪ OVERLAP_FEATURE_UNIQUE_IDX=field1 ERROR_VALUE_ID=gc_error OUTPUT=output.gpkg
```

This calls **Delete overlaps**: INPUT=input.gpkg — Input layer; ERRORS=input.gpkg — Error layer; UNIQUE_ID=field1 — Field of original feature unique identifier; OVERLAP_FEATURE_UNIQUE_IDX=field1 — Field of overlap feature unique identifier; ERROR_VALUE_ID=gc_error — Field of error value; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fixgeometryselfintersection — Split self-intersecting geometries

group: Fix geometry

This algorithm splits self intersecting lines or polygons according to the chosen method, based on an error layer from the ”Self-intersections” algorithm in the ”Check geometry” section.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ERRORS	source	yes	—	Error layer
METHOD	enum	yes	—	Method — options: 0=Split feature into a multi-part feature, 1=Split feature into multiple single-part features
UNIQUE_ID	field	yes	—	Field of original feature unique identifier
PART_IDX	field	no	gc_partidx	Field of part index
RING_IDX	field	no	gc_ringidx	Field of ring index
VERTEX_IDX	field	no	gc_vertidx	Field of vertex index
SEGMENT_1	field	no	gc_segment_1	Field of segment 1
SEGMENT_2	field	no	gc_segment_2	Field of segment 2
OUTPUT	sink	yes	—	Self-intersections fixed layer
REPORT	sink	yes	—	Report layer from fixing self-intersections
TOLERANCE *(adv)*	number	no	8	Tolerance

Outputs: OUTPUT (outputVector), REPORT (outputVector)

Example usage

```
run native:fixgeometryselfintersection INPUT=input.gpkg ERRORS=input.gpkg METHOD=0
↪ UNIQUE_ID=field1 PART_IDX=gc_partidx RING_IDX=gc_ringidx VERTEX_IDX=gc_vertidx
↪ OUTPUT=output.gpkg
```

This calls **Split self-intersecting geometries**: INPUT=input.gpkg — Input layer; ERRORS=input.gpkg — Error layer; METHOD=0 selects *Split feature into a multi-part feature*; UNIQUE_ID=field1 — Field of original feature unique identifier; PART_IDX=gc_partidx — Field of part index; RING_IDX=gc_ringidx — Field of ring index; VERTEX_IDX=gc_vertidx — Field of vertex index; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:flattenrelationships — Flatten relationship

group: Vector general

This algorithm flattens a relationship for a vector layer, exporting a single layer containing one master feature per related feature. This master feature contains all the attributes for the related features.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
OUTPUT	sink	yes	—	Flattened layer

Outputs: OUTPUT (outputVector)

Example usage

```
run native:flattenrelationships INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Flatten relationship**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:forceccw` — Force polygons counter-clockwise

group: Vector geometry

This algorithm forces polygon geometries to respect the convention where the exterior ring is oriented in a counter-clockwise direction and the interior rings in a clockwise direction.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Reoriented

Outputs: OUTPUT (outputVector)

Example usage

```
run native:forceccw INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Force polygons counter-clockwise**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:forcecw` — Force polygons clockwise

group: Vector geometry

This algorithm forces polygon geometries to respect the convention where the exterior ring is oriented in a clockwise direction and the interior rings in a counter-clockwise direction.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Reoriented

Outputs: OUTPUT (outputVector)

Example usage

```
run native:forcecw INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Force polygons clockwise**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:forcerhr — Force right-hand-rule ★

group: Vector geometry · **niva verb:** `forcerhr`

This algorithm forces polygon geometries to respect the Right-Hand-Rule, in which the area that is bounded by a polygon is to the right of the boundary. In particular, the exterior ring is oriented in a clockwise direction and the interior rings in a counter-clockwise direction.

Due to the inconsistency in the definition of the Right-Hand-Rule in some contexts, it is recommended to use the explicit ”Force polygons clockwise” processing algorithm instead.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Reoriented

Outputs: OUTPUT (outputVector)

Example usage

```
run native:forcerhr INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Force right-hand-rule**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fuzzifyrastergaussianmembership — Fuzzify raster (gaussian membership)

group: Raster analysis

This algorithm transforms an input raster to a fuzzified raster and thereby assigns values between 0 and 1 following a gaussian fuzzy membership function. The value of 0 implies no membership with the defined fuzzy set, a value of 1 depicts full membership. In between, the degree of membership of raster values follows a gaussian membership function.

The gaussian function is constructed using two user-defined input values which set the midpoint of the gaussian function (midpoint, results to 1) and a predefined function spread which controls the function spread.

This function is typically used when a certain range of raster values around a predefined function midpoint should become members of the fuzzy set.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input Raster
BAND	band	no	1	Band Number
FUZZYMIDPOINT	number	no	10	Function midpoint
FUZZYSPREAD	number	no	0.01	Function spread
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Fuzzified raster

Outputs: EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber), OUTPUT (outputRaster)

Example usage

```
run native:fuzzifyrastergaussianmembership INPUT=dem.tif BAND=1 FUZZYMIDPOINT=10
↪  FUZZYSPREAD=0.01 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Fuzzify raster (gaussian membership)**: `INPUT=dem.tif` — Input Raster; `BAND=1` — Band Number; `FUZZYMIDPOINT=10` — Function midpoint; `FUZZYSPREAD=0.01` — Function spread; `CREATE_OPTIONS=value` — Creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fuzzifyrasterlargemembership — Fuzzify raster (large membership)

group: Raster analysis

This algorithm transforms an input raster to a fuzzified raster and thereby assigns values between 0 and 1 following the 'large' fuzzy membership function. The value of 0 implies no membership with the defined fuzzy set, a value of 1 depicts full membership. In between, the degree of membership of raster values follows the 'large' membership function.

The 'large' function is constructed using two user-defined input raster values which set the point of half membership (midpoint, results to 0.5) and a predefined function spread which controls the function uptake.

This function is typically used when larger input raster values should become members of the fuzzy set more easily than smaller values.

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input Raster
`BAND`	band	no	1	Band Number
`FUZZYMIDPOINT`	number	no	50	Function midpoint
`FUZZYSPREAD`	number	no	5	Function spread
`CREATE_OPTIONS`	string	no	—	Creation options
`CREATION_OPTIONS *(adv)*`	string	no	—	Creation options
`OUTPUT`	rasterDestination	yes	—	Fuzzified raster

Outputs: `EXTENT` (outputString), `CRS_AUTHID` (outputString), `WIDTH_IN_PIXELS` (outputNumber), `HEIGHT_IN_PIXELS` (outputNumber), `TOTAL_PIXEL_COUNT` (outputNumber), `OUTPUT` (outputRaster)

Example usage

```
run native:fuzzifyrasterlargemembership INPUT=dem.tif BAND=1 FUZZYMIDPOINT=50
 ↪  FUZZYSPREAD=5 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Fuzzify raster (large membership)**: `INPUT=dem.tif` — Input Raster; `BAND=1` — Band Number; `FUZZYMIDPOINT=50` — Function midpoint; `FUZZYSPREAD=5` — Function spread; `CREATE_OPTIONS=value` — Creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fuzzifyrasterlinearmembership — Fuzzify raster (linear membership)

group: Raster analysis

This algorithm transforms an input raster to a fuzzified raster and thereby assigns values between 0 and 1 following a linear fuzzy membership function. The value of 0 implies no membership with the defined fuzzy set, a value of 1 depicts full membership. In between, the degree of membership of raster values follows a linear membership function.

The linear function is constructed using two user-defined input raster values which set the point of full membership (high bound, results to 1) and no membership (low bound, results to 0) respectively. The fuzzy set in between those values is defined as a linear function.

Both increasing and decreasing fuzzy sets can be modeled by swapping the high and low bound parameters.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input Raster
BAND	band	no	1	Band Number
FUZZYLOWBOUND	number	no	0	Low fuzzy membership bound
FUZZYHIGHBOUND	number	no	1	High fuzzy membership bound
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Fuzzified raster

Outputs: EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber), OUTPUT (outputRaster)

Example usage

```
run native:fuzzifyrasterlinearmembership INPUT=dem.tif BAND=1 FUZZYLOWBOUND=0
↪  FUZZYHIGHBOUND=1 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Fuzzify raster (linear membership)**: INPUT=dem.tif — Input Raster; BAND=1 — Band Number; FUZZYLOWBOUND=0 — Low fuzzy membership bound; FUZZYHIGHBOUND=1 — High fuzzy membership bound; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:fuzzifyrasternearmembership` — Fuzzify raster (near membership)

group: Raster analysis

The Fuzzify raster (near membership) algorithm transforms an input raster to a fuzzified raster and thereby assigns values between 0 and 1 following the 'near' fuzzy membership function. The value of 0 implies no membership with the defined fuzzy set, a value of 1 depicts full membership. In between, the degree of membership of raster values follows the 'near' membership function.

The 'near' function is constructed using two user-defined input values which set the midpoint of the 'near' function (midpoint, results to 1) and a predefined function spread which controls the function spread.

This function is typically used when a certain range of raster values near a predefined function midpoint should become members of the fuzzy set. The function generally shows a higher rate of decay than the gaussian membership function.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input Raster
BAND	band	no	1	Band Number
FUZZYMIDPOINT	number	no	50	Function midpoint
FUZZYSPREAD	number	no	0.01	Function spread
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Fuzzified raster

Outputs: EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber), OUTPUT (outputRaster)

Example usage

```
run native:fuzzifyrasternearmembership INPUT=dem.tif BAND=1 FUZZYMIDPOINT=50
↪  FUZZYSPREAD=0.01 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Fuzzify raster (near membership)**: `INPUT=dem.tif` — Input Raster; `BAND=1` — Band Number; `FUZZYMIDPOINT=50` — Function midpoint; `FUZZYSPREAD=0.01` — Function spread; `CREATE_OPTIONS=value` — Creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fuzzifyrasterpowermembership — Fuzzify raster (power membership)

group: Raster analysis

This algorithm transforms an input raster to a fuzzified raster and thereby assigns values between 0 and 1 following a power function. The value of 0 implies no membership with the defined fuzzy set, a value of 1 depicts full membership. In between, the degree of membership of raster values follows a power function.

The power function is constructed using three user-defined input raster values which set the point of full membership (high bound, results to 1), no membership (low bound, results to 0) and function exponent (only positive) respectively. The fuzzy set in between those the upper and lower bounds values is then defined as a power function.

Both increasing and decreasing fuzzy sets can be modeled by swapping the high and low bound parameters.

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input Raster
`BAND`	band	no	1	Band Number
`FUZZYLOWBOUND`	number	no	0	Low fuzzy membership bound
`FUZZYHIGHBOUND`	number	no	1	High fuzzy membership bound
`FUZZYEXPONENT`	number	no	2	Membership function exponent
`CREATE_OPTIONS`	string	no	—	Creation options
`CREATION_OPTIONS *(adv)*`	string	no	—	Creation options
`OUTPUT`	rasterDestination	yes	—	Fuzzified raster

Outputs: `EXTENT` (outputString), `CRS_AUTHID` (outputString), `WIDTH_IN_PIXELS` (outputNumber), `HEIGHT_IN_PIXELS` (outputNumber), `TOTAL_PIXEL_COUNT` (outputNumber), `OUTPUT` (outputRaster)

Example usage

```
run native:fuzzifyrasterpowermembership INPUT=dem.tif BAND=1 FUZZYLOWBOUND=0
↪  FUZZYHIGHBOUND=1 FUZZYEXPONENT=2 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Fuzzify raster (power membership)**: `INPUT=dem.tif` — Input Raster; `BAND=1` — Band Number; `FUZZYLOWBOUND=0` — Low fuzzy membership bound; `FUZZYHIGHBOUND=1` — High fuzzy membership bound; `FUZZYEXPONENT=2` — Membership function exponent; `CREATE_OPTIONS=value` — Creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:fuzzifyrastersmallmembership — Fuzzify raster (small membership)

group: Raster analysis

The Fuzzify raster (small membership) algorithm transforms an input raster to a fuzzified raster and thereby assigns values between 0 and 1 following the 'small' fuzzy membership function. The value of 0 implies no membership with the defined fuzzy set, a value of 1 depicts full membership. In between, the degree of membership of raster values follows the 'small' membership function.

The 'small' function is constructed using two user-defined input raster values which set the point of half membership (midpoint, results to 0.5) and a predefined function spread which controls the function uptake.

This function is typically used when smaller input raster values should become members of the fuzzy set more easily than higher values.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input Raster
BAND	band	no	1	Band Number
FUZZYMIDPOINT	number	no	50	Function midpoint
FUZZYSPREAD	number	no	5	Function spread
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Fuzzified raster

Outputs: EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber), OUTPUT (outputRaster)

Example usage

```
run native:fuzzifyrastersmallmembership INPUT=dem.tif BAND=1 FUZZYMIDPOINT=50
↪  FUZZYSPREAD=5 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Fuzzify raster (small membership)**: INPUT=dem.tif — Input Raster; BAND=1 — Band Number; FUZZYMIDPOINT=50 — Function midpoint; FUZZYSPREAD=5 — Function spread; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:generateelevationprofileimage — Generate elevation profile image

group: Plots

This algorithm creates an elevation profile image from a list of map layer and an optional terrain.

Parameter	Type	Required	Default	Description
CURVE	geometry	yes	—	Profile curve
MAP_LAYERS	multilayer	yes	—	Map layers
WIDTH	number	no	400	Chart width (in pixels)
HEIGHT	number	no	300	Chart height (in pixels)
TERRAIN_LAYER	layer	no	—	Terrain layer
MINIMUM_DISTANCE *(adv)*	number	no	—	Chart minimum distance (X axis)
MAXIMUM_DISTANCE *(adv)*	number	no	—	Chart maximum distance (X axis)
MINIMUM_ELEVATION *(adv)*	number	no	—	Chart minimum elevation (Y axis)
MAXIMUM_ELEVATION *(adv)*	number	no	—	Chart maximum elevation (Y axis)
TEXT_COLOR *(adv)*	color	no	<QColor: RGBA 0, 0, 0, 255>	Chart text color
TEXT_FONT_FAMILY	string	no	—	Chart text font family
TEXT_FONT_STYLE	string	no	—	Chart text font style
TEXT_FONT_SIZE	number	no	0	Chart text font size
BACKGROUND_COLOR *(adv)*	color	no	<QColor: RGBA 255, 255, 255, 255>	Chart background color
BORDER_COLOR *(adv)*	color	no	<QColor: RGBA 99, 99, 99, 255>	Chart border color
TOLERANCE *(adv)*	number	no	5.0	Profile tolerance
DPI *(adv)*	number	no	96	Chart DPI

Parameter	Type	Required	Default	Description
OUTPUT	fileDestination	yes	—	Output image

Outputs: OUTPUT (outputFile)

Example usage

```
run native:generateelevationprofileimage CURVE=… MAP_LAYERS="a.gpkg;b.gpkg" WIDTH=400
↪ HEIGHT=300 TERRAIN_LAYER=input.gpkg TEXT_FONT_FAMILY=value TEXT_FONT_STYLE=value
↪ OUTPUT=output.txt
```

This calls **Generate elevation profile image**: CURVE=… — Profile curve; MAP_LAYERS="a.gpkg;b.gpkg" — Map layers; WIDTH=400 — Chart width (in pixels); HEIGHT=300 — Chart height (in pixels); TERRAIN_LAYER=input.gpkg — Terrain layer; TEXT_FONT_FAMILY=value — Chart text font family; TEXT_FONT_STYLE=value — Chart text font style; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:generatepointspixelcentroidsinsidepolygons — Generate points (pixel centroids) inside polygons

group: Vector creation

This algorithm generates pixel centroids for the raster area falling inside polygons. Used to generate points for further raster sampling.

Parameter	Type	Required	Default	Description
INPUT_RASTER	raster	yes	—	Raster layer
INPUT_VECTOR	source	yes	—	Vector layer
OUTPUT	sink	yes	—	Pixel centroids

Outputs: OUTPUT (outputVector)

Example usage

```
run native:generatepointspixelcentroidsinsidepolygons INPUT_RASTER=dem.tif
↪ INPUT_VECTOR=input.gpkg OUTPUT=output.gpkg
```

This calls **Generate points (pixel centroids) inside polygons**: INPUT_RASTER=dem.tif — Raster layer; INPUT_VECTOR=input.gpkg — Vector layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:geometrybyexpression — Geometry by expression

group: Vector geometry

This algorithm updates existing geometries (or creates new geometries) for input features by use of a QGIS expression. This allows complex geometry modifications which can utilize all the flexibility of the QGIS expression engine to manipulate and create geometries for output features.

For help with QGIS expression functions, see the inbuilt help for specific functions which is available in the expression builder.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT_GEOMETRY	enum	no	0	Output geometry type — options: 0=Polygon, 1=Line, 2=Point
WITH_Z	boolean	no	False	Output geometry has z dimension
WITH_M	boolean	no	False	Output geometry has m values

Parameter	Type	Required	Default	Description
EXPRESSION	expression	no	@geometry	Geometry expression
OUTPUT	sink	yes	—	Modified geometry

Outputs: OUTPUT (outputVector)

Example usage

```
run native:geometrybyexpression INPUT=input.gpkg OUTPUT_GEOMETRY=0 WITH_Z=false
↪  WITH_M=false EXPRESSION="value > 0" OUTPUT=output.gpkg
```

This calls **Geometry by expression**: INPUT=input.gpkg — Input layer; OUTPUT_GEOMETRY=0 selects *Polygon*; WITH_Z=false — Output geometry has z dimension; WITH_M=false — Output geometry has m values; EXPRESSION="value > 0" — Geometry expression; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:gltftovector — Convert GLTF to vector features

group: 3D Tiles

This algorithm converts GLTF content to standard vector layer formats.

Parameter	Type	Required	Default	Description
INPUT	file	yes	—	Input GLTF
OUTPUT_POLYGONS	sink	no	—	Output polygons
OUTPUT_LINES	sink	no	—	Output lines

Outputs: OUTPUT_POLYGONS (outputVector), OUTPUT_LINES (outputVector)

Example usage

```
run native:gltftovector INPUT=input.dat OUTPUT_POLYGONS=output_polygons.gpkg
```

This calls **Convert GLTF to vector features**: INPUT=input.dat — Input GLTF; OUTPUT_POLYGONS=output_polygons.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:greaterthanfrequency — Greater than frequency

group: Raster analysis

This algorithm evaluates on a cell-by-cell basis the frequency (number of times) the values of an input stack of rasters are greater than the value of a value raster. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. The input value layer serves as reference layer for the sample layers. Any NoData cells in the value raster or the data layer stack will result in a NoData cell in the output raster if the ignore NoData parameter is not checked. The output NoData value can be set manually. The output rasters extent and resolution is defined by the input raster layer and is always of int32 type.

Parameter	Type	Required	Default	Description
INPUT_VALUE_RASTER	raster	yes	—	Input value raster
INPUT_VALUE_RASTER_BAND	band	no	1	Value raster band
INPUT_RASTERS	multilayer	yes	—	Input raster layers
IGNORE_NODATA	boolean	no	False	Ignore NoData values
OUTPUT_NODATA_VALUE *(adv)*	number	no	-9999	Output NoData value
CREATE_OPTIONS	string	no	—	Creation options

Parameter	Type	Required	Default	Description
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), OCCURRENCE_COUNT (outputNumber), FOUND_LOCATIONS_COUNT (outputNumber), MEAN_FREQUENCY_PER_LOCATION (outputNumber), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber)

Example usage

```
run native:greaterthanfrequency INPUT_VALUE_RASTER=dem.tif INPUT_RASTERS="a.gpkg;b.gpkg"
↪ INPUT_VALUE_RASTER_BAND=1 IGNORE_NODATA=false CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Greater than frequency**: INPUT_VALUE_RASTER=dem.tif — Input value raster; INPUT_RASTERS="a.gpkg;b.gpkg" — Input raster layers; INPUT_VALUE_RASTER_BAND=1 — Value raster band; IGNORE_NODATA=false — Ignore NoData values; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:highestpositioninrasterstack — Highest position in raster stack

group: Raster analysis

This algorithm evaluates on a cell-by-cell basis the position of the raster with the highest value in a stack of rasters. Position counts start with 1 and range to the total number of input rasters. The order of the input rasters is relevant for the algorithm. If multiple rasters feature the highest value, the first raster will be used for the position value. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. Any NoData cells in the raster layer stack will result in a NoData cell in the output raster unless the ”ignore NoData” parameter is checked. The output NoData value can be set manually. The output rasters extent and resolution is defined by a reference raster layer and is always of int32 type.

Parameter	Type	Required	Default	Description
INPUT_RASTERS	multilayer	yes	—	Input raster layers
REFERENCE_LAYER	raster	yes	—	Reference layer
IGNORE_NODATA	boolean	no	False	Ignore NoData values
OUTPUT_NODATA_VALUE *(adv)*	number	no	-9999	Output NoData value
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber)

Example usage

```
run native:highestpositioninrasterstack INPUT_RASTERS="a.gpkg;b.gpkg"
↪ REFERENCE_LAYER=dem.tif IGNORE_NODATA=false CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Highest position in raster stack**: INPUT_RASTERS="a.gpkg;b.gpkg" — Input raster layers; REFERENCE_LAYER=dem.tif — Reference layer; IGNORE_NODATA=false — Ignore NoData values; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:hillshade — Hillshade ★

group: Raster terrain analysis · **niva verb:** `hillshade`

This algorithm calculates the hillshade of the Digital Terrain Model in input.

The shading of the layer is calculated according to the sun position (azimuth and elevation).

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Elevation layer
Z_FACTOR	number	no	1.0	Z factor
AZIMUTH	number	no	300	Azimuth (horizontal angle)
V_ANGLE	number	no	40	Vertical angle
NODATA *(adv)*	number	no	-9999.0	Output NoData value
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Hillshade

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:hillshade INPUT=dem.tif Z_FACTOR=1 AZIMUTH=300 V_ANGLE=40 OUTPUT=output.tif
```

This calls **Hillshade**: INPUT=dem.tif — Elevation layer; Z_FACTOR=1 — Z factor; AZIMUTH=300 — Azimuth (horizontal angle); V_ANGLE=40 — Vertical angle; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:httprequest — HTTP(S) POST/GET request

group: File tools

This algorithm performs a HTTP(S) POST/GET request and returns the HTTP status code and the reply data. If an error occurs then the error code and the message will be returned.

Optionally, the result can be written to a file on disk.

By default the algorithm will warn on errors. Optionally, the algorithm can be set to treat HTTP errors as failures.

Parameter	Type	Required	Default	Description
URL	string	yes	—	URL
METHOD	enum	no	0	Method — options: 0=GET, 1=POST
DATA	string	no	—	POST data
OUTPUT	fileDestination	no	—	File destination
AUTH_CONFIG	authcfg	no	—	Authentication
FAIL_ON_ERROR	boolean	no	False	Consider HTTP errors as failures

Outputs: OUTPUT (outputFile), ERROR_CODE (outputNumber), ERROR_MESSAGE (outputString), STATUS_CODE (outputNumber), RESULT_DATA (outputString)

Example usage

```
run native:httprequest URL=value METHOD=0 DATA=value FAIL_ON_ERROR=false OUTPUT=output.txt
```

This calls **HTTP(S) POST/GET request**: URL=value — URL; METHOD=0 selects *GET*; DATA=value — POST data; FAIL_ON_ERROR=false — Consider HTTP errors as failures; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:hublines — Join by lines (hub lines)

group: Vector analysis

This algorithm creates hub and spoke diagrams by connecting lines from points on the Spoke layer to matching points in the Hub layer.

Determination of which hub goes with each point is based on a match between the Hub ID field on the hub points and the Spoke ID field on the spoke points.

If input layers are not point layers, a point on the surface of the geometries will be taken as the connecting location.

Optionally, geodesic lines can be created, which represent the shortest path on the surface of an ellipsoid. When geodesic mode is used, it is possible to split the created lines at the antimeridian (±180 degrees longitude), which can improve rendering of the lines. Additionally, the distance between vertices can be specified. A smaller distance results in a denser, more accurate line.

Parameter	Type	Required	Default	Description
HUBS	source	yes	—	Hub layer
HUB_FIELD	field	yes	—	Hub ID field
HUB_FIELDS	field	no	—	Hub layer fields to copy (leave empty to copy all fields)
SPOKES	source	yes	—	Spoke layer
SPOKE_FIELD	field	yes	—	Spoke ID field
SPOKE_FIELDS	field	no	—	Spoke layer fields to copy (leave empty to copy all fields)
GEODESIC	boolean	no	False	Create geodesic lines
GEODESIC_DISTANCE *(adv)*	distance	no	1000	Distance between vertices (geodesic lines only)
ANTIMERIDIAN_SPLIT *(adv)*	boolean	no	False	Split lines at antimeridian (±180 degrees longitude)
OUTPUT	sink	yes	—	Hub lines

Outputs: OUTPUT (outputVector)

Example usage

```
run native:hublines HUBS=input.gpkg HUB_FIELD=field1 SPOKES=input.gpkg SPOKE_FIELD=field1
 ↪  HUB_FIELDS=field1 SPOKE_FIELDS=field1 GEODESIC=false OUTPUT=output.gpkg
```

This calls **Join by lines (hub lines)**: HUBS=input.gpkg — Hub layer; HUB_FIELD=field1 — Hub ID field; SPOKES=input.gpkg — Spoke layer; SPOKE_FIELD=field1 — Spoke ID field; HUB_FIELDS=field1 — Hub layer fields to copy (leave empty to copy all fields); SPOKE_FIELDS=field1 — Spoke layer fields to copy (leave empty to copy all fields); GEODESIC=false — Create geodesic lines; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:importintopostgis — Export to PostgreSQL

group: Database

This algorithm exports a vector layer to a PostgreSQL database, creating a new table.

Prior to this a connection between QGIS and the PostgreSQL database has to be created (for example through the QGIS Browser panel).

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Layer to export
DATABASE	providerconnection	yes	—	Database (connection name)
SCHEMA	databaseschema	no	public	Schema (schema name)
TABLENAME	databasetable	no	—	Table to export to (leave blank to use layer name)
PRIMARY_KEY	field	no	—	Primary key field

Parameter	Type	Required	Default	Description
GEOMETRY_COLUMN	string	no	geom	Geometry column
ENCODING	string	no	UTF-8	Encoding
OVERWRITE	boolean	no	True	Overwrite
CREATEINDEX	boolean	no	True	Create spatial index
LOWERCASE_NAMES	boolean	no	True	Convert field names to lowercase
DROP_STRING_-LENGTH	boolean	no	False	Drop length constraints on character fields
FORCE_SINGLEPART	boolean	no	False	Create single-part geometries instead of multipart

Example usage

```
run native:importintopostgis INPUT=input.gpkg DATABASE=… SCHEMA=public PRIMARY_KEY=field1
↪  GEOMETRY_COLUMN=geom ENCODING=UTF-8
```

This calls **Export to PostgreSQL**: INPUT=input.gpkg — Layer to export; DATABASE=… — Database (connection name); SCHEMA=public — Schema (schema name); PRIMARY_KEY=field1 — Primary key field; GEOMETRY_COLUMN=geom — Geometry column; ENCODING=UTF-8 — Encoding. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:importintospatialite — Export to SpatiaLite

group: Database

This algorithm exports a vector layer to a SpatiaLite database, creating a new table.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Layer to export
DATABASE	vector	yes	—	Database layer (or file)
TABLENAME	string	no	—	Table to export to (leave blank to use layer name)
PRIMARY_KEY	field	no	—	Primary key field
GEOMETRY_COLUMN	string	no	geom	Geometry column
ENCODING	string	no	UTF-8	Encoding
OVERWRITE	boolean	no	True	Overwrite
CREATEINDEX	boolean	no	True	Create spatial index
LOWERCASE_NAMES	boolean	no	True	Convert field names to lowercase
DROP_STRING_LENGTH	boolean	no	False	Drop length constraints on character fields
FORCE_SINGLEPART	boolean	no	False	Create single-part geometries instead of multipart

Example usage

```
run native:importintospatialite INPUT=input.gpkg DATABASE=input.gpkg TABLENAME=value
↪  PRIMARY_KEY=field1 GEOMETRY_COLUMN=geom ENCODING=UTF-8 OVERWRITE=true
```

This calls **Export to SpatiaLite**: INPUT=input.gpkg — Layer to export; DATABASE=input.gpkg — Database layer (or file); TABLENAME=value — Table to export to (leave blank to use layer name); PRIMARY_KEY=field1 — Primary key field; GEOMETRY_COLUMN=geom — Geometry column; ENCODING=UTF-8 — Encoding; OVERWRITE=true — Overwrite. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:importintospatialiteregistered — Export to SpatiaLite (registered)

group: Database

Exports a vector layer to a SpatiaLite database, creating a new table.

Prior to this a connection between QGIS and the SpatiaLite database has to be created (for example through the QGIS Browser panel).

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Layer to export
DATABASE	providerconnection	yes	—	Database (connection name)
TABLENAME	databasetable	no	—	Table to export to (leave blank to use layer name)
PRIMARY_KEY	field	no	—	Primary key field
GEOMETRY_COLUMN	string	no	geom	Geometry column
ENCODING	string	no	UTF-8	Encoding
OVERWRITE	boolean	no	True	Overwrite
CREATEINDEX	boolean	no	True	Create spatial index
LOWERCASE_NAMES	boolean	no	True	Convert field names to lowercase
DROP_STRING_-LENGTH	boolean	no	False	Drop length constraints on character fields
FORCE_SINGLEPART	boolean	no	False	Create single-part geometries instead of multipart

Example usage

```
run native:importintospatialiteregistered INPUT=input.gpkg DATABASE=… PRIMARY_KEY=field1
↪  GEOMETRY_COLUMN=geom ENCODING=UTF-8 OVERWRITE=true
```

This calls **Export to SpatiaLite (registered)**: INPUT=input.gpkg — Layer to export; DATABASE=… — Database (connection name); PRIMARY_KEY=field1 — Primary key field; GEOMETRY_COLUMN=geom — Geometry column; ENCODING=UTF-8 — Encoding; OVERWRITE=true — Overwrite. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:importphotos — Import geotagged photos

group: Vector creation

This algorithm creates a point layer corresponding to the geotagged locations from JPEG or HEIF/HEIC images from a source folder. Optionally the folder can be recursively scanned.

The point layer will contain a single PointZ feature per input file from which the geotags could be read. Any altitude information from the geotags will be used to set the point's Z value.

Optionally, a table of unreadable or non-geotagged photos can also be created.

Parameter	Type	Required	Default	Description
FOLDER	file	yes	—	Input folder
RECURSIVE	boolean	no	False	Scan recursively
OUTPUT	sink	no	—	Photos
INVALID	sink	no	—	Invalid photos table

Outputs: OUTPUT (outputVector), INVALID (outputVector)

Example usage

```
run native:importphotos FOLDER=input.dat RECURSIVE=false OUTPUT=output.gpkg
```

This calls **Import geotagged photos**: FOLDER=input.dat — Input folder; RECURSIVE=false — Scan recursively; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:interpolatepoint — Interpolate point on line

group: Vector geometry

This algorithm creates a point geometry interpolated at a set distance along line or curve geometries.

Z and M values are linearly interpolated from existing values.

If a multipart geometry is encountered, only the first part is considered when interpolating the point.

If the specified distance is greater than the curve's length, the resultant feature will have a null geometry.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
DISTANCE	distance	no	0.0	Distance
OUTPUT	sink	yes	—	Interpolated points

Outputs: OUTPUT (outputVector)

Example usage

```
run native:interpolatepoint INPUT=input.gpkg DISTANCE=0 OUTPUT=output.gpkg
```

This calls **Interpolate point on line**: INPUT=input.gpkg — Input layer; DISTANCE=0 — Distance; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:intersection — Intersection ★

group: Vector overlay · **niva verb:** intersect

This algorithm extracts the overlapping portions of features in the Input and Overlay layers. Features in the output Intersection layer are assigned the attributes of the overlapping features from both the Input and Overlay layers.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OVERLAY	source	yes	—	Overlay layer
INPUT_FIELDS	field	no	—	Input fields to keep (leave empty to keep all fields)
OVERLAY_FIELDS	field	no	—	Overlay fields to keep (leave empty to keep all fields)
OVERLAY_FIELDS_PREFIX *(adv)*	string	no	—	Overlay fields prefix
OUTPUT	sink	yes	—	Intersection
GRID_SIZE *(adv)*	number	no	—	Grid size

Outputs: OUTPUT (outputVector)

Example usage

```
run native:intersection INPUT=input.gpkg OVERLAY=input.gpkg INPUT_FIELDS=field1
↪  OVERLAY_FIELDS=field1 OUTPUT=output.gpkg
```

This calls **Intersection**: INPUT=input.gpkg — Input layer; OVERLAY=input.gpkg — Overlay layer; INPUT_FIELDS=field1 — Input fields to keep (leave empty to keep all fields); OVERLAY_FIELDS=field1 — Overlay fields to keep (leave empty to keep all fields); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:joinattributesbylocation — Join attributes by location ★

group: Vector general · **niva verb:** spatialjoin

This algorithm takes an input vector layer and creates a new vector layer that is an extended version of the input one, with additional attributes in its attribute table.

The additional attributes and their values are taken from a second vector layer. A spatial criteria is applied to select the values from the second layer that are added to each feature from the first layer in the resulting one.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Join to features in
PREDICATE	enum	no	0	Features they (geometric predicate) — options: 0=intersect, 1=contain, 2=equal, 3=touch, 4=overlap, 5=are within, 6=cross
JOIN	source	yes	—	By comparing to
JOIN_FIELDS	field	no	—	Fields to add (leave empty to use all fields)
METHOD	enum	no	0	Join type — options: 0=Create separate feature for each matching feature (one-to-many), 1=Take attributes of the first matching feature only (one-to-one), 2=Take attributes of the feature with largest overlap only (one-to-one)
DISCARD_NONMATCHING	boolean	no	False	Discard records which could not be joined
PREFIX	string	no	—	Joined field prefix
OUTPUT	sink	no	—	Joined layer
NON_MATCHING	sink	no	—	Unjoinable features from first layer

Outputs: OUTPUT (outputVector), NON_MATCHING (outputVector), JOINED_COUNT (outputNumber)

Example usage

```
run native:joinattributesbylocation INPUT=input.gpkg JOIN=input.gpkg PREDICATE=0
 ↪  JOIN_FIELDS=field1 METHOD=0 DISCARD_NONMATCHING=false PREFIX=value OUTPUT=output.gpkg
```

This calls **Join attributes by location**: INPUT=input.gpkg — Join to features in; JOIN=input.gpkg — By comparing to; PREDICATE=0 selects *intersect*; JOIN_FIELDS=field1 — Fields to add (leave empty to use all fields); METHOD=0 selects *Create separate feature for each matching feature (one-to-many)*; DISCARD_NONMATCHING=false — Discard records which could not be joined; PREFIX=value — Joined field prefix; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:joinattributestable — Join attributes by field value ★

group: Vector general · **niva verb:** join

This algorithm takes an input vector layer and creates a new vector layer that is an extended version of the input one, with additional attributes in its attribute table.

The additional attributes and their values are taken from a second vector layer. An attribute is selected in each of them to define the join criteria.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	yes	—	Table field
INPUT_2	source	yes	—	Input layer 2
FIELD_2	field	yes	—	Table field 2
FIELDS_TO_COPY	field	no	—	Layer 2 fields to copy (leave empty to copy all fields)
METHOD	enum	no	1	Join type — options: 0=Create separate feature for each matching feature (one-to-many), 1=Take attributes of the first matching feature only (one-to-one)
DISCARD_NONMATCHING	boolean	no	False	Discard records which could not be joined
PREFIX	string	no	—	Joined field prefix
OUTPUT	sink	no	—	Joined layer
NON_MATCHING	sink	no	—	Unjoinable features from first layer

Outputs: OUTPUT (outputVector), NON_MATCHING (outputVector), JOINED_COUNT (outputNumber), UNJOINABLE_COUNT (outputNumber)

Example usage

```
run native:joinattributestable INPUT=input.gpkg FIELD=field1 INPUT_2=input.gpkg
 ↪  FIELD_2=field1 FIELDS_TO_COPY=field1 METHOD=1 DISCARD_NONMATCHING=false
 ↪  OUTPUT=output.gpkg
```

This calls **Join attributes by field value**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — Table field; `INPUT_2=input.gpkg` — Input layer 2; `FIELD_2=field1` — Table field 2; `FIELDS_TO_COPY=field1` — Layer 2 fields to copy (leave empty to copy all fields); `METHOD=1` selects *Take attributes of the first matching feature only (one-to-one)*; `DISCARD_NONMATCHING=false` — Discard records which could not be joined; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:joinbylocationsummary` — Join attributes by location (summary)

group: Vector general

This algorithm takes an input vector layer and creates a new vector layer that is an extended version of the input one, with additional attributes in its attribute table.

The additional attributes and their values are taken from a second vector layer. A spatial criteria is applied to select the values from the second layer that are added to each feature from the first layer in the resulting one.

The algorithm calculates a statistical summary for the values from matching features in the second layer(e.g. maximum value, mean value, etc).

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Join to features in
`PREDICATE`	enum	no	0	Where the features — options: 0=intersect, 1=contain, 2=equal, 3=touch, 4=overlap, 5=are within, 6=cross
`JOIN`	source	yes	—	By comparing to
`JOIN_FIELDS`	field	no	—	Fields to summarise (leave empty to use all fields)
`SUMMARIES`	enum	no	—	Summaries to calculate (leave empty to use all available) — options: 0=count, 1=unique, 2=min, 3=max, 4=range, 5=sum, 6=mean, 7=median, 8=stddev, 9=minority, 10=majority, 11=q1, 12=q3, 13=iqr, 14=empty, 15=filled, 16=min_length, 17=max_length, 18=mean_length
`DISCARD_NONMATCHING`	boolean	no	False	Discard records which could not be joined
`OUTPUT`	sink	yes	—	Joined layer

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:joinbylocationsummary INPUT=input.gpkg JOIN=input.gpkg PREDICATE=0
 ↪  JOIN_FIELDS=field1 SUMMARIES=0 DISCARD_NONMATCHING=false OUTPUT=output.gpkg
```

This calls **Join attributes by location (summary)**: `INPUT=input.gpkg` — Join to features in; `JOIN=input.gpkg` — By comparing to; `PREDICATE=0` selects *intersect*; `JOIN_FIELDS=field1` — Fields to summarise (leave empty to use all fields); `SUMMARIES=0` selects *count*; `DISCARD_NONMATCHING=false` — Discard records which could not be joined; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:joinbynearest` — Join attributes by nearest

group: Vector general

This algorithm takes an input vector layer and creates a new vector layer that is an extended version of the input one, with additional attributes in its attribute table.

The additional attributes and their values are taken from a second vector layer, where features are joined by finding the closest features from each layer. By default only the single nearest feature is joined,but optionally the join can use the n-nearest neighboring features instead. If multiple features are found with identical distances these will all be returned (even if the total number of features exceeds the specified maximum feature count).

If a maximum distance is specified, then only features which are closer than this distance will be matched.

The output features will contain the selected attributes from the nearest feature, along with new attributes for the distance to the near feature, the index of the feature, and the coordinates of the closest point on the input feature (feature_x, feature_y) to the matched nearest feature, and the coordinates of the closet point on the matched feature (nearest_x, nearest_y).

This algorithm uses purely Cartesian calculations for distance, and does not consider geodetic or ellipsoid properties when determining feature proximity.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
INPUT_2	source	yes	—	Input layer 2
FIELDS_TO_COPY	field	no	—	Layer 2 fields to copy (leave empty to copy all fields)
DISCARD_NONMATCHING	boolean	no	False	Discard records which could not be joined
PREFIX	string	no	—	Joined field prefix
NEIGHBORS	number	no	1	Maximum nearest neighbors
MAX_DISTANCE	distance	no	—	Maximum distance
OUTPUT	sink	no	—	Joined layer
NON_MATCHING	sink	no	—	Unjoinable features from first layer

Outputs: OUTPUT (outputVector), NON_MATCHING (outputVector), JOINED_COUNT (outputNumber), UNJOINABLE_COUNT (outputNumber)

Example usage

```
run native:joinbynearest INPUT=input.gpkg INPUT_2=input.gpkg FIELDS_TO_COPY=field1
 ↪  DISCARD_NONMATCHING=false PREFIX=value NEIGHBORS=1 MAX_DISTANCE=10 OUTPUT=output.gpkg
```

This calls **Join attributes by nearest**: INPUT=input.gpkg — Input layer; INPUT_2=input.gpkg — Input layer 2; FIELDS_TO_COPY=field1 — Layer 2 fields to copy (leave empty to copy all fields); DISCARD_NONMATCHING=false — Discard records which could not be joined; PREFIX=value — Joined field prefix; NEIGHBORS=1 — Maximum nearest neighbors; MAX_DISTANCE=10 — Maximum distance; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:keepnbiggestparts — Keep N biggest parts

group: Vector geometry

This algorithm takes a polygon layer and creates a new polygon layer in which multipart geometries have been removed, leaving only the n largest (in terms of area) parts.

Parameter	Type	Required	Default	Description
POLYGONS	source	yes	—	Input layer
PARTS	number	no	1.0	Parts to keep
OUTPUT	sink	yes	—	Parts

Outputs: OUTPUT (outputVector)

Example usage

```
run native:keepnbiggestparts POLYGONS=input.gpkg PARTS=1 OUTPUT=output.gpkg
```

This calls **Keep N biggest parts**: `POLYGONS=input.gpkg` — Input layer; `PARTS=1` — Parts to keep; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:kmeansclustering` — K-means clustering

group: Vector analysis

This algorithm calculates the 2D distance based k-means cluster number for each input feature.

If input geometries are lines or polygons, the clustering is based on the centroid of the feature.

References: Arthur, David & Vassilvitskii, Sergei. (2007). K-Means++: The Advantages of Careful Seeding. Proc. of the Annu. ACM-SIAM Symp. on Discrete Algorithms. 8.

Bhattacharya, Anup & Eube, Jan & Röglin, Heiko & Schmidt, Melanie. (2019). Noisy, Greedy and Not So Greedy k-means++

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`CLUSTERS`	number	no	5	Number of clusters
`METHOD`	enum	no	0	Method — options: 0=Farthest points, 1=K-means++
`FIELD_NAME *(adv)*`	string	no	CLUSTER_ID	Cluster field name
`SIZE_FIELD_NAME *(adv)*`	string	no	CLUSTER_SIZE	Cluster size field name
`OUTPUT`	sink	yes	—	Clusters

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:kmeansclustering INPUT=input.gpkg CLUSTERS=5 METHOD=0 OUTPUT=output.gpkg
```

This calls **K-means clustering**: `INPUT=input.gpkg` — Input layer; `CLUSTERS=5` — Number of clusters; `METHOD=0` selects *Farthest points*; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:layertobookmarks` — Convert layer to spatial bookmarks

group: Vector general

This algorithm creates spatial bookmarks corresponding to the extent of features contained in a layer.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`DESTINATION`	enum	no	0	Bookmark destination — options: 0=Project bookmarks, 1=User bookmarks
`NAME_EXPRESSION`	expression	yes	—	Name field
`GROUP_EXPRESSION`	expression	no	—	Group field

Outputs: `COUNT` (outputNumber)

Example usage

```
run native:layertobookmarks INPUT=input.gpkg NAME_EXPRESSION="value > 0" DESTINATION=0
↪  GROUP_EXPRESSION="value > 0"
```

This calls **Convert layer to spatial bookmarks**: `INPUT=input.gpkg` — Input layer; `NAME_EXPRESSION="value > 0"` — Name field; `DESTINATION=0` selects *Project bookmarks*; `GROUP_EXPRESSION="value > 0"` — Group field. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:lessthanfrequency` — Less than frequency

group: Raster analysis

This algorithm evaluates on a cell-by-cell basis the frequency (number of times) the values of an input stack of rasters are less than the value of a value raster. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. The input value layer serves as reference layer for the sample layers. Any NoData cells in the value raster or the data layer stack will result in a NoData cell in the output raster if the ignore NoData parameter is not checked. The output NoData value can be set manually. The output rasters extent and resolution is defined by the input raster layer and is always of int32 type.

Parameter	Type	Required	Default	Description
`INPUT_VALUE_RASTER`	raster	yes	—	Input value raster
`INPUT_VALUE_RASTER_BAND`	band	no	1	Value raster band
`INPUT_RASTERS`	multilayer	yes	—	Input raster layers
`IGNORE_NODATA`	boolean	no	False	Ignore NoData values
`OUTPUT_NODATA_VALUE *(adv)*`	number	no	-9999	Output NoData value
`CREATE_OPTIONS`	string	no	—	Creation options
`CREATION_OPTIONS *(adv)*`	string	no	—	Creation options
`OUTPUT`	rasterDestination	yes	—	Output layer

Outputs: `OUTPUT` (outputRaster), `OCCURRENCE_COUNT` (outputNumber), `FOUND_LOCATIONS_COUNT` (outputNumber), `MEAN_FREQUENCY_PER_LOCATION` (outputNumber), `EXTENT` (outputString), `CRS_AUTHID` (outputString), `WIDTH_IN_PIXELS` (outputNumber), `HEIGHT_IN_PIXELS` (outputNumber), `TOTAL_PIXEL_COUNT` (outputNumber)

Example usage

```
run native:lessthanfrequency INPUT_VALUE_RASTER=dem.tif INPUT_RASTERS="a.gpkg;b.gpkg"
 ↪  INPUT_VALUE_RASTER_BAND=1 IGNORE_NODATA=false CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Less than frequency**: `INPUT_VALUE_RASTER=dem.tif` — Input value raster; `INPUT_RASTERS="a.gpkg;b.gpkg"` — Input raster layers; `INPUT_VALUE_RASTER_BAND=1` — Value raster band; `IGNORE_NODATA=false` — Ignore NoData values; `CREATE_OPTIONS=value` — Creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:linedensity` — Line density

group: Interpolation

This algorithm calculates a density measure of linear features which is obtained in a circular neighborhood within each raster cell. First, the length of the segment of each line that is intersected by the circular neighborhood is multiplied with the lines weight factor. In a second step, all length values are summed and divided by the area of the circular neighborhood. This process is repeated for all raster cells.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input line layer
`WEIGHT`	field	no	—	Weight field

Parameter	Type	Required	Default	Description
RADIUS	distance	no	10	Search radius
PIXEL_SIZE	distance	no	10	Pixel size
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Line density raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:linedensity INPUT=input.gpkg WEIGHT=field1 RADIUS=10 PIXEL_SIZE=10
↪  CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Line density**: INPUT=input.gpkg — Input line layer; WEIGHT=field1 — Weight field; RADIUS=10 — Search radius; PIXEL_SIZE=10 — Pixel size; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:lineintersections — Line intersections

group: Vector overlay

This algorithm creates point features where the lines in the Intersect layer intersect the lines in the Input layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
INTERSECT	source	yes	—	Intersect layer
INPUT_FIELDS	field	no	—	Input fields to keep (leave empty to keep all fields)
INTERSECT_FIELDS	field	no	—	Intersect fields to keep (leave empty to keep all fields)
INTERSECT_FIELDS_PREFIX *(adv)*	string	no	—	Intersect fields prefix
OUTPUT	sink	yes	—	Intersections

Outputs: OUTPUT (outputVector)

Example usage

```
run native:lineintersections INPUT=input.gpkg INTERSECT=input.gpkg INPUT_FIELDS=field1
↪  INTERSECT_FIELDS=field1 OUTPUT=output.gpkg
```

This calls **Line intersections**: INPUT=input.gpkg — Input layer; INTERSECT=input.gpkg — Intersect layer; INPUT_FIELDS=field1 — Input fields to keep (leave empty to keep all fields); INTERSECT_FIELDS=field1 — Intersect fields to keep (leave empty to keep all fields); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:linesubstring — Line substring

group: Vector geometry

This algorithm returns the portion of a line (or curve) which falls between the specified start and end distances (measured from the beginning of the line).

Z and M values are linearly interpolated from existing values.

If a multipart geometry is encountered, only the first part is considered when calculating the substring.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
START_DISTANCE	distance	no	0.0	Start distance
END_DISTANCE	distance	no	1.0	End distance
OUTPUT	sink	yes	—	Substring

Outputs: OUTPUT (outputVector)

Example usage

```
run native:linesubstring INPUT=input.gpkg START_DISTANCE=0 END_DISTANCE=1
↪  OUTPUT=output.gpkg
```

This calls **Line substring**: INPUT=input.gpkg — Input layer; START_DISTANCE=0 — Start distance; END_DISTANCE=1 — End distance; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:listuniquevalues — List unique values

group: Vector analysis

This algorithm generates a report with information about the unique values found in a given attribute (or attributes) of a vector layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELDS	field	yes	—	Target field(s)
OUTPUT	sink	no	—	Unique values
OUTPUT_HTML_FILE	fileDestination	no	—	HTML report

Outputs: OUTPUT (outputVector), OUTPUT_HTML_FILE (outputHtml), TOTAL_VALUES (outputNumber), UNIQUE_VALUES (outputString)

Example usage

```
run native:listuniquevalues INPUT=input.gpkg FIELDS=field1 OUTPUT=output.gpkg
```

This calls **List unique values**: INPUT=input.gpkg — Input layer; FIELDS=field1 — Target field(s); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:loadlayer — Load layer into project

group: Modeler tools

This algorithm loads a layer to the current project.

Parameter	Type	Required	Default	Description
INPUT	layer	yes	—	Layer
NAME	string	yes	—	Loaded layer name

Outputs: OUTPUT (outputLayer)

Example usage

```
run native:loadlayer INPUT=input.gpkg NAME=value
```

This calls **Load layer into project**: INPUT=input.gpkg — Layer; NAME=value — Loaded layer name. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:lowestpositioninrasterstack — Lowest position in raster stack

group: Raster analysis

This algorithm evaluates on a cell-by-cell basis the position of the raster with the lowest value in a stack of rasters. Position counts start with 1 and range to the total number of input rasters. The order of the input rasters is relevant for the algorithm. If multiple rasters feature the lowest value, the first raster will be used for the position value. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. Any NoData cells in the raster layer stack will result in a NoData cell in the output raster unless the ”ignore NoData” parameter is checked. The output NoData value can be set manually. The output rasters extent and resolution is defined by a reference raster layer and is always of int32 type.

Parameter	Type	Required	Default	Description
INPUT_RASTERS	multilayer	yes	—	Input raster layers
REFERENCE_LAYER	raster	yes	—	Reference layer
IGNORE_NODATA	boolean	no	False	Ignore NoData values
OUTPUT_NODATA_VALUE *(adv)*	number	no	-9999	Output NoData value
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber)

Example usage

```
run native:lowestpositioninrasterstack INPUT_RASTERS="a.gpkg;b.gpkg"
 ↪  REFERENCE_LAYER=dem.tif IGNORE_NODATA=false CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Lowest position in raster stack**: INPUT_RASTERS="a.gpkg;b.gpkg" — Input raster layers; REFERENCE_LAYER=dem.tif — Reference layer; IGNORE_NODATA=false — Ignore NoData values; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:meancoordinates — Mean coordinate(s)

group: Vector analysis

This algorithm computes a point layer with the center of mass of geometries in an input layer.

An attribute can be specified as containing weights to be applied to each feature when computing the center of mass.

If an attribute is selected in the parameter, features will be grouped according to values in this field. Instead of a single point with the center of mass of the whole layer, the output layer will contain a center of mass for the features in each category.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
WEIGHT	field	no	—	Weight field
UID	field	no	—	Unique ID field
OUTPUT	sink	yes	—	Mean coordinates

Outputs: OUTPUT (outputVector)

Example usage

```
run native:meancoordinates INPUT=input.gpkg WEIGHT=field1 UID=field1 OUTPUT=output.gpkg
```

This calls **Mean coordinate(s)**: `INPUT=input.gpkg` — Input layer; `WEIGHT=field1` — Weight field; `UID=field1` — Unique ID field; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:mergelines` — Merge lines

group: Vector geometry

This algorithm joins all connected parts of MultiLineString geometries into single LineString geometries.

If any parts of the input MultiLineString geometries are not connected, the resultant geometry will be a MultiLineString containing any lines which could be merged and any non-connected line parts.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`OUTPUT`	sink	yes	—	Merged

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:mergelines INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Merge lines**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:mergevectorlayers` — Merge vector layers

group: Vector general

This algorithm combines multiple vector layers of the same geometry type into a single one.

The attribute table of the resulting layer will contain the fields from all input layers. If fields with the same name but different types are found then the exported field will be automatically converted into a string type field. Optionally, new fields storing the original layer name and source can be added.

If any input layers contain Z or M values, then the output layer will also contain these values. Similarly, if any of the input layers are multi-part, the output layer will also be a multi-part layer.

Optionally, the destination coordinate reference system (CRS) for the merged layer can be set. If it is not set, the CRS will be taken from the first input layer. All layers will all be reprojected to match this CRS.

Parameter	Type	Required	Default	Description
`LAYERS`	multilayer	yes	—	Input layers
`CRS`	crs	no	—	Destination CRS
`OUTPUT`	sink	yes	—	Merged
`ADD_SOURCE_FIELDS`	boolean	no	True	Add source layer information (layer name and path)

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:mergevectorlayers LAYERS="a.gpkg;b.gpkg" CRS=EPSG:6346 ADD_SOURCE_FIELDS=true
↪ OUTPUT=output.gpkg
```

This calls **Merge vector layers**: LAYERS="a.gpkg;b.gpkg" — Input layers; CRS=EPSG:6346 — Destination CRS; ADD_SOURCE_FIELDS=true — Add source layer information (layer name and path); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:meshcontours — Export contours

group: Mesh

This algorithm creates contours as a vector layer from a mesh scalar dataset.

Parameter	Type	Required	Default	Description
INPUT	mesh	yes	—	Input mesh layer
DATASET_GROUPS	meshdatasetgroups	no	[]	Dataset groups
DATASET_TIME	meshdatasettime	yes	—	Dataset time
INCREMENT	number	no	—	Increment between contour levels
MINIMUM	number	no	—	Minimum contour level
MAXIMUM	number	no	—	Maximum contour level
CONTOUR_LEVEL_LIST	string	no	—	List of contours level
CRS_OUTPUT	crs	no	—	Output coordinate system
OUTPUT_LINES	sink	yes	—	Exported contour lines
OUTPUT_POLYGONS	sink	yes	—	Exported contour polygons

Outputs: OUTPUT_LINES (outputVector), OUTPUT_POLYGONS (outputVector)

Example usage

```
run native:meshcontours INPUT=mesh.nc DATASET_TIME=… INCREMENT=10 MINIMUM=10 MAXIMUM=10
↪  CONTOUR_LEVEL_LIST=value OUTPUT_LINES=output_lines.gpkg
```

This calls **Export contours**: INPUT=mesh.nc — Input mesh layer; DATASET_TIME=… — Dataset time; INCREMENT=10 — Increment between contour levels; MINIMUM=10 — Minimum contour level; MAXIMUM=10 — Maximum contour level; CONTOUR_LEVEL_LIST=value — List of contours level; OUTPUT_LINES=output_lines.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:meshexportcrosssection — Export cross section dataset values on lines from mesh

group: Mesh

This algorithm extracts mesh's dataset values from line contained in a vector layer. Each line is discretized with a resolution distance parameter for extraction of values on its vertices.

Parameter	Type	Required	Default	Description
INPUT	mesh	yes	—	Input mesh layer
DATASET_GROUPS	meshdatasetgroups	no	[]	Dataset groups
DATASET_TIME	meshdatasettime	yes	—	Dataset time
INPUT_LINES	source	yes	—	Lines for data export
RESOLUTION	distance	no	10.0	Line segmentation resolution
COORDINATES_DIGITS	number	no	2	Digits count for coordinates
DATASET_DIGITS	number	no	2	Digits count for dataset value
OUTPUT	fileDestination	yes	—	Exported data CSV file

Outputs: OUTPUT (outputFile)

Example usage

```
run native:meshexportcrosssection INPUT=mesh.nc DATASET_TIME=… INPUT_LINES=input.gpkg
↪  RESOLUTION=10 COORDINATES_DIGITS=2 DATASET_DIGITS=2 OUTPUT=output.txt
```

This calls **Export cross section dataset values on lines from mesh**: `INPUT=mesh.nc` — Input mesh layer; `DATASET_TIME=…` — Dataset time; `INPUT_LINES=input.gpkg` — Lines for data export; `RESOLUTION=10` — Line segmentation resolution; `COORDINATES_DIGITS=2` — Digits count for coordinates; `DATASET_DIGITS=2` — Digits count for dataset value; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:meshexporttimeseries` — Export time series values from points of a mesh dataset

group: Mesh

This algorithm extracts mesh's dataset time series values from points contained in a vector layer. If the time step is kept to its default value (0 hours), the time step used is the one of the two first datasets of the first selected dataset group.

Parameter	Type	Required	Default	Description
`INPUT`	mesh	yes	—	Input mesh layer
`DATASET_GROUPS`	meshdatasetgroups	no	[]	Dataset groups
`STARTING_TIME`	meshdatasettime	yes	—	Starting time
`FINISHING_TIME`	meshdatasettime	yes	—	Finishing time
`TIME_STEP`	number	no	0	Time step (hours)
`INPUT_POINTS`	source	yes	—	Points for data export
`COORDINATES_DIGITS`	number	no	2	Digits count for coordinates
`DATASET_DIGITS`	number	no	2	Digits count for dataset value
`OUTPUT`	fileDestination	yes	—	Exported data CSV file

Outputs: `OUTPUT` (outputFile)

Example usage

```
run native:meshexporttimeseries INPUT=mesh.nc STARTING_TIME=… FINISHING_TIME=…
↪  INPUT_POINTS=input.gpkg TIME_STEP=0 COORDINATES_DIGITS=2 OUTPUT=output.txt
```

This calls **Export time series values from points of a mesh dataset**: `INPUT=mesh.nc` — Input mesh layer; `STARTING_TIME=…` — Starting time; `FINISHING_TIME=…` — Finishing time; `INPUT_POINTS=input.gpkg` — Points for data export; `TIME_STEP=0` — Time step (hours); `COORDINATES_DIGITS=2` — Digits count for coordinates; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:meshrasterize` — Rasterize mesh dataset

group: Mesh

This algorithm creates a raster layer from a mesh dataset. For data on volume (3D stacked dataset values), the exported dataset values are averaged on faces using the method defined in the mesh layer properties (default is Multi level averaging method). 1D meshes are not supported.

Parameter	Type	Required	Default	Description
`INPUT`	mesh	yes	—	Input mesh layer
`DATASET_GROUPS`	meshdatasetgroups	no	[]	Dataset groups
`DATASET_TIME`	meshdatasettime	yes	—	Dataset time
`EXTENT`	extent	no	—	Extent
`PIXEL_SIZE`	distance	no	1	Pixel size
`CRS_OUTPUT`	crs	no	—	Output coordinate system

Parameter	Type	Required	Default	Description
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster layer

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:meshrasterize INPUT=mesh.nc DATASET_TIME=… EXTENT=0,1000,0,1000 PIXEL_SIZE=1
↪  CRS_OUTPUT=EPSG:6346 CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Rasterize mesh dataset**: INPUT=mesh.nc — Input mesh layer; DATASET_TIME=… — Dataset time; EXTENT=0,1000,0,1000 — Extent; PIXEL_SIZE=1 — Pixel size; CRS_OUTPUT=EPSG:6346 — Output coordinate system; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:minimumboundinggeometry — Minimum bounding geometry

group: Vector geometry

This algorithm creates geometries which enclose the features from an input layer.

Numerous enclosing geometry types are supported, including bounding boxes (envelopes), oriented rectangles, circles and convex hulls.

Optionally, the features can be grouped by a field. If set, this causes the output layer to contain one feature per grouped value with a minimal geometry covering just the features with matching values.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	no	—	Field (optional, set if features should be grouped by class)
TYPE	enum	yes	—	Geometry type — options: 0=Envelope (Bounding Box), 1=Minimum Oriented Rectangle, 2=Minimum Enclosing Circle, 3=Convex Hull
OUTPUT	sink	yes	—	Bounding geometry

Outputs: OUTPUT (outputVector)

Example usage

```
run native:minimumboundinggeometry INPUT=input.gpkg TYPE=0 FIELD=field1 OUTPUT=output.gpkg
```

This calls **Minimum bounding geometry**: INPUT=input.gpkg — Input layer; TYPE=0 selects *Envelope (Bounding Box)*; FIELD=field1 — Field (optional, set if features should be grouped by class); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:minimumenclosingcircle — Minimum enclosing circles

group: Vector geometry

This algorithm calculates the minimum enclosing circle which covers each feature in an input layer.

See the 'Minimum bounding geometry' algorithm for a minimal enclosing circle calculation which covers the whole layer or grouped subsets of features.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer

Parameter	Type	Required	Default	Description
SEGMENTS	number	no	72	Number of segments in circles
OUTPUT	sink	yes	—	Minimum enclosing circles

Outputs: OUTPUT (outputVector)

Example usage

```
run native:minimumenclosingcircle INPUT=input.gpkg SEGMENTS=72 OUTPUT=output.gpkg
```

This calls **Minimum enclosing circles**: INPUT=input.gpkg — Input layer; SEGMENTS=72 — Number of segments in circles; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:modelerrastercalc — Raster calculator

group: Raster analysis

This algorithm performs algebraic operations using raster layers.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layers
EXPRESSION	expression	yes	—	Expression
EXTENT	extent	no	—	Output extent
CELL_SIZE	number	no	—	Output cell size (leave empty to set automatically)
CRS	crs	no	—	Output CRS
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Calculated

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:modelerrastercalc LAYERS="a.gpkg;b.gpkg" EXPRESSION="value > 0"
 ↪  EXTENT=0,1000,0,1000 CELL_SIZE=10 CRS=EPSG:6346 OUTPUT=output.tif
```

This calls **Raster calculator**: LAYERS="a.gpkg;b.gpkg" — Input layers; EXPRESSION="value > 0" — Expression; EXTENT=0,1000,0,1000 — Output extent; CELL_SIZE=10 — Output cell size (leave empty to set automatically); CRS=EPSG:6346 — Output CRS; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:modelervirtualrastercalc — Raster calculator (virtual)

group: Raster analysis

This algorithm performs algebraic operations using raster layers and generates in-memory result.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layers
EXPRESSION	expression	yes	—	Expression
EXTENT	extent	no	—	Output extent
CELL_SIZE	number	no	—	Output cell size (leave empty to set automatically)
CRS	crs	no	—	Output CRS
LAYER_NAME	string	no	—	Output layer name

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:modelervirtualrastercalc LAYERS="a.gpkg;b.gpkg" EXPRESSION="value > 0"
 ↪  EXTENT=0,1000,0,1000 CELL_SIZE=10 CRS=EPSG:6346 LAYER_NAME=value
```

This calls **Raster calculator (virtual)**: LAYERS="a.gpkg;b.gpkg" — Input layers; EXPRESSION="value > 0" — Expression; EXTENT=0,1000,0,1000 — Output extent; CELL_SIZE=10 — Output cell size (leave empty to set automatically); CRS=EPSG:6346 — Output CRS; LAYER_NAME=value — Output layer name. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:multidifference — Difference (multiple)

group: Vector overlay

This algorithm extracts features from the Input layer that fall completely outside or only partially overlap the features from any of the Overlay layer(s). For each overlay layer the difference is calculated between the result of all previous difference operations and this overlay layer. Input layer features that partially overlap feature(s) in the Overlay layers are split along those features' boundary and only the portions outside the Overlay layer features are retained.

Attributes are not modified, although properties such as area or length of the features will be modified by the difference operation. If such properties are stored as attributes, those attributes will have to be manually updated.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OVERLAYS	multilayer	yes	—	Overlay layers
OUTPUT	sink	yes	—	Difference

Outputs: OUTPUT (outputVector)

Example usage

```
run native:multidifference INPUT=input.gpkg OVERLAYS="a.gpkg;b.gpkg" OUTPUT=output.gpkg
```

This calls **Difference (multiple)**: INPUT=input.gpkg — Input layer; OVERLAYS="a.gpkg;b.gpkg" — Overlay layers; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:multiintersection — Intersection (multiple)

group: Vector overlay

This algorithm extracts the overlapping portions of features in the Input and all Overlay layers. Features in the output layer are assigned the attributes of the overlapping features from both the Input and Overlay layers.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OVERLAYS	multilayer	yes	—	Overlay layers
OVERLAY_FIELDS_PREFIX *(adv)*	string	no	—	Overlay fields prefix
OUTPUT	sink	yes	—	Intersection

Outputs: OUTPUT (outputVector)

Example usage

```
run native:multiintersection INPUT=input.gpkg OVERLAYS="a.gpkg;b.gpkg" OUTPUT=output.gpkg
```

This calls **Intersection (multiple)**: INPUT=input.gpkg — Input layer; OVERLAYS="a.gpkg;b.gpkg" — Overlay layers; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:multiparttosingleparts` — Multipart to singleparts ★

group: Vector geometry · **niva verb:** `explode`

This algorithm takes a vector layer with multipart geometries and generates a new one in which all geometries contain a single part. Features with multipart geometries are divided in as many different features as parts the geometry contain, and the same attributes are used for each of them.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`OUTPUT`	sink	yes	—	Single parts

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:multiparttosingleparts INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Multipart to singleparts**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:multiringconstantbuffer` — Multi-ring buffer (constant distance)

group: Vector geometry

This algorithm computes multi-ring ('donuts') buffer for the features in an input layer, using a fixed or dynamic distance and number of rings.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`RINGS`	number	no	1	Number of rings
`DISTANCE`	distance	no	1	Distance between rings
`OUTPUT`	sink	yes	—	Multi-ring buffer (constant distance)

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:multiringconstantbuffer INPUT=input.gpkg RINGS=1 DISTANCE=1 OUTPUT=output.gpkg
```

This calls **Multi-ring buffer (constant distance)**: `INPUT=input.gpkg` — Input layer; `RINGS=1` — Number of rings; `DISTANCE=1` — Distance between rings; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:multiunion` — Union (multiple)

group: Vector overlay

This algorithm checks overlaps between features within the Input layer and creates separate features for overlapping and non-overlapping parts. The area of overlap will create as many identical overlapping features as there are features that participate in that overlap.

Multiple Overlay layers can also be used, in which case features from each layer are split at their overlap with features from all other layers, creating a layer containing all the portions from both Input and Overlay layers. The attribute table of the Union layer is filled with attribute values from the respective original layer for non-overlapping features, and attribute values from both layers for overlapping features.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OVERLAYS	multilayer	no	—	Overlay layers
OVERLAY_FIELDS_PREFIX *(adv)*	string	no	—	Overlay fields prefix
OUTPUT	sink	yes	—	Union

Outputs: OUTPUT (outputVector)

Example usage

```
run native:multiunion INPUT=input.gpkg OVERLAYS="a.gpkg;b.gpkg" OUTPUT=output.gpkg
```

This calls **Union (multiple)**: INPUT=input.gpkg — Input layer; OVERLAYS="a.gpkg;b.gpkg" — Overlay layers; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:nearestneighbouranalysis — Nearest neighbour analysis

group: Vector analysis

This algorithm performs nearest neighbor analysis for a point layer.

The output describes how the data are distributed (clustered, randomly or distributed).

Output is generated as an HTML file with the computed statistical values.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT_HTML_FILE	fileDestination	no	—	Nearest neighbour

Outputs: OUTPUT_HTML_FILE (outputHtml), OBSERVED_MD (outputNumber), EXPECTED_MD (outputNumber), NN_INDEX (outputNumber), POINT_COUNT (outputNumber), Z_SCORE (outputNumber)

Example usage

```
run native:nearestneighbouranalysis INPUT=input.gpkg OUTPUT_HTML_FILE=output_html_file.txt
```

This calls **Nearest neighbour analysis**: INPUT=input.gpkg — Input layer; OUTPUT_HTML_FILE=output_html_file.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:offsetline — Offset lines ★

group: Vector geometry · **niva verb:** offset

This algorithm offsets lines by a specified distance. Positive distances will offset lines to the left, and negative distances will offset to the right of lines.

The segments parameter controls the number of line segments to use to approximate a quarter circle when creating rounded offsets.

The join style parameter specifies whether round, miter or beveled joins should be used when offsetting corners in a line.

The miter limit parameter is only applicable for miter join styles, and controls the maximum distance from the offset curve to use when creating a mitered join.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
DISTANCE	distance	no	10.0	Distance
SEGMENTS	number	no	8	Segments

Parameter	Type	Required	Default	Description
JOIN_STYLE	enum	no	0	Join style — options: 0=Round, 1=Miter, 2=Bevel
MITER_LIMIT	number	no	2	Miter limit
OUTPUT	sink	yes	—	Offset

Outputs: OUTPUT (outputVector)

Example usage

```
run native:offsetline INPUT=input.gpkg DISTANCE=10 SEGMENTS=8 JOIN_STYLE=0 MITER_LIMIT=2
↪ OUTPUT=output.gpkg
```

This calls **Offset lines**: INPUT=input.gpkg — Input layer; DISTANCE=10 — Distance; SEGMENTS=8 — Segments; JOIN_STYLE=0 selects *Round*; MITER_LIMIT=2 — Miter limit; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:openurl — Open file or URL

group: File tools

This algorithm opens files in their default associated application, or URLs in the user's default web browser.

Parameter	Type	Required	Default	Description
URL	string	yes	—	URL or file path

Outputs: SUCCESS (outputBoolean)

Example usage

```
run native:openurl URL=value
```

This calls **Open file or URL**: URL=value — URL or file path. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:orderbyexpression — Order by expression

group: Vector general

This algorithm sorts a vector layer according to an expression. Be careful, it might not work as expected with some providers, the order might not be kept every time.

For help with QGIS expression functions, see the inbuilt help for specific functions which is available in the expression builder.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
EXPRESSION	expression	yes	—	Expression
ASCENDING	boolean	no	True	Sort ascending
NULLS_FIRST	boolean	no	False	Sort nulls first
OUTPUT	sink	yes	—	Ordered

Outputs: OUTPUT (outputVector)

Example usage

```
run native:orderbyexpression INPUT=input.gpkg EXPRESSION="value > 0" ASCENDING=true
↪ NULLS_FIRST=false OUTPUT=output.gpkg
```

This calls **Order by expression**: `INPUT=input.gpkg` — Input layer; `EXPRESSION="value > 0"` — Expression; `ASCENDING=true` — Sort ascending; `NULLS_FIRST=false` — Sort nulls first; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:orientedminimumboundingbox` — Oriented minimum bounding box ★

group: Vector geometry · **niva verb:** `minrect`

This algorithm calculates the minimum area rotated rectangle which covers each feature in an input layer.

For singlepart point features, the output corresponds to the bounding box of the geometry.

See the 'Minimum bounding geometry' algorithm for an oriented bounding box calculation which covers the whole layer or grouped subsets of features.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`OUTPUT`	sink	yes	—	Bounding boxes

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:orientedminimumboundingbox INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Oriented minimum bounding box**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:orthogonalize` — Orthogonalize

group: Vector geometry

Takes a line or polygon layer and attempts to orthogonalize all the geometries in the layer. This process shifts the nodes in the geometries to try to make every angle in the geometry either a right angle or a straight line.

The angle tolerance parameter is used to specify the maximum deviation from a right angle or straight line a node can have for it to be adjusted. Smaller tolerances mean that only nodes which are already closer to right angles will be adjusted, and larger tolerances mean that nodes which deviate further from right angles will also be adjusted.

The algorithm is iterative. Setting a larger number for the maximum iterations will result in a more orthogonal geometry at the cost of extra processing time.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`ANGLE_TOLERANCE`	number	no	15.0	Maximum angle tolerance (degrees)
`MAX_ITERATIONS *(adv)*`	number	no	1000	Maximum algorithm iterations
`OUTPUT`	sink	yes	—	Orthogonalized

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:orthogonalize INPUT=input.gpkg ANGLE_TOLERANCE=15 OUTPUT=output.gpkg
```

This calls **Orthogonalize**: INPUT=input.gpkg — Input layer; ANGLE_TOLERANCE=15 — Maximum angle tolerance (degrees); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:package — Package layers

group: Database

This algorithm collects a number of existing layers and packages them together into a single GeoPackage database.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layers
OUTPUT	fileDestination	yes	—	Destination GeoPackage
OVERWRITE	boolean	no	False	Overwrite existing GeoPackage
SAVE_STYLES	boolean	no	True	Save layer styles into GeoPackage
SAVE_METADATA	boolean	no	True	Save layer metadata into GeoPackage
SELECTED_FEATURES_-ONLY	boolean	no	False	Save only selected features
EXPORT_RELATED_LAYERS	boolean	no	False	Export related layers following relations defined in the project
EXTENT	extent	no	—	Extent
CRS *(adv)*	crs	no	—	Destination CRS

Outputs: OUTPUT (outputFile), OUTPUT_LAYERS (outputMultilayer)

Example usage

```
run native:package LAYERS="a.gpkg;b.gpkg" OVERWRITE=false SAVE_STYLES=true
↪ SAVE_METADATA=true SELECTED_FEATURES_ONLY=false EXPORT_RELATED_LAYERS=false
↪ EXTENT=0,1000,0,1000 OUTPUT=output.txt
```

This calls **Package layers**: LAYERS="a.gpkg;b.gpkg" — Input layers; OVERWRITE=false — Overwrite existing GeoPackage; SAVE_STYLES=true — Save layer styles into GeoPackage; SAVE_METADATA=true — Save layer metadata into GeoPackage; SELECTED_FEATURES_ONLY=false — Save only selected features; EXPORT_RELATED_LAYERS=false — Export related layers following relations defined in the project; EXTENT=0,1000,0,1000 — Extent; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:pixelstopoints — Raster pixels to points

group: Vector creation

This algorithm converts a raster layer to a vector layer, by creating point features for each individual pixel's center in the raster layer.

Any NoData pixels are skipped in the output.

Parameter	Type	Required	Default	Description
INPUT_RASTER	raster	yes	—	Raster layer
RASTER_BAND	band	no	1	Band number
FIELD_NAME	string	no	VALUE	Field name
OUTPUT	sink	yes	—	Vector points

Outputs: OUTPUT (outputVector)

Example usage

```
run native:pixelstopoints INPUT_RASTER=dem.tif RASTER_BAND=1 FIELD_NAME=VALUE
↪ OUTPUT=output.gpkg
```

This calls **Raster pixels to points**: `INPUT_RASTER=dem.tif` — Raster layer; `RASTER_BAND=1` — Band number; `FIELD_NAME=VALUE` — Field name; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:pixelstopolygons` — Raster pixels to polygons

group: Vector creation

This algorithm converts a raster layer to a vector layer, by creating polygon features for each individual pixel's extent in the raster layer.

Any NoData pixels are skipped in the output.

Parameter	Type	Required	Default	Description
`INPUT_RASTER`	raster	yes	—	Raster layer
`RASTER_BAND`	band	no	1	Band number
`FIELD_NAME`	string	no	VALUE	Field name
`OUTPUT`	sink	yes	—	Vector polygons

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:pixelstopolygons INPUT_RASTER=dem.tif RASTER_BAND=1 FIELD_NAME=VALUE
↪ OUTPUT=output.gpkg
```

This calls **Raster pixels to polygons**: `INPUT_RASTER=dem.tif` — Raster layer; `RASTER_BAND=1` — Band number; `FIELD_NAME=VALUE` — Field name; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:pointonsurface` — Point on surface ★

group: Vector geometry · **niva verb:** `pointonsurface`

This algorithm returns a point guaranteed to lie on the surface of a geometry.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`ALL_PARTS`	boolean	no	False	Create point on surface for each part
`OUTPUT`	sink	yes	—	Point

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:pointonsurface INPUT=input.gpkg ALL_PARTS=false OUTPUT=output.gpkg
```

This calls **Point on surface**: `INPUT=input.gpkg` — Input layer; `ALL_PARTS=false` — Create point on surface for each part; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:pointsalonglines` — Points along geometry ★

group: Vector geometry · **niva verb:** `pointsalong`

This algorithm creates a points layer, with points distributed along the lines of an input vector layer. The distance between points (measured along the line) is defined as a parameter.

Start and end offset distances can be defined, so the first and last point will not fall exactly on the line's first and last nodes. These start and end offsets are defined as distances, measured along the line from the first and last nodes of the lines.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`DISTANCE`	distance	no	1.0	Distance
`START_OFFSET`	distance	no	0.0	Start offset
`END_OFFSET`	distance	no	0.0	End offset
`OUTPUT`	sink	yes	—	Interpolated points

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:pointsalonglines INPUT=input.gpkg DISTANCE=1 START_OFFSET=0 END_OFFSET=0
↪  OUTPUT=output.gpkg
```

This calls **Points along geometry**: `INPUT=input.gpkg` — Input layer; `DISTANCE=1` — Distance; `START_OFFSET=0` — Start offset; `END_OFFSET=0` — End offset; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:pointstopath` — Points to path

group: Vector creation

This algorithm takes a point layer and connects its features to create a new line layer.

An attribute or expression may be specified to define the order the points should be connected. If no order expression is specified, the feature ID is used.

A natural sort can be used when sorting by a string attribute or expression (ie. place 'a9' before 'a10').

An attribute or expression can be selected to group points having the same value into the same resulting line.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`CLOSE_PATH`	boolean	no	False	Create closed paths
`ORDER_EXPRESSION`	expression	no	—	Order expression
`NATURAL_SORT`	boolean	no	False	Sort text containing numbers naturally
`GROUP_EXPRESSION`	expression	no	—	Path group expression
`OUTPUT`	sink	yes	—	Paths
`OUTPUT_TEXT_DIR`	folderDestination	no	—	Directory for text output
`ORDER_FIELD`	field	no	—	Order field
`GROUP_FIELD`	field	no	—	Group field
`DATE_FORMAT`	string	no	—	Date format (if order field is DateTime)

Outputs: `OUTPUT` (outputVector), `OUTPUT_TEXT_DIR` (outputFolder), `NUM_PATHS` (outputNumber)

Example usage

```
run native:pointstopath INPUT=input.gpkg CLOSE_PATH=false ORDER_EXPRESSION="value > 0"
 ↪  NATURAL_SORT=false GROUP_EXPRESSION="value > 0" ORDER_FIELD=field1 GROUP_FIELD=field1
 ↪  OUTPUT=output.gpkg
```

This calls **Points to path**: `INPUT=input.gpkg` — Input layer; `CLOSE_PATH=false` — Create closed paths; `ORDER_EXPRESSION="value > 0"` — Order expression; `NATURAL_SORT=false` — Sort text containing numbers naturally; `GROUP_EXPRESSION="value > 0"` — Path group expression; `ORDER_FIELD=field1` — Order field; `GROUP_FIELD=field1` — Group field; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:pointtolayer` — Create layer from point

group: Vector creation

This algorithm creates a new vector layer that contains a single feature with geometry matching a point parameter.

It can be used in models to convert a point into a layer which can be used for other algorithms which require a layer based input.

Parameter	Type	Required	Default	Description
`INPUT`	point	yes	—	Point
`OUTPUT`	sink	yes	—	Point

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:pointtolayer INPUT=100,100 OUTPUT=output.gpkg
```

This calls **Create layer from point**: `INPUT=100,100` — Point; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:poleofinaccessibility` — Pole of inaccessibility

group: Vector geometry

This algorithm calculates the pole of inaccessibility for a polygon layer, which is the most distant internal point from the boundary of the surface. This algorithm uses the 'polylabel' algorithm (Vladimir Agafonkin, 2016), which is an iterative approach guaranteed to find the true pole of inaccessibility within a specified tolerance (in layer units). More precise tolerances require more iterations and will take longer to calculate.

The distance from the calculated pole to the polygon boundary will be stored as a new attribute in the output layer.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`TOLERANCE`	distance	no	1.0	Tolerance
`OUTPUT`	sink	yes	—	Point

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:poleofinaccessibility INPUT=input.gpkg TOLERANCE=1 OUTPUT=output.gpkg
```

This calls **Pole of inaccessibility**: `INPUT=input.gpkg` — Input layer; `TOLERANCE=1` — Tolerance; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:polygonfromlayerextent` — Extract layer extent

group: Layer tools

This algorithm takes a map layer and generates a new vector layer with the minimum bounding box (rectangle polygon with N-S orientation) that covers the input layer. Optionally, the extent can be enlarged to a rounded value.

Parameter	Type	Required	Default	Description
`INPUT`	layer	yes	—	Input layer
`ROUND_TO *(adv)*`	distance	no	0	Round values to
`OUTPUT`	sink	yes	—	Extent

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:polygonfromlayerextent INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Extract layer extent**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:polygonize` — Polygonize

group: Vector geometry

This algorithm creates a polygon layer from the input lines layer.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`KEEP_FIELDS`	boolean	no	False	Keep table structure of line layer
`OUTPUT`	sink	yes	—	Polygons

Outputs: `OUTPUT` (outputVector), `NUM_POLYGONS` (outputNumber)

Example usage

```
run native:polygonize INPUT=input.gpkg KEEP_FIELDS=false OUTPUT=output.gpkg
```

This calls **Polygonize**: `INPUT=input.gpkg` — Input layer; `KEEP_FIELDS=false` — Keep table structure of line layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:polygonstolines` — Polygons to lines

group: Vector geometry

This algorithm converts polygons to lines.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`OUTPUT`	sink	yes	—	Lines

Outputs: OUTPUT (outputVector)

Example usage

```
run native:polygonstolines INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Polygons to lines**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:postgisexecuteandloadsql` — PostgreSQL execute and load SQL

group: Database

This algorithm performs a SQL database query on a PostgreSQL database connected to QGIS and loads the query results as a new layer.

Parameter	Type	Required	Default	Description
`DATABASE`	providerconnection	yes	—	Database (connection name)
`SQL`	string	yes	—	SQL query
`ID_FIELD`	string	no	id	Unique ID field name
`GEOMETRY_FIELD`	string	no	geom	Geometry field name

Outputs: OUTPUT (outputVector)

Example usage

```
run native:postgisexecuteandloadsql DATABASE=… SQL=value ID_FIELD=id GEOMETRY_FIELD=geom
```

This calls **PostgreSQL execute and load SQL**: `DATABASE=…` — Database (connection name); `SQL=value` — SQL query; `ID_FIELD=id` — Unique ID field name; `GEOMETRY_FIELD=geom` — Geometry field name. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:postgisexecutesql` — PostgreSQL execute SQL

group: Database

This algorithm executes a SQL command on a PostgreSQL database.

Parameter	Type	Required	Default	Description
`DATABASE`	providerconnection	yes	—	Database (connection name)
`SQL`	string	yes	—	SQL query

Example usage

```
run native:postgisexecutesql DATABASE=… SQL=value
```

This calls **PostgreSQL execute SQL**: `DATABASE=…` — Database (connection name); `SQL=value` — SQL query. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:printlayoutmapextenttolayer — Print layout map extent to layer

group: Cartography

This algorithm creates a polygon layer containing the extent of a print layout map item (or items), with attributes specifying the map size (in layout units), scale and rotation.

If the map item parameter is specified, then only the matching map extent will be exported. If it is not specified, all map extents from the layout will be exported.

Optionally, a specific output CRS can be specified. If it is not specified, the original map item CRS will be used.

Parameter	Type	Required	Default	Description
LAYOUT	layout	yes	—	Print layout
MAP	layoutitem	no	—	Map item
CRS *(adv)*	crs	no	—	Override CRS
OUTPUT	sink	yes	—	Extent

Outputs: OUTPUT (outputVector), WIDTH (outputNumber), HEIGHT (outputNumber), SCALE (outputNumber), ROTATION (outputNumber)

Example usage

```
run native:printlayoutmapextenttolayer LAYOUT=… OUTPUT=output.gpkg
```

This calls **Print layout map extent to layer**: LAYOUT=**…** — Print layout; OUTPUT=`output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:printlayouttoimage — Export print layout as image

group: Cartography

This algorithm outputs a print layout as an image file (e.g. PNG or JPEG images).

Parameter	Type	Required	Default	Description
LAYOUT	layout	yes	—	Print layout
LAYERS *(adv)*	multilayer	no	—	Map layers to assign to unlocked map item(s)
DPI *(adv)*	number	no	—	DPI (leave blank for default layout DPI)
GEOREFERENCE *(adv)*	boolean	no	True	Generate world file
INCLUDE_METADATA *(adv)*	boolean	no	True	Export RDF metadata (title, author, etc.)
ANTIALIAS *(adv)*	boolean	no	True	Enable antialiasing
OUTPUT	fileDestination	yes	—	Image file

Outputs: OUTPUT (outputFile)

Example usage

```
run native:printlayouttoimage LAYOUT=… OUTPUT=output.txt
```

This calls **Export print layout as image**: LAYOUT=**…** — Print layout; OUTPUT=`output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:printlayouttopdf — Export print layout as PDF

group: Cartography

This algorithm outputs a print layout as a PDF file.

Parameter	Type	Required	Default	Description
LAYOUT	layout	yes	—	Print layout
LAYERS *(adv)*	multilayer	no	—	Map layers to assign to unlocked map item(s)
DPI *(adv)*	number	no	—	DPI (leave blank for default layout DPI)
FORCE_VECTOR *(adv)*	boolean	no	False	Always export as vectors
FORCE_RASTER *(adv)*	boolean	no	False	Always export as raster
GEOREFERENCE *(adv)*	boolean	no	True	Append georeference information
INCLUDE_METADATA *(adv)*	boolean	no	True	Export RDF metadata (title, author, etc.)
DISABLE_TILED *(adv)*	boolean	no	False	Disable tiled raster layer exports
SIMPLIFY *(adv)*	boolean	no	True	Simplify geometries to reduce output file size
TEXT_FORMAT *(adv)*	enum	no	0	Text export — options: 0=Always Export Text as Paths (Recommended), 1=Always Export Text as Text Objects, 2=Prefer Exporting Text as Text Objects
IMAGE_COMPRESSION *(adv)*	enum	no	0	Image compression — options: 0=Lossy (JPEG), 1=Lossless
SEPARATE_LAYERS *(adv)*	boolean	no	False	Export layers as separate PDF files
OUTPUT	fileDestination	yes	—	PDF file

Outputs: OUTPUT (outputFile)

Example usage

```
run native:printlayouttopdf LAYOUT=… OUTPUT=output.txt
```

This calls **Export print layout as PDF**: LAYOUT=… — Print layout; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:projectpointcartesian — Project points (Cartesian)

group: Vector geometry

This algorithm projects point geometries by a specified distance and bearing (azimuth), creating a new point layer with the projected points.

The distance is specified in layer units, and the bearing in degrees clockwise from North.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
BEARING	number	no	0	Bearing (degrees from North)
DISTANCE	distance	no	1	Distance
OUTPUT	sink	yes	—	Projected

Outputs: OUTPUT (outputVector)

Example usage

```
run native:projectpointcartesian INPUT=input.gpkg BEARING=0 DISTANCE=1 OUTPUT=output.gpkg
```

This calls **Project points (Cartesian)**: INPUT=input.gpkg — Input layer; BEARING=0 — Bearing (degrees from North); DISTANCE=1 — Distance; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:promotetomulti — Promote to multipart ★

group: Vector geometry · **niva verb:** promote

This algorithm takes a vector layer with singlepart geometries and generates a new one in which all geometries are multipart. Input features which are already multipart features will remain unchanged.

This algorithm can be used to force geometries to multipart types in order to be compatible with data providers with strict singlepart/multipart compatibility checks.

See the 'Collect geometries' or 'Aggregate' algorithms for alternative options.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Multiparts

Outputs: OUTPUT (outputVector)

Example usage

```
run native:promotetomulti INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Promote to multipart**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:raiseexception — Raise exception

group: Modeler tools

This algorithm raises an exception and cancels a model's execution.

The exception message can be customized, and optionally an expression based condition can be specified. If an expression condition is used, then the exception will only be raised if the expression result is true. A false result indicates that no exception will be raised, and the model execution can continue uninterrupted.

Parameter	Type	Required	Default	Description
MESSAGE	string	yes	—	Error message
CONDITION	expression	no	—	Condition

Example usage

```
run native:raiseexception MESSAGE=value CONDITION="value > 0"
```

This calls **Raise exception**: MESSAGE=value — Error message; CONDITION="value > 0" — Condition. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:raisemessage — Raise message

group: Modeler tools

This algorithm raises an information message in the log.

The message can be customized, and optionally an expression based condition can be specified. If an expression condition is used, then the message will only be logged if the expression result is true. A false result indicates that no message will be logged.

Parameter	Type	Required	Default	Description
MESSAGE	string	yes	—	Information message
CONDITION	expression	no	—	Condition

Example usage

```
run native:raisemessage MESSAGE=value CONDITION="value > 0"
```

This calls **Raise message**: `MESSAGE=value` — Information message; `CONDITION="value > 0"` — Condition. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:raisewarning` — Raise warning

group: Modeler tools

This algorithm raises a warning message in the log.

The warning message can be customized, and optionally an expression based condition can be specified. If an expression condition is used, then the warning will only be logged if the expression result is true. A false result indicates that no warning will be logged.

Parameter	Type	Required	Default	Description
`MESSAGE`	string	yes	—	Warning message
`CONDITION`	expression	no	—	Condition

Example usage

```
run native:raisewarning MESSAGE=value CONDITION="value > 0"
```

This calls **Raise warning**: `MESSAGE=value` — Warning message; `CONDITION="value > 0"` — Condition. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:randomextract` — Random extract ★

group: Vector selection · **niva verb:** `sample`

This algorithm takes a vector layer and generates a new one that contains only a subset of the features in the input layer.

The subset is defined randomly, using a percentage or count value to define the total number of features in the subset.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`METHOD`	enum	no	0	Method — options: 0=Number of features, 1=Percentage of features
`NUMBER`	number	no	10	Number/percentage of features
`OUTPUT`	sink	yes	—	Extracted (random)

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:randomextract INPUT=input.gpkg METHOD=0 NUMBER=10 OUTPUT=output.gpkg
```

This calls **Random extract**: `INPUT=input.gpkg` — Input layer; `METHOD=0` selects *Number of features*; `NUMBER=10` — Number/percentage of features; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:randompointsinextent — Random points in extent

group: Vector creation

This algorithm creates a new point layer with a given number of random points, all of them within a given extent. A distance factor can be specified, to avoid points being too close to each other. If the minimum distance between points makes it impossible to create new points, either distance can be decreased or the maximum number of attempts may be increased.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Input extent
POINTS_NUMBER	number	no	1	Number of points
MIN_DISTANCE	distance	no	0	Minimum distance between points
TARGET_CRS	crs	no	ProjectCrs	Target CRS
MAX_ATTEMPTS *(adv)*	number	no	200	Maximum number of search attempts given the minimum distance
OUTPUT	sink	yes	—	Random points

Outputs: OUTPUT (outputVector)

Example usage

```
run native:randompointsinextent EXTENT=0,1000,0,1000 POINTS_NUMBER=1 MIN_DISTANCE=0
↪  TARGET_CRS=EPSG:6346 OUTPUT=output.gpkg
```

This calls **Random points in extent**: EXTENT=0,1000,0,1000 — Input extent; POINTS_NUMBER=1 — Number of points; MIN_DISTANCE=0 — Minimum distance between points; TARGET_CRS=EPSG:6346 — Target CRS; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:randompointsinpolygons — Random points in polygons

group: Vector creation

This algorithm creates a point layer, with points placed randomly in the polygons of the Input polygon layer. For each feature in the Input polygon layer, the algorithm attempts to add the specified Number of points for each feature to the output layer. A Minimum distance between points and a Global minimum distance between points can be specified. A point will not be added if there is an already generated point within this (Euclidean) distance from the generated location. With Minimum distance between points, only points in the same polygon feature are considered, while for Global minimum distance between points all previously generated points are considered. If the Global minimum distance between points is set equal to or larger than the (local) Minimum distance between points, the latter has no effect. If the Minimum distance between points is too large, it may not be possible to generate the specified Number of points for each feature, but all the generated points are returned. The Maximum number of attempts per point can be specified. The seed for the random generator can be provided (Random seed - integer, greater than 0). The user can choose not to Include polygon feature attributes in the attributes of the generated point features. The total number of points will be 'number of input features' * Number of points for each feature if there are no misses. The Number of points for each feature, Minimum distance between points and Maximum number of attempts per point can be data defined. Output from the algorithm: The number of features with an empty or no geometry (FEATURES_WITH_EMPTY_OR_NO_GEOMETRY). A point layer containing the random points (OUTPUT). The number of generated features (OUTPUT_POINTS). The number of missed points (POINTS_MISSED). The number of features with non-empty geometry and missing points (POLYGONS_WITH_MISSED_POINTS).

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input polygon layer
POINTS_NUMBER	number	no	1	Number of points for each feature

Parameter	Type	Required	Default	Description
MIN_DISTANCE	distance	no	0	Minimum distance between points
MIN_DISTANCE_GLOBAL *(adv)*	distance	no	0	Global minimum distance between points
MAX_TRIES_PER_POINT *(adv)*	number	no	10	Maximum number of search attempts (for Min. dist. > 0)
SEED *(adv)*	number	no	—	Random seed
INCLUDE_POLYGON_ATTRIBUTES *(adv)*	boolean	no	True	Include polygon attributes
OUTPUT	sink	yes	—	Random points in polygons

Outputs: OUTPUT (outputVector), OUTPUT_POINTS (outputNumber), POINTS_MISSED (outputNumber), POLYGONS_WITH_MISSED_POINTS (outputNumber), FEATURES_WITH_EMPTY_OR_NO_GEOMETRY (outputNumber)

Example usage

```
run native:randompointsinpolygons INPUT=input.gpkg POINTS_NUMBER=1 MIN_DISTANCE=0
 ↪  OUTPUT=output.gpkg
```

This calls **Random points in polygons**: INPUT=input.gpkg — Input polygon layer; POINTS_NUMBER=1 — Number of points for each feature; MIN_DISTANCE=0 — Minimum distance between points; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:randompointsonlines — Random points on lines

group: Vector creation

This algorithm creates a point layer with points placed randomly on the lines of the Input line layer. The default behavior is that the generated point features inherit the attributes of the line feature on which they were generated.Parameters / options: For each feature in the Input line layer, the algorithm attempts to add the specified Number of points for each feature to the output layer. A Minimum distance between points and a Global minimum distance between points can be specified. A point will not be added if there is an already generated point within this (Euclidean) distance from the generated location. With Minimum distance between points, only points on the same line feature are considered, while for Global minimum distance between points all previously generated points are considered. If the Global minimum distance between points is set larger than the (local) Minimum distance between points, the latter has no effect. If the Minimum distance between points is too large, it may not be possible to generate the specified Number of points for each feature. The Maximum number of attempts per point is only relevant if Minimum distance between points or Global minimum distance between points is greater than 0. The total number of points will be number of input features * Number of points for each feature if there are no misses and all features have proper geometries. The seed for the random generator can be provided (Random seed - integer, greater than 0). The user can choose not to Include line feature attributes in the generated point features. Output from the algorithm: A point layer containing the random points (OUTPUT). The number of generated features (POINTS_GENERATED). The number of missed points (POINTS_MISSED). The number of features with non-empty geometry and missing points (LINES_WITH_MISSED_POINTS). The number of features with an empty or no geometry (LINES_WITH_EMPTY_OR_NO_GEOMETRY).

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input line layer
POINTS_NUMBER	number	no	1	Number of points for each feature
MIN_DISTANCE	distance	no	0	Minimum distance between points
MIN_DISTANCE_GLOBAL *(adv)*	distance	no	0	Global minimum distance between points
MAX_TRIES_PER_POINT *(adv)*	number	no	10	Maximum number of search attempts (for Min. dist. > 0)
SEED *(adv)*	number	no	—	Random seed

Parameter	Type	Required	Default	Description
INCLUDE_LINE_ATTRIBUTES *(adv)*	boolean	no	True	Include line attributes
OUTPUT	sink	yes	—	Random points on lines

Outputs: OUTPUT (outputVector), OUTPUT_POINTS (outputNumber), POINTS_MISSED (outputNumber), LINES_WITH_MISSED_POINTS (outputNumber), FEATURES_WITH_EMPTY_OR_NO_GEOMETRY (outputNumber)

Example usage

```
run native:randompointsonlines INPUT=input.gpkg POINTS_NUMBER=1 MIN_DISTANCE=0
↪ OUTPUT=output.gpkg
```

This calls **Random points on lines**: INPUT=input.gpkg — Input line layer; POINTS_NUMBER=1 — Number of points for each feature; MIN_DISTANCE=0 — Minimum distance between points; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:randomselection — Random selection

group: Vector selection

This algorithm takes a vector layer and selects a subset of its features. No new layer is generated by this algorithm.

The subset is defined randomly, using a percentage or count value to define the total number of features in the subset.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
METHOD	enum	no	0	Method — options: 0=Number of features, 1=Percentage of features
NUMBER	number	no	10	Number/percentage of features

Outputs: OUTPUT (outputVector)

Example usage

```
run native:randomselection INPUT=input.gpkg METHOD=0 NUMBER=10
```

This calls **Random selection**: INPUT=input.gpkg — Input layer; METHOD=0 selects *Number of features*; NUMBER=10 — Number/percentage of features. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rasterbooleanand — Raster boolean AND

group: Raster analysis

This algorithm calculates the boolean AND for a set of input rasters. If all of the input rasters have a non-zero value for a pixel, that pixel will be set to 1 in the output raster. If any of the input rasters have 0 values for the pixel it will be set to 0 in the output raster.

The reference layer parameter specifies an existing raster layer to use as a reference when creating the output raster. The output raster will have the same extent, CRS, and pixel dimensions as this layer.

By default, a NoData pixel in ANY of the input layers will result in a NoData pixel in the output raster. If the 'Treat NoData values as false' option is checked, then NoData inputs will be treated the same as a 0 input value.

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input layers
REF_LAYER	raster	yes	—	Reference layer
NODATA_AS_FALSE	boolean	no	False	Treat NoData values as false
NO_DATA *(adv)*	number	no	-9999	Output NoData value
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=Int32, 4=UInt32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber), NODATA_PIXEL_COUNT (outputNumber), TRUE_PIXEL_COUNT (outputNumber), FALSE_PIXEL_COUNT (outputNumber)

Example usage

```
run native:rasterbooleanand INPUT="a.gpkg;b.gpkg" REF_LAYER=dem.tif NODATA_AS_FALSE=false
↪  CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Raster boolean AND**: INPUT="a.gpkg;b.gpkg" — Input layers; REF_LAYER=dem.tif — Reference layer; NODATA_AS_FALSE=false — Treat NoData values as false; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rastercalc — Raster calculator

group: Raster analysis

This algorithm performs algebraic operations using raster layers.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layers
EXPRESSION	expression	yes	—	Expression
EXTENT	extent	no	—	Output extent
CELL_SIZE	number	no	—	Output cell size (leave empty to set automatically)
CRS	crs	no	—	Output CRS
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Calculated

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:rastercalc LAYERS="a.gpkg;b.gpkg" EXPRESSION="value > 0" EXTENT=0,1000,0,1000
↪  CELL_SIZE=10 CRS=EPSG:6346 OUTPUT=output.tif
```

This calls **Raster calculator**: LAYERS="a.gpkg;b.gpkg" — Input layers; EXPRESSION="value > 0" — Expression; EXTENT=0,1000,0,1000 — Output extent; CELL_SIZE=10 — Output cell size (leave empty to set automatically); CRS=EPSG:6346 — Output CRS; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rasterfeaturepreservingsmoothing — Feature preserving DEM smoothing

group: Raster analysis

This algorithm applies the Feature-Preserving DEM Smoothing (FPDEMS) method, as described by Lindsay et al. (2019).

It is effective at removing surface roughness from Digital Elevation Models (DEMs) without significantly altering sharp features such as breaks-in-slope, stream banks, or terrace scarps. This makes it superior to standard low-pass filters (e.g., mean, median, Gaussian) or resampling, which often blur distinct topographic features.

The algorithm works in three steps:

1. Calculating surface normal 3D vectors for each grid cell.
2. Smoothing the normal vector field using a filter that applies more weight to neighbors with similar surface normals (preserving edges).
3. Iteratively updating the elevations in the DEM to match the smoothed normal field.

References: Lindsay, John et al (2019): LiDAR DEM Smoothing and the Preservation of Drainage Features. Remote Sensing, Vol. 11, Issue 16, https://doi.org/10.3390/rs11161926

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
RADIUS	number	no	5	Filter radius (pixels)
THRESHOLD	number	no	15.0	Normal difference threshold (degrees)
ITERATIONS	number	no	3	Elevation update iterations
MAX_ELEVATION_CHANGE	number	no	—	Maximum elevation change
Z_FACTOR *(adv)*	number	no	1.0	Z factor
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:rasterfeaturepreservingsmoothing INPUT=dem.tif BAND=1 RADIUS=5 THRESHOLD=15
 ↪  ITERATIONS=3 MAX_ELEVATION_CHANGE=10 OUTPUT=output.tif
```

This calls **Feature preserving DEM smoothing**: INPUT=dem.tif — Input layer; BAND=1 — Band number; RADIUS=5 — Filter radius (pixels); THRESHOLD=15 — Normal difference threshold (degrees); ITERATIONS=3 — Elevation update iterations; MAX_ELEVATION_CHANGE=10 — Maximum elevation change; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rastergaussianblur — Gaussian blur

group: Raster analysis

This algorithm applies a Gaussian blur filter to an input raster layer.

The radius parameter controls the strength of the blur. A larger radius results in a smoother output, at the cost of execution time.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
RADIUS	number	no	2.0	Blur radius (pixels)
CREATION_OPTIONS *(adv)*	string	no	—	Creation options

Parameter	Type	Required	Default	Description
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:rastergaussianblur INPUT=dem.tif BAND=1 RADIUS=2 OUTPUT=output.tif
```

This calls **Gaussian blur**: `INPUT=dem.tif` — Input layer; `BAND=1` — Band number; `RADIUS=2` — Blur radius (pixels); `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rasterize — Convert map to raster

group: Raster tools

This algorithm rasterizes map canvas content.

A map theme can be selected to render a predetermined set of layers with a defined style for each layer. Alternatively, a set of layers can be selected if no map theme is set. If neither map theme nor layer is set, all the visible layers in the set extent will be rendered.

The minimum extent entered will internally be extended to a multiple of the tile size.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Minimum extent to render
EXTENT_BUFFER	number	no	0	Buffer around tiles in map units
TILE_SIZE	number	no	1024	Tile size
MAP_UNITS_PER_PIXEL	number	no	100	Map units per pixel
MAKE_BACKGROUND_TRANSPARENT	boolean	no	False	Make background transparent
MAP_THEME	maptheme	no	—	Map theme to render
LAYERS	multilayer	no	—	Layers to render
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:rasterize EXTENT=0,1000,0,1000 EXTENT_BUFFER=0 TILE_SIZE=1024
 ↪ MAP_UNITS_PER_PIXEL=100 MAKE_BACKGROUND_TRANSPARENT=false LAYERS="a.gpkg;b.gpkg"
 ↪ OUTPUT=output.tif
```

This calls **Convert map to raster**: `EXTENT=0,1000,0,1000` — Minimum extent to render; `EXTENT_BUFFER=0` — Buffer around tiles in map units; `TILE_SIZE=1024` — Tile size; `MAP_UNITS_PER_PIXEL=100` — Map units per pixel; `MAKE_BACKGROUND_TRANSPARENT=false` — Make background transparent; `LAYERS="a.gpkg;b.gpkg"` — Layers to render; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rasterlayerproperties — Raster layer properties

group: Raster analysis

This algorithm returns basic properties of the given raster layer, including the extent, size in pixels and dimensions of pixels (in map units).

If an optional band number is specified then the NoData value for the selected band will also be returned.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	—	Band number

Outputs: X_MIN (outputNumber), X_MAX (outputNumber), Y_MIN (outputNumber), Y_MAX (outputNumber), EXTENT (outputString), PIXEL_WIDTH (outputNumber), PIXEL_HEIGHT (outputNumber), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), HAS_NODATA_VALUE (outputBoolean), NODATA_VALUE (outputNumber), BAND_COUNT (outputNumber)

Example usage

```
run native:rasterlayerproperties INPUT=dem.tif BAND=1
```

This calls **Raster layer properties**: INPUT=dem.tif — Input layer; BAND=1 — Band number. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rasterlayerstatistics — Raster layer statistics

group: Raster analysis

This algorithm computes basic statistics from the values in a given band of the raster layer.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
OUTPUT_HTML_FILE	fileDestination	no	—	Statistics

Outputs: OUTPUT_HTML_FILE (outputHtml), COUNT (outputNumber), MIN (outputNumber), MAX (outputNumber), RANGE (outputNumber), SUM (outputNumber), MEAN (outputNumber), STD_DEV (outputNumber), SUM_OF_SQUARES (outputNumber)

Example usage

```
run native:rasterlayerstatistics INPUT=dem.tif BAND=1
↪ OUTPUT_HTML_FILE=output_html_file.txt
```

This calls **Raster layer statistics**: INPUT=dem.tif — Input layer; BAND=1 — Band number; OUTPUT_HTML_FILE=output_html_file.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rasterlayeruniquevaluesreport — Raster layer unique values report

group: Raster analysis

This algorithm returns the count and area of each unique value in a given raster layer. The area calculation is done in the area unit of the layer's CRS.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
OUTPUT_HTML_FILE	fileDestination	no	—	Unique values report
OUTPUT_TABLE	sink	no	—	Unique values table

Outputs: OUTPUT_HTML_FILE (outputHtml), OUTPUT_TABLE (outputVector), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber), NODATA_PIXEL_COUNT (outputNumber)

Example usage

```
run native:rasterlayeruniquevaluesreport INPUT=dem.tif BAND=1
↪ OUTPUT_HTML_FILE=output_html_file.txt
```

This calls **Raster layer unique values report**: `INPUT=dem.tif` — Input layer; `BAND=1` — Band number; `OUTPUT_HTML_FILE=output_html_file.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:rasterlayerzonalstats` — Raster layer zonal statistics

group: Raster analysis

This algorithm calculates statistics for a raster layer's values, categorized by zones defined in another raster layer.

If the reference layer parameter is set to ”Input layer”, then zones are determined by sampling the zone raster layer value at the centroid of each pixel from the source raster layer.

If the reference layer parameter is set to ”Zones layer”, then the input raster layer will be sampled at the centroid of each pixel from the zones raster layer.

If either the source raster layer or the zone raster layer value is NoData for a pixel, that pixel's value will be skipped and not included in the calculated statistics.

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input layer
`BAND`	band	no	1	Band number
`ZONES`	raster	yes	—	Zones layer
`ZONES_BAND`	band	no	1	Zones band number
`REF_LAYER *(adv)*`	enum	no	0	Reference layer — options: 0=Input layer, 1=Zones layer
`OUTPUT_TABLE`	sink	yes	—	Statistics

Outputs: `OUTPUT_TABLE` (outputVector), `EXTENT` (outputString), `CRS_AUTHID` (outputString), `WIDTH_IN_PIXELS` (outputNumber), `HEIGHT_IN_PIXELS` (outputNumber), `TOTAL_PIXEL_COUNT` (outputNumber), `NODATA_PIXEL_COUNT` (outputNumber)

Example usage

```
run native:rasterlayerzonalstats INPUT=dem.tif ZONES=dem.tif BAND=1 ZONES_BAND=1
↪ OUTPUT_TABLE=output_table.gpkg
```

This calls **Raster layer zonal statistics**: `INPUT=dem.tif` — Input layer; `ZONES=dem.tif` — Zones layer; `BAND=1` — Band number; `ZONES_BAND=1` — Zones band number; `OUTPUT_TABLE=output_table.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:rasterlogicalor` — Raster boolean OR

group: Raster analysis

This algorithm calculates the boolean OR for a set of input rasters. If any of the input rasters have a non-zero value for a pixel, that pixel will be set to 1 in the output raster. If all the input rasters have 0 values for the pixel it will be set to 0 in the output raster.

The reference layer parameter specifies an existing raster layer to use as a reference when creating the output raster. The output raster will have the same extent, CRS, and pixel dimensions as this layer.

By default, a NoData pixel in ANY of the input layers will result in a NoData pixel in the output raster. If the 'Treat NoData values as false' option is checked, then NoData inputs will be treated the same as a 0 input value.

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input layers
REF_LAYER	raster	yes	—	Reference layer
NODATA_AS_FALSE	boolean	no	False	Treat NoData values as false
NO_DATA *(adv)*	number	no	-9999	Output NoData value
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=Int32, 4=UInt32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output layer

Outputs: OUTPUT (outputRaster), EXTENT (outputString), CRS_AUTHID (outputString), WIDTH_IN_PIXELS (outputNumber), HEIGHT_IN_PIXELS (outputNumber), TOTAL_PIXEL_COUNT (outputNumber), NODATA_PIXEL_COUNT (outputNumber), TRUE_PIXEL_COUNT (outputNumber), FALSE_PIXEL_COUNT (outputNumber)

Example usage

```
run native:rasterlogicalor INPUT="a.gpkg;b.gpkg" REF_LAYER=dem.tif NODATA_AS_FALSE=false
↪  CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Raster boolean OR**: INPUT="a.gpkg;b.gpkg" — Input layers; REF_LAYER=dem.tif — Reference layer; NODATA_AS_FALSE=false — Treat NoData values as false; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rasterminmax — Raster minimum/maximum

group: Raster analysis

This algorithm extracts extrema (minimum and maximum) values from a given band of the raster layer.

The output is a vector layer containing point features for the selected extrema, at the center of the associated pixel.

If multiple pixels in the raster share the minimum or maximum value, then only one of these pixels will be included in the output.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
EXTRACT	enum	no	0	Extract extrema — options: 0=Minimum and Maximum, 1=Minimum, 2=Maximum
OUTPUT	sink	no	—	Output

Outputs: OUTPUT (outputVector), MINIMUM (outputNumber), MAXIMUM (outputNumber)

Example usage

```
run native:rasterminmax INPUT=dem.tif BAND=1 EXTRACT=0 OUTPUT=output.gpkg
```

This calls **Raster minimum/maximum**: INPUT=dem.tif — Input layer; BAND=1 — Band number; EXTRACT=0 selects *Minimum and Maximum*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rasterrank — Raster rank

group: Raster analysis

This algorithm performs a cell-by-cell analysis in which output values match the rank of a sorted list of overlapping cell values from input layers. The output raster will be multi-band if multiple ranks are provided. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters.

Parameter	Type	Required	Default	Description
INPUT_RASTERS	multilayer	yes	—	Input raster layers
RANKS	string	no	1	Rank(s) (separate multiple ranks using commas)
NODATA_HANDLING	enum	no	0	NoData value handling — options: 0=Exclude NoData from values lists, 1=Presence of NoData in a values list results in NoData output cell
EXTENT *(adv)*	extent	no	—	Output extent
CELL_SIZE *(adv)*	number	no	—	Output cell size
CRS *(adv)*	crs	no	—	Output CRS
OUTPUT	rasterDestination	yes	—	Ranked

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:rasterrank INPUT_RASTERS="a.gpkg;b.gpkg" RANKS=value NODATA_HANDLING=0
 ↪  OUTPUT=output.tif
```

This calls **Raster rank**: INPUT_RASTERS="a.gpkg;b.gpkg" — Input raster layers; RANKS=value — Rank(s) (separate multiple ranks using commas); NODATA_HANDLING=0 selects *Exclude NoData from values lists*; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rastersampling — Sample raster values

group: Raster analysis

This algorithm creates a new vector layer with the same attributes of the input layer and the raster values corresponding on the point location.

If the raster layer has more than one band, all the band values are sampled.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
RASTERCOPY	raster	yes	—	Raster layer
COLUMN_PREFIX	string	no	SAMPLE_	Output column prefix
OUTPUT	sink	yes	—	Sampled

Outputs: OUTPUT (outputVector)

Example usage

```
run native:rastersampling INPUT=input.gpkg RASTERCOPY=dem.tif COLUMN_PREFIX=SAMPLE_
 ↪  OUTPUT=output.gpkg
```

This calls **Sample raster values**: INPUT=input.gpkg — Input layer; RASTERCOPY=dem.tif — Raster layer; COLUMN_PREFIX=SAMPLE_ — Output column prefix; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rastersurfacevolume — Raster surface volume

group: Raster analysis

This algorithm calculates the volume under a raster grid's surface.

Several methods of volume calculation are available, which control whether only values above or below the specified base level are considered, or whether volumes below the base level should be added or subtracted from the total volume.

The algorithm outputs the calculated volume, the total area, and the total number of pixels analysed. If the 'Count Only Above Base Level' or 'Count Only Below Base Level' methods are used, then the calculated area and pixel count only includes pixels which are above or below the specified base level respectively.

Units of the calculated volume are dependent on the coordinate reference system of the input raster file. For a CRS in meters, with a DEM height in meters, the calculated value will be in meters³. If instead the input raster is in a geographic coordinate system (e.g. latitude/longitude values), then the result will be in degrees² × meters, and an appropriate scaling factor will need to be applied in order to convert to meters³.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
LEVEL	number	no	0	Base level
METHOD	enum	yes	—	Method — options: 0=Count Only Above Base Level, 1=Count Only Below Base Level, 2=Subtract Volumes Below Base Level, 3=Add Volumes Below Base Level
OUTPUT_HTML_FILE	fileDestination	no	—	Surface volume report
OUTPUT_TABLE	sink	no	—	Surface volume table

Outputs: OUTPUT_HTML_FILE (outputHtml), OUTPUT_TABLE (outputVector), VOLUME (outputNumber), PIXEL_COUNT (outputNumber), AREA (outputNumber)

Example usage

```
run native:rastersurfacevolume INPUT=dem.tif METHOD=0 BAND=1 LEVEL=0
↪  OUTPUT_HTML_FILE=output_html_file.txt
```

This calls **Raster surface volume**: INPUT=dem.tif — Input layer; METHOD=0 selects *Count Only Above Base Level*; BAND=1 — Band number; LEVEL=0 — Base level; OUTPUT_HTML_FILE=output_html_file.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:reclassifybylayer — Reclassify by layer

group: Raster analysis

This algorithm reclassifies a raster band by assigning new class values based on the ranges specified in a vector table.

Parameter	Type	Required	Default	Description
INPUT_RASTER	raster	yes	—	Raster layer
RASTER_BAND	band	no	1	Band number
INPUT_TABLE	source	yes	—	Layer containing class breaks
MIN_FIELD	field	yes	—	Minimum class value field
MAX_FIELD	field	yes	—	Maximum class value field
VALUE_FIELD	field	yes	—	Output value field
NO_DATA *(adv)*	number	no	-9999	Output NoData value

Parameter	Type	Required	Default	Description
RANGE_BOUNDARIES *(adv)*	enum	no	0	Range boundaries — options: 0=min < value <= max, 1=min <= value < max, 2=min <= value <= max, 3=min < value < max
NODATA_FOR_MISSING *(adv)*	boolean	no	False	Use NoData when no range matches value
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=Int32, 4=UInt32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Reclassified raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:reclassifybylayer INPUT_RASTER=dem.tif INPUT_TABLE=input.gpkg MIN_FIELD=field1
↪ MAX_FIELD=field1 VALUE_FIELD=field1 RASTER_BAND=1 CREATE_OPTIONS=value
↪ OUTPUT=output.tif
```

This calls **Reclassify by layer**: INPUT_RASTER=dem.tif — Raster layer; INPUT_TABLE=input.gpkg — Layer containing class breaks; MIN_FIELD=field1 — Minimum class value field; MAX_FIELD=field1 — Maximum class value field; VALUE_FIELD=field1 — Output value field; RASTER_BAND=1 — Band number; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:reclassifybytable — Reclassify by table

group: Raster analysis

This algorithm reclassifies a raster band by assigning new class values based on the ranges specified in a fixed table.

Parameter	Type	Required	Default	Description
INPUT_RASTER	raster	yes	—	Raster layer
RASTER_BAND	band	no	1	Band number
TABLE	matrix	yes	—	Reclassification table
NO_DATA *(adv)*	number	no	-9999	Output NoData value
RANGE_BOUNDARIES *(adv)*	enum	no	0	Range boundaries — options: 0=min < value <= max, 1=min <= value < max, 2=min <= value <= max, 3=min < value < max
NODATA_FOR_MISSING *(adv)*	boolean	no	False	Use NoData when no range matches value
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=Int32, 4=UInt32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Reclassified raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:reclassifybytable INPUT_RASTER=dem.tif TABLE=… RASTER_BAND=1
↪ CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Reclassify by table**: `INPUT_RASTER=dem.tif` — Raster layer; `TABLE=…` — Reclassification table; `RASTER_BAND=1` — Band number; `CREATE_OPTIONS=value` — Creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:rectanglesovalsdiamonds` — Rectangles, ovals, diamonds

group: Vector geometry

This algorithm creates rectangle, oval or diamond-shaped polygons from the input point layer using specified width, height and (optional) rotation values. Multipart inputs should be promoted to singleparts first.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`SHAPE`	enum	no	0	Shape — options: 0=Rectangle, 1=Diamond, 2=Oval
`WIDTH`	distance	no	1.0	Width
`HEIGHT`	distance	no	1.0	Height
`ROTATION`	number	no	0.0	Rotation
`SEGMENTS`	number	no	36	Segments
`OUTPUT`	sink	yes	—	Polygon

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:rectanglesovalsdiamonds INPUT=input.gpkg SHAPE=0 WIDTH=1 HEIGHT=1 ROTATION=0
↪  SEGMENTS=36 OUTPUT=output.gpkg
```

This calls **Rectangles, ovals, diamonds**: `INPUT=input.gpkg` — Input layer; `SHAPE=0` selects *Rectangle*; `WIDTH=1` — Width; `HEIGHT=1` — Height; `ROTATION=0` — Rotation; `SEGMENTS=36` — Segments; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:refactorfields` — Refactor fields

group: Vector table

This algorithm allows editing the structure of the attributes table of a vector layer. Fields can be modified in their type and name, using a fields mapping.

The original layer is not modified. A new layer is generated, which contains a modified attribute table, according to the provided fields mapping.

Rows in orange have constraints in the template layer from which these fields were loaded. Treat this information as a hint during configuration. No constraints will be added on an output layer nor will they be checked or enforced by the algorithm.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`FIELDS_MAPPING`	fields_mapping	yes	—	Fields mapping
`OUTPUT`	sink	yes	—	Refactored

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:refactorfields INPUT=input.gpkg FIELDS_MAPPING=… OUTPUT=output.gpkg
```

This calls **Refactor fields**: `INPUT=input.gpkg` — Input layer; `FIELDS_MAPPING=…` — Fields mapping; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:removeduplicatesbyattribute` — Delete duplicates by attribute

group: Vector general

This algorithm removes duplicate rows by a field value (or multiple field values). The first matching row will be retained, and duplicates will be discarded.

Optionally, these duplicate records can be saved to a separate output for analysis.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`FIELDS`	field	yes	—	Field to match duplicates by
`OUTPUT`	sink	yes	—	Filtered (no duplicates)
`DUPLICATES`	sink	no	—	Filtered (duplicates)

Outputs: `OUTPUT` (outputVector), `DUPLICATES` (outputVector), `RETAINED_COUNT` (outputNumber), `DUPLICATE_COUNT` (outputNumber)

Example usage

```
run native:removeduplicatesbyattribute INPUT=input.gpkg FIELDS=field1 OUTPUT=output.gpkg
```

This calls **Delete duplicates by attribute**: `INPUT=input.gpkg` — Input layer; `FIELDS=field1` — Field to match duplicates by; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:removeduplicatevertices` — Remove duplicate vertices

group: Vector geometry

This algorithm removes duplicate vertices from features, wherever removing the vertices does not result in a degenerate geometry.

The tolerance parameter specifies the tolerance for coordinates when determining whether vertices are identical.

By default, z values are not considered when detecting duplicate vertices. E.g. two vertices with the same x and y coordinate but different z values will still be considered duplicate and one will be removed. If the Use Z Value parameter is true, then the z values are also tested and vertices with the same x and y but different z will be maintained.

Note that duplicate vertices are not tested between different parts of a multipart geometry. E.g. a multipoint geometry with overlapping points will not be changed by this method.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`TOLERANCE`	distance	no	1e-06	Tolerance
`USE_Z_VALUE`	boolean	no	False	Use Z Value
`OUTPUT`	sink	yes	—	Cleaned

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:removeduplicatevertices INPUT=input.gpkg TOLERANCE=1e-06 USE_Z_VALUE=false
↪ OUTPUT=output.gpkg
```

This calls **Remove duplicate vertices**: `INPUT=input.gpkg` — Input layer; `TOLERANCE=1e-06` — Tolerance; `USE_Z_VALUE=false` — Use Z Value; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:removenullgeometries` — Remove null geometries

group: Vector geometry

This algorithm removes any features which do not have a geometry from a vector layer. All other features will be copied unchanged.

Optionally, the features with null geometries can be saved to a separate output.

If 'Also remove empty geometries' is checked, the algorithm removes features whose geometries have no coordinates, i.e., geometries that are empty. In that case, also the null output will reflect this option, containing both null and empty geometries.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`REMOVE_EMPTY`	boolean	no	False	Also remove empty geometries
`OUTPUT`	sink	no	—	Non null geometries
`NULL_OUTPUT`	sink	no	—	Null geometries

Outputs: `OUTPUT` (outputVector), `NULL_OUTPUT` (outputVector)

Example usage

```
run native:removenullgeometries INPUT=input.gpkg REMOVE_EMPTY=false OUTPUT=output.gpkg
```

This calls **Remove null geometries**: `INPUT=input.gpkg` — Input layer; `REMOVE_EMPTY=false` — Also remove empty geometries; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:removepartsbyarea` — Remove parts by area

group: Vector geometry

This algorithm takes a polygon layer and removes polygons which are smaller than a specified area.

If the input geometry is a multipart geometry, then the parts will be filtered by their individual areas. If no parts match the required minimum area, then the feature will be skipped and omitted from the output layer.

If the input geometry is a singlepart geometry, then the feature will be skipped if the geometry's area is below the required size and omitted from the output layer.

The area will be calculated using Cartesian calculations in the source layer's coordinate reference system.

Attributes are not modified.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`MIN_AREA`	area	no	0.0	Remove parts with area less than
`OUTPUT`	sink	yes	—	Cleaned

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:removepartsbyarea INPUT=input.gpkg MIN_AREA=0 OUTPUT=output.gpkg
```

This calls **Remove parts by area**: `INPUT=input.gpkg` — Input layer; `MIN_AREA=0` — Remove parts with area less than; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:removepartsbylength` — Remove parts by length

group: Vector geometry

This algorithm takes a line layer and removes lines which are shorter than a specified length.

If the input geometry is a multipart geometry, then the parts will be filtered by their individual lengths. If no parts match the required minimum length, then the feature will be skipped and omitted from the output layer.

If the input geometry is a singlepart geometry, then the feature will be skipped if the geometry's length is below the required size and omitted from the output layer.

The length will be calculated using Cartesian calculations in the source layer's coordinate reference system.

Attributes are not modified.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`MIN_LENGTH`	distance	no	0.0	Remove parts with lengths less than
`OUTPUT`	sink	yes	—	Cleaned

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:removepartsbylength INPUT=input.gpkg MIN_LENGTH=0 OUTPUT=output.gpkg
```

This calls **Remove parts by length**: `INPUT=input.gpkg` — Input layer; `MIN_LENGTH=0` — Remove parts with lengths less than; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:renamelayer` — Rename layer

group: Modeler tools

This algorithm renames a layer.

Parameter	Type	Required	Default	Description
`INPUT`	layer	yes	—	Layer
`NAME`	string	yes	—	New name

Outputs: `OUTPUT` (outputLayer)

Example usage

```
run native:renamelayer INPUT=input.gpkg NAME=value
```

This calls **Rename layer**: `INPUT=input.gpkg` — Layer; `NAME=value` — New name. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:renametablefield — Rename field ★

group: Vector table · **niva verb:** `renamefield`

This algorithm renames an existing field from a vector layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	yes	—	Field to rename
NEW_NAME	string	yes	—	New field name
OUTPUT	sink	yes	—	Renamed

Outputs: OUTPUT (outputVector)

Example usage

```
run native:renametablefield INPUT=input.gpkg FIELD=field1 NEW_NAME=value
↪  OUTPUT=output.gpkg
```

This calls **Rename field**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — Field to rename; `NEW_NAME=value` — New field name; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:repairshapefile — Repair Shapefile

group: Vector general

Repairs a broken Shapefile by recreating missing or broken SHX files.

Parameter	Type	Required	Default	Description
INPUT	file	yes	—	Input Shapefile

Outputs: OUTPUT (outputVector)

Example usage

```
run native:repairshapefile INPUT=input.dat
```

This calls **Repair Shapefile**: `INPUT=input.dat` — Input Shapefile. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:reprojectlayer — Reproject layer ★

group: Vector general · **niva verb:** `reproject`

This algorithm reprojects a vector layer. It creates a new layer with the same features as the input one, but with geometries reprojected to a new CRS.

Attributes are not modified by this algorithm.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
TARGET_CRS	crs	no	EPSG:4326	Target CRS
CONVERT_CURVED_GEOME-TRIES	boolean	no	False	Convert curved geometries to straight segments
TRANSFORM_Z	boolean	no	False	Also transform Z coordinates
OPERATION *(adv)*	coordinateoperation	no	—	Coordinate operation
OUTPUT	sink	yes	—	Reprojected

Outputs: OUTPUT (outputVector)

Example usage

```
run native:reprojectlayer INPUT=input.gpkg TARGET_CRS=EPSG:6346
 ↪  CONVERT_CURVED_GEOMETRIES=false TRANSFORM_Z=false OUTPUT=output.gpkg
```

This calls **Reproject layer**: INPUT=input.gpkg — Input layer; TARGET_CRS=EPSG:6346 — Target CRS; CONVERT_CURVED_GEOMETRIES=false — Convert curved geometries to straight segments; TRANSFORM_Z=false — Also transform Z coordinates; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rescaleraster — Rescale raster

group: Raster analysis

This algorithm rescales a raster layer to a new value range, while preserving the shape (distribution) of the raster's histogram (pixel values). Input values are mapped using a linear interpolation from the source raster's minimum and maximum pixel values to the destination minimum and maximum pixel range.

By default the algorithm preserves the original NoData value, but there is an option to override it.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input raster
BAND	band	no	1	Band number
MINIMUM	number	no	0	New minimum value
MAXIMUM	number	no	255	New maximum value
NODATA	number	no	—	New NoData value
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Rescaled

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:rescaleraster INPUT=dem.tif BAND=1 MINIMUM=0 MAXIMUM=255 NODATA=10
 ↪  CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Rescale raster**: INPUT=dem.tif — Input raster; BAND=1 — Band number; MINIMUM=0 — New minimum value; MAXIMUM=255 — New maximum value; NODATA=10 — New NoData value; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:retainfields — Retain fields ★

group: Vector table · **niva verb:** keepfields

This algorithm takes a vector layer and generates a new one that retains only the selected fields. All other fields will be dropped.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELDS	field	yes	—	Fields to retain
OUTPUT	sink	yes	—	Retained fields

Outputs: OUTPUT (outputVector)

Example usage

```
run native:retainfields INPUT=input.gpkg FIELDS=field1 OUTPUT=output.gpkg
```

This calls **Retain fields**: INPUT=input.gpkg — Input layer; FIELDS=field1 — Fields to retain; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:reverselinedirection — Reverse line direction

group: Vector geometry

This algorithm reverses the direction of curve or LineString geometries.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Reversed

Outputs: OUTPUT (outputVector)

Example usage

```
run native:reverselinedirection INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Reverse line direction**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:rotatefeatures — Rotate

group: Vector geometry

This algorithm rotates feature geometries by the specified angle clockwise.

Optionally, the rotation can occur around a preset point. If not set the rotation occurs around each feature's centroid.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ANGLE	number	no	0.0	Rotation (degrees clockwise)
ANCHOR	point	no	—	Rotation anchor point
OUTPUT	sink	yes	—	Rotated

Outputs: OUTPUT (outputVector)

Example usage

```
run native:rotatefeatures INPUT=input.gpkg ANGLE=0 ANCHOR=100,100 OUTPUT=output.gpkg
```

This calls **Rotate**: INPUT=input.gpkg — Input layer; ANGLE=0 — Rotation (degrees clockwise); ANCHOR=100,100 — Rotation anchor point; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:roundness` — Roundness

group: Vector geometry

Calculates the roundness of each feature and stores it as a new field. The input vector layer must contain polygons.

The roundness of a polygon is defined as 4π × polygon area / perimeter². The roundness value varies between 0 and 1. A perfect circle has a roundness of 1, while a completely flat polygon has a roundness of 0.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Roundness

Outputs: OUTPUT (outputVector)

Example usage

```
run native:roundness INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Roundness**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:roundrastervalues` — Round raster

group: Raster analysis

This algorithm rounds the cell values of a raster dataset to the specified number of decimals. Alternatively, a negative number of decimal places may be used to round values to powers of a base n (specified in the advanced parameter Base n). For example, with a Base value n of 10 and Decimal places of -1 the algorithm rounds cell values to multiples of 10, -2 rounds to multiples of 100, and so on. Arbitrary base values may be chosen, the algorithm applies the same multiplicative principle. Rounding cell values to multiples of a base n may be used to generalize raster layers. The algorithm preserves the data type of the input raster. Therefore byte/integer rasters can only be rounded to multiples of a base n, otherwise a warning is raised and the raster gets copied as byte/integer raster.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input raster
BAND	band	no	1	Band number
ROUNDING_DIRECTION	enum	no	1	Rounding direction — options: 0=Round up, 1=Round to nearest, 2=Round down
DECIMAL_PLACES	number	no	2	Number of decimals places
BASE_N *(adv)*	number	no	10	Base n for rounding to multiples of n
CREATE_OPTIONS	string	no	—	Creation options
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Output raster

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:roundrastervalues INPUT=dem.tif BAND=1 ROUNDING_DIRECTION=1 DECIMAL_PLACES=2
↪  CREATE_OPTIONS=value OUTPUT=output.tif
```

This calls **Round raster**: INPUT=dem.tif — Input raster; BAND=1 — Band number; ROUNDING_DIRECTION=1 selects *Round to nearest*; DECIMAL_PLACES=2 — Number of decimals places; CREATE_OPTIONS=value — Creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:ruggednessindex — Ruggedness index

group: Raster terrain analysis

This algorithm calculates the quantitative measurement of terrain heterogeneity described by Riley et al. (1999).

It is calculated for every location, by summarizing the change in elevation within the 3x3 pixel grid. Each pixel contains the difference in elevation from a center cell and the 8 cells surrounding it.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Elevation layer
Z_FACTOR	number	no	1.0	Z factor
NODATA *(adv)*	number	no	-9999.0	Output NoData value
CREATION_OPTIONS *(adv)*	string	no	—	Creation options
OUTPUT	rasterDestination	yes	—	Ruggedness

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:ruggednessindex INPUT=dem.tif Z_FACTOR=1 OUTPUT=output.tif
```

This calls **Ruggedness index**: INPUT=dem.tif — Elevation layer; Z_FACTOR=1 — Z factor; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:savefeatures — Save vector features to file

group: Vector general

This algorithm saves vector features to a specified file dataset.

For dataset formats supporting layers, an optional layer name parameter can be used to specify a custom string.

Optional GDAL-defined dataset and layer options can be specified. For more information on this, read the online GDAL documentation.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Vector features
OUTPUT	fileDestination	yes	—	Saved features
LAYER_NAME *(adv)*	string	no	—	Layer name
DATASOURCE_OPTIONS *(adv)*	string	no	—	GDAL dataset options (separate individual options with semicolons)
LAYER_OPTIONS *(adv)*	string	no	—	GDAL layer options (separate individual options with semicolons)
ACTION_ON_EXISTING_FILE *(adv)*	enum	no	0	Action to take on pre-existing file — options: 0=Create or overwrite file, 1=Create or overwrite layer, 2=Append features to existing layer, but do not create new fields, 3=Append features to existing layer, and create new fields if needed

Outputs: OUTPUT (outputFile), FILE_PATH (outputString), LAYER_NAME (outputString)

Example usage

```
run native:savefeatures INPUT=input.gpkg OUTPUT=output.txt
```

This calls **Save vector features to file**: INPUT=input.gpkg — Vector features; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:savelog` — Save log to file

group: Modeler tools

This algorithm saves the model's execution log to a file. Optionally, the log can be saved in a HTML formatted version.

Parameter	Type	Required	Default	Description
OUTPUT	fileDestination	yes	—	Log file
USE_HTML	boolean	no	False	Use HTML formatting

Outputs: OUTPUT (outputHtml)

Example usage

```
run native:savelog USE_HTML=false OUTPUT=output.txt
```

This calls **Save log to file**: `USE_HTML=false` — Use HTML formatting; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:saveselectedfeatures` — Extract selected features

group: Vector general

This algorithm creates a new layer with all the selected features in a given vector layer.

If the selected layer has no selected features, the newly created layer will be empty.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
OUTPUT	sink	yes	—	Selected features

Outputs: OUTPUT (outputVector)

Example usage

```
run native:saveselectedfeatures INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Extract selected features**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:segmentizebymaxangle` — Segmentize by maximum angle

group: Vector geometry

This algorithm segmentizes a geometry by converting curved sections to linear sections.

The segmentization is performed by specifying the maximum allowed radius angle between vertices on the straightened geometry (e.g the angle of the arc created from the original arc center to consecutive output vertices on the linearized geometry).

Non-curved geometries will be retained without change.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
ANGLE	number	no	5.0	Maximum angle between vertices (degrees)
OUTPUT	sink	yes	—	Segmentized

Outputs: OUTPUT (outputVector)

Example usage

```
run native:segmentizebymaxangle INPUT=input.gpkg ANGLE=5 OUTPUT=output.gpkg
```

This calls **Segmentize by maximum angle**: `INPUT=input.gpkg` — Input layer; `ANGLE=5` — Maximum angle between vertices (degrees); `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:segmentizebymaxdistance` — Segmentize by maximum distance

group: Vector geometry

This algorithm segmentizes a geometry by converting curved sections to linear sections.

The segmentization is performed by specifying the maximum allowed offset distance between the original curve and the segmentized representation.

Non-curved geometries will be retained without change.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
DISTANCE	distance	no	1.0	Maximum offset distance
OUTPUT	sink	yes	—	Segmentized

Outputs: OUTPUT (outputVector)

Example usage

```
run native:segmentizebymaxdistance INPUT=input.gpkg DISTANCE=1 OUTPUT=output.gpkg
```

This calls **Segmentize by maximum distance**: `INPUT=input.gpkg` — Input layer; `DISTANCE=1` — Maximum offset distance; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:selectbylocation` — Select by location

group: Vector selection

This algorithm creates a selection in a vector layer. The criteria for selecting features is based on the spatial relationship between each feature and the features in an additional layer.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Select features from
PREDICATE	enum	no	[0]	Where the features (geometric predicate) — options: 0=intersect, 1=contain, 2=disjoint, 3=equal, 4=touch, 5=overlap, 6=are within, 7=cross
INTERSECT	source	yes	—	By comparing to the features from
METHOD	enum	no	0	Modify current selection by — options: 0=creating new selection, 1=adding to current selection, 2=selecting within current selection, 3=removing from current selection

Example usage

```
run native:selectbylocation INPUT=input.gpkg INTERSECT=input.gpkg PREDICATE=0 METHOD=0
```

This calls **Select by location**: `INPUT=input.gpkg` — Select features from; `INTERSECT=input.gpkg` — By comparing to the features from; `PREDICATE=0` selects *intersect*; `METHOD=0` selects *creating new selection*. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:selectwithindistance` — Select within distance

group: Vector selection

This algorithm creates a selection in a vector layer. Features are selected wherever they are within the specified maximum distance from the features in an additional reference layer.

Parameter	Type	Required	Default	Description
`INPUT`	vector	yes	—	Select features from
`REFERENCE`	source	yes	—	By comparing to the features from
`DISTANCE`	distance	no	100	Where the features are within
`METHOD`	enum	no	0	Modify current selection by — options: 0=creating new selection, 1=adding to current selection, 2=selecting within current selection, 3=removing from current selection

Example usage

```
run native:selectwithindistance INPUT=input.gpkg REFERENCE=input.gpkg DISTANCE=100
↪  METHOD=0
```

This calls **Select within distance**: `INPUT=input.gpkg` — Select features from; `REFERENCE=input.gpkg` — By comparing to the features from; `DISTANCE=100` — Where the features are within; `METHOD=0` selects *creating new selection*. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:serviceareafromlayer` — Service area (from layer)

group: Network analysis

This algorithm creates a new vector layer with all the edges or parts of edges of a network line layer that can be reached within a distance or a time, starting from features of a point layer. The distance and the time (both referred to as ”travel cost”) must be specified respectively in the network layer units or in hours.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Vector layer representing network
`STRATEGY`	enum	no	0	Path type to calculate — options: 0=Shortest, 1=Fastest
`DIRECTION_FIELD *(adv)*`	field	no	—	Direction field
`VALUE_FORWARD *(adv)*`	string	no	—	Value for forward direction
`VALUE_BACKWARD *(adv)*`	string	no	—	Value for backward direction
`VALUE_BOTH *(adv)*`	string	no	—	Value for both directions
`DEFAULT_DIRECTION *(adv)*`	enum	no	2	Default direction — options: 0=Forward direction, 1=Backward direction, 2=Both directions
`SPEED_FIELD *(adv)*`	field	no	—	Speed field
`DEFAULT_SPEED *(adv)*`	number	no	50	Default speed (km/h)
`TOLERANCE *(adv)*`	distance	no	0	Topology tolerance
`START_POINTS`	source	yes	—	Vector layer with start points
`TRAVEL_COST`	number	no	0	Travel cost (distance for 'Shortest', time for 'Fastest')
`TRAVEL_COST2`	number	no	0	Travel cost (distance for 'Shortest', time for 'Fastest')
`INCLUDE_BOUNDS *(adv)*`	boolean	no	False	Include upper/lower bound points
`POINT_TOLERANCE *(adv)*`	distance	no	—	Maximum point distance from network
`OUTPUT_LINES`	sink	no	—	Service area (lines)

Parameter	Type	Required	Default	Description
OUTPUT	sink	no	—	Service area (boundary nodes)
OUTPUT_NON_ROUTABLE	sink	no	—	Non-routable features

Outputs: OUTPUT_LINES (outputVector), OUTPUT (outputVector), OUTPUT_NON_ROUTABLE (outputVector)

Example usage

```
run native:serviceareafromlayer INPUT=input.gpkg START_POINTS=input.gpkg STRATEGY=0
↪  TRAVEL_COST=0 TRAVEL_COST2=0 OUTPUT_LINES=output_lines.gpkg
```

This calls **Service area (from layer)**: INPUT=input.gpkg — Vector layer representing network; START_POINTS=input.gpkg — Vector layer with start points; STRATEGY=0 selects *Shortest*; TRAVEL_COST=0 — Travel cost (distance for 'Shortest', time for 'Fastest'); TRAVEL_COST2=0 — Travel cost (distance for 'Shortest', time for 'Fastest'); OUTPUT_LINES=output_lines.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:serviceareafrompoint — Service area (from point)

group: Network analysis

This algorithm creates a new vector layer with all the edges or parts of edges of a network line layer that can be reached within a distance or a time, starting from a point feature. The distance and the time (both referred to as ”travel cost”) must be specified respectively in the network layer units or in hours.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Vector layer representing network
STRATEGY	enum	no	0	Path type to calculate — options: 0=Shortest, 1=Fastest
DIRECTION_FIELD *(adv)*	field	no	—	Direction field
VALUE_FORWARD *(adv)*	string	no	—	Value for forward direction
VALUE_BACKWARD *(adv)*	string	no	—	Value for backward direction
VALUE_BOTH *(adv)*	string	no	—	Value for both directions
DEFAULT_DIRECTION *(adv)*	enum	no	2	Default direction — options: 0=Forward direction, 1=Backward direction, 2=Both directions
SPEED_FIELD *(adv)*	field	no	—	Speed field
DEFAULT_SPEED *(adv)*	number	no	50	Default speed (km/h)
TOLERANCE *(adv)*	distance	no	0	Topology tolerance
START_POINT	point	yes	—	Start point
TRAVEL_COST	number	no	0	Travel cost (distance for 'Shortest', time for 'Fastest')
TRAVEL_COST2	number	no	0	Travel cost (distance for 'Shortest', time for 'Fastest')
POINT_TOLERANCE *(adv)*	distance	no	—	Maximum point distance from network
INCLUDE_BOUNDS *(adv)*	boolean	no	False	Include upper/lower bound points
OUTPUT_LINES	sink	no	—	Service area (lines)
OUTPUT	sink	no	—	Service area (boundary nodes)

Outputs: OUTPUT_LINES (outputVector), OUTPUT (outputVector)

Example usage

```
run native:serviceareafrompoint INPUT=input.gpkg START_POINT=100,100 STRATEGY=0
↪  TRAVEL_COST=0 TRAVEL_COST2=0 OUTPUT_LINES=output_lines.gpkg
```

This calls **Service area (from point)**: INPUT=input.gpkg — Vector layer representing network; START_POINT=100,100 — Start point; STRATEGY=0 selects *Shortest*; TRAVEL_COST=0 — Travel cost (distance for 'Shortest', time for 'Fastest'); TRAVEL_COST2=0 — Travel cost (distance for 'Shortest',

time for 'Fastest'); OUTPUT_LINES=output_lines.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:setlayerencoding — Set layer encoding

group: Vector general

This algorithm sets the encoding used for reading a layer's attributes. No permanent changes are made to the layer, rather it affects only how the layer is read during the current session.

Changing the encoding is only supported for some vector layer data sources.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
ENCODING	string	yes	—	Encoding

Outputs: OUTPUT (outputVector)

Example usage

```
run native:setlayerencoding INPUT=input.gpkg ENCODING=value
```

This calls **Set layer encoding**: INPUT=input.gpkg — Input layer; ENCODING=value — Encoding. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:setlayermetadata — Set layer metadata

group: Metadata tools

This algorithm applies the metadata to a layer. The metadata must be defined as QMD file.

Any existing metadata in the layer will be replaced.

Parameter	Type	Required	Default	Description
INPUT	layer	yes	—	Layer
METADATA	file	yes	—	Metadata file
DEFAULT	boolean	no	False	Save metadata as default

Outputs: OUTPUT (outputLayer)

Example usage

```
run native:setlayermetadata INPUT=input.gpkg METADATA=input.dat DEFAULT=false
```

This calls **Set layer metadata**: INPUT=input.gpkg — Layer; METADATA=input.dat — Metadata file; DEFAULT=false — Save metadata as default. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:setlayerstyle — Set layer style

group: Cartography

This algorithm applies the style to a layer. The style must be defined as QML file.

Parameter	Type	Required	Default	Description
INPUT	layer	yes	—	Layer
STYLE	file	yes	—	Style file

Outputs: OUTPUT (outputLayer)

Example usage

```
run native:setlayerstyle INPUT=input.gpkg STYLE=input.dat
```

This calls **Set layer style**: INPUT=input.gpkg — Layer; STYLE=input.dat — Style file. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:setmetadatafields — Set metadata fields

group: Metadata tools

This algorithm sets various metadata fields for a layer.

Parameter	Type	Required	Default	Description
INPUT	layer	yes	—	Layer
IDENTIFIER	string	no	—	Identifier
PARENT_IDENTIFIER	string	no	—	Parent identifier
TITLE	string	no	—	Title
TYPE	string	no	—	Type
LANGUAGE	string	no	—	Language
ENCODING	string	no	—	Encoding
ABSTRACT	string	no	—	Abstract
CRS	crs	no	—	Coordinate reference system
FEES	string	no	—	Fees
IGNORE_EMPTY	boolean	no	False	Ignore empty fields

Outputs: OUTPUT (outputLayer)

Example usage

```
run native:setmetadatafields INPUT=input.gpkg IDENTIFIER=value PARENT_IDENTIFIER=value
↪  TITLE=value TYPE=value LANGUAGE=value ENCODING=value
```

This calls **Set metadata fields**: INPUT=input.gpkg — Layer; IDENTIFIER=value — Identifier; PARENT_IDENTIFIER=value — Parent identifier; TITLE=value — Title; TYPE=value — Type; LANGUAGE=value — Language; ENCODING=value — Encoding. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:setmfromraster — Set M value from raster

group: Vector geometry

This algorithm sets the M value for every vertex in the feature geometry to a value sampled from a band within a raster layer.

The raster values can optionally be scaled by a preset amount and an offset can be algebraically added.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
RASTER	raster	yes	—	Raster layer
BAND	band	no	1	Band number
NODATA	number	no	0.0	Value for NoData or non-intersecting vertices
SCALE	number	no	1.0	Scale factor
OFFSET	number	no	0.0	Offset

Parameter	Type	Required	Default	Description
OUTPUT	sink	yes	—	Draped

Outputs: OUTPUT (outputVector)

Example usage

```
run native:setmfromraster INPUT=input.gpkg RASTER=dem.tif BAND=1 NODATA=0 SCALE=1 OFFSET=0
↪  OUTPUT=output.gpkg
```

This calls **Set M value from raster**: INPUT=input.gpkg — Input layer; RASTER=dem.tif — Raster layer; BAND=1 — Band number; NODATA=0 — Value for NoData or non-intersecting vertices; SCALE=1 — Scale factor; OFFSET=0 — Offset; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:setmvalue — Set M value

group: Vector geometry

This algorithm sets the M value for geometries in a layer.

If M values already exist in the layer, they will be overwritten with the new value. If no M values exist, the geometry will be upgraded to include M values and the specified value used as the initial M value for all geometries.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
M_VALUE	number	no	0.0	M Value
OUTPUT	sink	yes	—	M Added

Outputs: OUTPUT (outputVector)

Example usage

```
run native:setmvalue INPUT=input.gpkg M_VALUE=0 OUTPUT=output.gpkg
```

This calls **Set M value**: INPUT=input.gpkg — Input layer; M_VALUE=0 — M Value; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:setprojectvariable — Set project variable

group: Modeler tools

This algorithm sets an expression variable for the current project.

Parameter	Type	Required	Default	Description
NAME	string	yes	—	Variable name
VALUE	string	yes	—	Variable value

Example usage

```
run native:setprojectvariable NAME=value VALUE=value
```

This calls **Set project variable**: NAME=value — Variable name; VALUE=value — Variable value. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:setzfromraster — Drape (set Z value from raster)

group: Vector geometry

This algorithm sets the z value of every vertex in the feature geometry to a value sampled from a band within a raster layer.

The raster values can optionally be scaled by a preset amount and an offset can be algebraically added.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
RASTER	raster	yes	—	Raster layer
BAND	band	no	1	Band number
NODATA	number	no	0.0	Value for NoData or non-intersecting vertices
SCALE	number	no	1.0	Scale factor
OFFSET	number	no	0.0	Offset
OUTPUT	sink	yes	—	Draped

Outputs: OUTPUT (outputVector)

Example usage

```
run native:setzfromraster INPUT=input.gpkg RASTER=dem.tif BAND=1 NODATA=0 SCALE=1 OFFSET=0
↪  OUTPUT=output.gpkg
```

This calls **Drape (set Z value from raster)**: INPUT=input.gpkg — Input layer; RASTER=dem.tif — Raster layer; BAND=1 — Band number; NODATA=0 — Value for NoData or non-intersecting vertices; SCALE=1 — Scale factor; OFFSET=0 — Offset; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:setzvalue — Set Z value

group: Vector geometry

This algorithm sets the Z value for geometries in a layer.

If Z values already exist in the layer, they will be overwritten with the new value. If no Z values exist, the geometry will be upgraded to include Z values and the specified value used as the initial Z value for all geometries.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
Z_VALUE	number	no	0.0	Z Value
OUTPUT	sink	yes	—	Z Added

Outputs: OUTPUT (outputVector)

Example usage

```
run native:setzvalue INPUT=input.gpkg Z_VALUE=0 OUTPUT=output.gpkg
```

This calls **Set Z value**: INPUT=input.gpkg — Input layer; Z_VALUE=0 — Z Value; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:shortestline — Shortest line between features

group: Vector analysis

This algorithm creates a line layer as the shortest line between the source and the destination layer. By default only the first nearest feature of the destination layer is taken into account. The n-nearest neighboring features number can be specified.

If a maximum distance is specified, then only features which are closer than this distance will be considered.

The output features will contain all the source layer attributes, all the attributes from the n-nearest feature and the additional field of the distance.

This algorithm uses purely Cartesian calculations for distance, and does not consider geodetic or ellipsoid properties when determining feature proximity. The measurement and output coordinate system is based on the coordinate system of the source layer.

Parameter	Type	Required	Default	Description
SOURCE	source	yes	—	Source layer
DESTINATION	source	yes	—	Destination layer
METHOD	enum	no	0	Method — options: 0=Distance to Nearest Point on feature, 1=Distance to Feature Centroid
NEIGHBORS	number	no	1	Maximum number of neighbors
DISTANCE	distance	no	—	Maximum distance
OUTPUT	sink	yes	—	Shortest lines

Outputs: OUTPUT (outputVector)

Example usage

```
run native:shortestline SOURCE=input.gpkg DESTINATION=input.gpkg METHOD=0 NEIGHBORS=1
↪  DISTANCE=10 OUTPUT=output.gpkg
```

This calls **Shortest line between features**: SOURCE=input.gpkg — Source layer; DESTINATION=input.gpkg — Destination layer; METHOD=0 selects *Distance to Nearest Point on feature*; NEIGHBORS=1 — Maximum number of neighbors; DISTANCE=10 — Maximum distance; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:shortestpathlayertopoint — Shortest path (layer to point)

group: Network analysis

This algorithm computes optimal (shortest or fastest) routes from multiple start points defined by a vector layer and a given end point.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Vector layer representing network
STRATEGY	enum	no	0	Path type to calculate — options: 0=Shortest, 1=Fastest
DIRECTION_FIELD *(adv)*	field	no	—	Direction field
VALUE_FORWARD *(adv)*	string	no	—	Value for forward direction
VALUE_BACKWARD *(adv)*	string	no	—	Value for backward direction
VALUE_BOTH *(adv)*	string	no	—	Value for both directions
DEFAULT_DIRECTION *(adv)*	enum	no	2	Default direction — options: 0=Forward direction, 1=Backward direction, 2=Both directions
SPEED_FIELD *(adv)*	field	no	—	Speed field
DEFAULT_SPEED *(adv)*	number	no	50	Default speed (km/h)
TOLERANCE *(adv)*	distance	no	0	Topology tolerance

Parameter	Type	Required	Default	Description
START_POINTS	source	yes	—	Vector layer with start points
END_POINT	point	yes	—	End point
POINT_TOLERANCE *(adv)*	distance	no	—	Maximum point distance from network
OUTPUT	sink	yes	—	Shortest path
OUTPUT_NON_ROUTABLE	sink	no	—	Non-routable features

Outputs: OUTPUT (outputVector), OUTPUT_NON_ROUTABLE (outputVector)

Example usage

```
run native:shortestpathlayertopoint INPUT=input.gpkg START_POINTS=input.gpkg
 ↪  END_POINT=100,100 STRATEGY=0 OUTPUT=output.gpkg
```

This calls **Shortest path (layer to point)**: INPUT=input.gpkg — Vector layer representing network; START_POINTS=input.gpkg — Vector layer with start points; END_POINT=100,100 — End point; STRATEGY=0 selects *Shortest*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:shortestpathpointtolayer — Shortest path (point to layer)

group: Network analysis

This algorithm computes optimal (shortest or fastest) route between a given start point and multiple end points defined by a point vector layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Vector layer representing network
STRATEGY	enum	no	0	Path type to calculate — options: 0=Shortest, 1=Fastest
DIRECTION_FIELD *(adv)*	field	no	—	Direction field
VALUE_FORWARD *(adv)*	string	no	—	Value for forward direction
VALUE_BACKWARD *(adv)*	string	no	—	Value for backward direction
VALUE_BOTH *(adv)*	string	no	—	Value for both directions
DEFAULT_DIRECTION *(adv)*	enum	no	2	Default direction — options: 0=Forward direction, 1=Backward direction, 2=Both directions
SPEED_FIELD *(adv)*	field	no	—	Speed field
DEFAULT_SPEED *(adv)*	number	no	50	Default speed (km/h)
TOLERANCE *(adv)*	distance	no	0	Topology tolerance
START_POINT	point	yes	—	Start point
END_POINTS	source	yes	—	Vector layer with end points
POINT_TOLERANCE *(adv)*	distance	no	—	Maximum point distance from network
OUTPUT	sink	yes	—	Shortest path
OUTPUT_NON_ROUTABLE	sink	no	—	Non-routable features

Outputs: OUTPUT (outputVector), OUTPUT_NON_ROUTABLE (outputVector)

Example usage

```
run native:shortestpathpointtolayer INPUT=input.gpkg START_POINT=100,100
 ↪  END_POINTS=input.gpkg STRATEGY=0 OUTPUT=output.gpkg
```

This calls **Shortest path (point to layer)**: INPUT=input.gpkg — Vector layer representing network; START_POINT=100,100 — Start point; END_POINTS=input.gpkg — Vector layer with end points; STRATEGY=0 selects *Shortest*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:shortestpathpointtopoint` — Shortest path (point to point)

group: Network analysis

This algorithm computes optimal (shortest or fastest) route between given start and end points.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Vector layer representing network
`STRATEGY`	enum	no	0	Path type to calculate — options: 0=Shortest, 1=Fastest
`DIRECTION_FIELD *(adv)*`	field	no	—	Direction field
`VALUE_FORWARD *(adv)*`	string	no	—	Value for forward direction
`VALUE_BACKWARD *(adv)*`	string	no	—	Value for backward direction
`VALUE_BOTH *(adv)*`	string	no	—	Value for both directions
`DEFAULT_DIRECTION *(adv)*`	enum	no	2	Default direction — options: 0=Forward direction, 1=Backward direction, 2=Both directions
`SPEED_FIELD *(adv)*`	field	no	—	Speed field
`DEFAULT_SPEED *(adv)*`	number	no	50	Default speed (km/h)
`TOLERANCE *(adv)*`	distance	no	0	Topology tolerance
`START_POINT`	point	yes	—	Start point
`END_POINT`	point	yes	—	End point
`OUTPUT`	sink	yes	—	Shortest path
`POINT_TOLERANCE *(adv)*`	distance	no	—	Maximum point distance from network

Outputs: `OUTPUT` (outputVector), `TRAVEL_COST` (outputNumber)

Example usage

```
run native:shortestpathpointtopoint INPUT=input.gpkg START_POINT=100,100 END_POINT=100,100
↪  STRATEGY=0 OUTPUT=output.gpkg
```

This calls **Shortest path (point to point)**: `INPUT=input.gpkg` — Vector layer representing network; `START_POINT=100,100` — Start point; `END_POINT=100,100` — End point; `STRATEGY=0` selects *Shortest*; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:shpencodinginfo` — Extract Shapefile encoding

group: Vector general

This algorithm extracts the attribute encoding information embedded in a Shapefile.

Both the encoding specified by an optional .cpg file and any encoding details present in the .dbf LDID header block are considered.

Parameter	Type	Required	Default	Description
`INPUT`	file	yes	—	Input layer

Outputs: `ENCODING` (outputString), `CPG_ENCODING` (outputString), `LDID_ENCODING` (outputString)

Example usage

```
run native:shpencodinginfo INPUT=input.dat
```

This calls **Extract Shapefile encoding**: `INPUT=input.dat` — Input layer. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:simplifygeometries — Simplify ★

group: Vector geometry · **niva verb:** `simplify`

This algorithm simplifies the geometries in a line or polygon layer. It creates a new layer with the same features as the ones in the input layer, but with geometries containing a lower number of vertices.

The algorithm gives a choice of simplification methods, including distance based (the ”Douglas-Peucker” algorithm), area based (”Visvalingam” algorithm) and snapping geometries to a grid.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
METHOD	enum	no	0	Simplification method — options: 0=Distance (Douglas-Peucker), 1=Snap to grid, 2=Area (Visvalingam)
TOLERANCE	distance	no	1.0	Tolerance
OUTPUT	sink	yes	—	Simplified

Outputs: OUTPUT (outputVector)

Example usage

```
run native:simplifygeometries INPUT=input.gpkg METHOD=0 TOLERANCE=1 OUTPUT=output.gpkg
```

This calls **Simplify**: INPUT=input.gpkg — Input layer; METHOD=0 selects *Distance (Douglas-Peucker)*; TOLERANCE=1 — Tolerance; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:singlesidedbuffer — Single sided buffer

group: Vector geometry

This algorithm buffers lines by a specified distance on one side of the line only.

The segments parameter controls the number of line segments to use to approximate a quarter circle when creating rounded buffers. The join style parameter specifies whether round, miter or beveled joins should be used when buffering corners in a line. The miter limit parameter is only applicable for miter join styles, and controls the maximum distance from the buffer to use when creating a mitered join.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
DISTANCE	distance	no	10	Distance
SIDE	enum	no	0	Side — options: 0=Left, 1=Right
SEGMENTS	number	no	8	Segments
JOIN_STYLE	enum	no	0	Join style — options: 0=Round, 1=Miter, 2=Bevel
MITER_LIMIT	number	no	2	Miter limit
OUTPUT	sink	yes	—	Buffered

Outputs: OUTPUT (outputVector)

Example usage

```
run native:singlesidedbuffer INPUT=input.gpkg DISTANCE=10 SIDE=0 SEGMENTS=8 JOIN_STYLE=0
↪ MITER_LIMIT=2 OUTPUT=output.gpkg
```

This calls **Single sided buffer**: INPUT=input.gpkg — Input layer; DISTANCE=10 — Distance; SIDE=0 selects *Left*; SEGMENTS=8 — Segments; JOIN_STYLE=0 selects *Round*; MITER_LIMIT=2 — Miter limit; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:slope` — Slope

group: Raster terrain analysis

This algorithm calculates the angle of inclination of the terrain from an input raster layer. The slope is expressed in degrees.

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Elevation layer
`Z_FACTOR`	number	no	1.0	Z factor
`NODATA *(adv)*`	number	no	-9999.0	Output NoData value
`CREATION_OPTIONS *(adv)*`	string	no	—	Creation options
`OUTPUT`	rasterDestination	yes	—	Slope

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run native:slope INPUT=dem.tif Z_FACTOR=1 OUTPUT=output.tif
```

This calls **Slope**: `INPUT=dem.tif` — Elevation layer; `Z_FACTOR=1` — Z factor; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:smoothgeometry` — Smooth ★

group: Vector geometry · **niva verb:** `smooth`

This algorithm smooths the geometries in a line or polygon layer. It creates a new layer with the same features as the ones in the input layer, but with geometries containing a higher number of vertices and corners in the geometries smoothed out.

The iterations parameter dictates how many smoothing iterations will be applied to each geometry. A higher number of iterations results in smoother geometries with the cost of greater number of nodes in the geometries.

The offset parameter controls how ”tightly” the smoothed geometries follow the original geometries. Smaller values results in a tighter fit, and larger values will create a looser fit.

The maximum angle parameter can be used to prevent smoothing of nodes with large angles. Any node where the angle of the segments to either side is larger than this will not be smoothed. For example, setting the maximum angle to 90 degrees or lower would preserve right angles in the geometry.

If input geometries contain Z or M values, these will also be smoothed and the output geometry will retain the same dimensionality as the input geometry.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`ITERATIONS`	number	no	1	Iterations
`OFFSET`	number	no	0.25	Offset
`MAX_ANGLE`	number	no	180.0	Maximum node angle to smooth
`OUTPUT`	sink	yes	—	Smoothed

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:smoothgeometry INPUT=input.gpkg ITERATIONS=1 OFFSET=0.25 MAX_ANGLE=180
↪ OUTPUT=output.gpkg
```

This calls **Smooth**: `INPUT=input.gpkg` — Input layer; `ITERATIONS=1` — Iterations; `OFFSET=0.25` — Offset; `MAX_ANGLE=180` — Maximum node angle to smooth; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:snapgeometries` — Snap geometries to layer ★

group: Vector geometry · **niva verb:** snap

This algorithm snaps the geometries in a layer. Snapping can be done either to the geometries from another layer, or to geometries within the same layer.

A tolerance is specified in layer units to control how close vertices need to be to the reference layer geometries before they are snapped.

Snapping occurs to both nodes and edges. Depending on the snapping behavior, either nodes or edges will be preferred.

Vertices will be inserted or removed as required to make the geometries match the reference geometries.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`REFERENCE_LAYER`	source	yes	—	Reference layer
`TOLERANCE`	distance	no	10.0	Tolerance
`BEHAVIOR`	enum	no	[0]	Behavior — options: 0=Prefer aligning nodes, insert extra vertices where required, 1=Prefer closest point, insert extra vertices where required, 2=Prefer aligning nodes, don't insert new vertices, 3=Prefer closest point, don't insert new vertices, 4=Move end points only, prefer aligning nodes, 5=Move end points only, prefer closest point, 6=Snap end points to end points only, 7=Snap to anchor nodes (single layer only)
`OUTPUT`	sink	yes	—	Snapped geometry

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:snapgeometries INPUT=input.gpkg REFERENCE_LAYER=input.gpkg TOLERANCE=10
 ↪  BEHAVIOR=0 OUTPUT=output.gpkg
```

This calls **Snap geometries to layer**: `INPUT=input.gpkg` — Input layer; `REFERENCE_LAYER=input.gpkg` — Reference layer; `TOLERANCE=10` — Tolerance; `BEHAVIOR=0` selects *Prefer aligning nodes, insert extra vertices where required*; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:snappointstogrid` — Snap points to grid

group: Vector geometry

This algorithm modifies the coordinates of geometries in a vector layer, so that all points or vertices are snapped to the closest point of the grid.

If the snapped geometry cannot be calculated (or is totally collapsed) the feature's geometry will be cleared.

Note that snapping to grid may generate an invalid geometry in some corner cases.

Snapping can be performed on the X, Y, Z or M axis. A grid spacing of 0 for any axis will disable snapping for that axis.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
HSPACING	distance	no	1	X Grid Spacing
VSPACING	distance	no	1	Y Grid Spacing
ZSPACING	number	no	0	Z Grid Spacing
MSPACING	number	no	0	M Grid Spacing
OUTPUT	sink	yes	—	Snapped

Outputs: OUTPUT (outputVector)

Example usage

```
run native:snappointstogrid INPUT=input.gpkg HSPACING=1 VSPACING=1 ZSPACING=0 MSPACING=0
↪  OUTPUT=output.gpkg
```

This calls **Snap points to grid**: INPUT=input.gpkg — Input layer; HSPACING=1 — X Grid Spacing; VSPACING=1 — Y Grid Spacing; ZSPACING=0 — Z Grid Spacing; MSPACING=0 — M Grid Spacing; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:spatialiteexecutesql — SpatiaLite execute SQL

group: Database

This algorithm executes a SQL command on a SpatiaLite database. The database is determined by an input layer or file.

Parameter	Type	Required	Default	Description
DATABASE	vector	yes	—	Database layer (or file)
SQL	string	yes	—	SQL query

Example usage

```
run native:spatialiteexecutesql DATABASE=input.gpkg SQL=value
```

This calls **SpatiaLite execute SQL**: DATABASE=input.gpkg — Database layer (or file); SQL=value — SQL query. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:spatialiteexecutesqlregistered — SpatiaLite execute SQL (registered DB)

group: Database

This algorithm executes a SQL command on a SpatiaLite database.

Parameter	Type	Required	Default	Description
DATABASE	providerconnection	yes	—	Database (connection name)
SQL	string	yes	—	SQL query

Example usage

```
run native:spatialiteexecutesqlregistered DATABASE=… SQL=value
```

This calls **SpatiaLite execute SQL (registered DB)**: DATABASE=… — Database (connection name); SQL=value — SQL query. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:splitfeaturesbycharacter` — Split features by character

group: Vector general

This algorithm splits features into multiple output features by splitting a field's value with a specified character.

For instance, if a layer contains features with multiple comma separated values contained in a single field, this algorithm can be used to split these values up across multiple output features.

Geometries and other attributes remain unchanged in the output.

Optionally, the separator string can be a regular expression for added flexibility.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`FIELD`	field	yes	—	Split using values in field
`CHAR`	string	yes	—	Split values using character
`REGEX *(adv)*`	boolean	yes	—	Use regular expression separator
`OUTPUT`	sink	yes	—	Split

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:splitfeaturesbycharacter INPUT=input.gpkg FIELD=field1 CHAR=value REGEX=false
↪  OUTPUT=output.gpkg
```

This calls **Split features by character**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — Split using values in field; `CHAR=value` — Split values using character; `REGEX=false` — Use regular expression separator; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:splitlinesbylength` — Split lines by maximum length

group: Vector geometry

This algorithm takes a line (or curve) layer and splits each feature into multiple parts, where each part is of a specified maximum length.

Z and M values at the start and end of the new line substrings are linearly interpolated from existing values.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`LENGTH`	distance	no	10	Maximum line length
`OUTPUT`	sink	yes	—	Split

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:splitlinesbylength INPUT=input.gpkg LENGTH=10 OUTPUT=output.gpkg
```

This calls **Split lines by maximum length**: `INPUT=input.gpkg` — Input layer; `LENGTH=10` — Maximum line length; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:splitvectorlayer — Split vector layer

group: Vector general

This algorithm splits input vector layer into multiple layers by specified unique ID field.

Each of the layers created in the output folder contains all features from the input layer with the same value for the specified attribute. The number of files generated is equal to the number of different values found for the specified attribute.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	yes	—	Unique ID field
PREFIX_FIELD	boolean	no	True	Add field prefix to file names
FILE_TYPE *(adv)*	enum	no	0	Output file type — options: 0=gpkg, 1=shp, 2=000, 3=csv, 4=dgn, 5=dxf, 6=fgb, 7=gdb, 8=geojson, 9=geojsonl, 10=geojsons, 11=gml, 12=gpkg.zip, 13=gpx, 14=json, 15=kml, 16=ods, 17=sql, 18=sqlite, 19=tab, 20=xlsx, 21=xml
OUTPUT	folderDestination	yes	—	Output directory

Outputs: OUTPUT (outputFolder), OUTPUT_LAYERS (outputMultilayer)

Example usage

```
run native:splitvectorlayer INPUT=input.gpkg FIELD=field1 PREFIX_FIELD=true OUTPUT=output/
```

This calls **Split vector layer**: INPUT=input.gpkg — Input layer; FIELD=field1 — Unique ID field; PREFIX_FIELD=true — Add field prefix to file names; OUTPUT=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:splitwithlines — Split with lines

group: Vector overlay

This algorithm splits the lines or polygons in one layer using the lines or polygon rings in another layer to define the breaking points. Intersection between geometries in both layers are considered as split points.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
LINES	source	yes	—	Split layer
OUTPUT	sink	yes	—	Split

Outputs: OUTPUT (outputVector)

Example usage

```
run native:splitwithlines INPUT=input.gpkg LINES=input.gpkg OUTPUT=output.gpkg
```

This calls **Split with lines**: INPUT=input.gpkg — Input layer; LINES=input.gpkg — Split layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:stdbscanclustering — ST-DBSCAN clustering

group: Vector analysis

Clusters point features based on a 2D implementation of spatiotemporal density-based clustering of applications with noise (ST-DBSCAN) algorithm.

For more details, please see the following papers:

- Ester, M., H. P. Kriegel, J. Sander, and X. Xu, ”A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, pp. 226-231. 1996
- Birant, Derya, and Alp Kut. ”ST-DBSCAN: An algorithm for clustering spatial–temporal data.” Data & Knowledge Engineering 60.1 (2007): 208-221.
- Peca, I., Fuchs, G., Vrotsou, K., Andrienko, N. V., & Andrienko, G. L. (2012). Scalable Cluster Analysis of Spatial Events. In EuroVA@ EuroVis.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
DATETIME_FIELD	field	yes	—	Date/time field
MIN_SIZE	number	no	5	Minimum cluster size
EPS	distance	no	1	Maximum distance between clustered points
EPS2	duration	no	0	Maximum time duration between clustered points
DBSCAN* *(adv)*	boolean	no	False	Treat border points as noise (DBSCAN*)
FIELD_NAME *(adv)*	string	no	CLUSTER_ID	Cluster field name
SIZE_FIELD_NAME *(adv)*	string	no	CLUSTER_SIZE	Cluster size field name
OUTPUT	sink	yes	—	Clusters

Outputs: OUTPUT (outputVector), NUM_CLUSTERS (outputNumber)

Example usage

```
run native:stdbscanclustering INPUT=input.gpkg DATETIME_FIELD=field1 MIN_SIZE=5 EPS=1
↪  EPS2=0 OUTPUT=output.gpkg
```

This calls **ST-DBSCAN clustering**: INPUT=input.gpkg — Input layer; DATETIME_FIELD=field1 — Date/time field; MIN_SIZE=5 — Minimum cluster size; EPS=1 — Maximum distance between clustered points; EPS2=0 — Maximum time duration between clustered points; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:stringconcatenation — String concatenation

group: Modeler tools

This algorithm concatenates two strings together.

Parameter	Type	Required	Default	Description
INPUT_1	string	yes	—	Input 1
INPUT_2	string	yes	—	Input 2

Outputs: CONCATENATION (outputString)

Example usage

```
run native:stringconcatenation INPUT_1=value INPUT_2=value
```

This calls **String concatenation**: INPUT_1=value — Input 1; INPUT_2=value — Input 2. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:stylefromproject — Create style database from project

group: Cartography

This algorithm extracts all style objects (including symbols, color ramps, text formats and label settings) from a QGIS project.

The extracted symbols are saved to a QGIS style database (XML format), which can be managed and imported via the Style Manager dialog.

Parameter	Type	Required	Default	Description
INPUT	file	no	—	Input project (leave blank to use current)
OUTPUT	fileDestination	yes	—	Output style database
OBJECTS	enum	no	[0, 1, 2, 3]	Objects to extract — options: 0=Symbols, 1=Color ramps, 2=Text formats, 3=Label settings

Outputs: OUTPUT (outputFile), SYMBOLS (outputNumber), COLORRAMPS (outputNumber), TEXTFORMATS (outputNumber), LABELSETTINGS (outputNumber)

Example usage

```
run native:stylefromproject INPUT=input.dat OBJECTS=0 OUTPUT=output.txt
```

This calls **Create style database from project**: INPUT=input.dat — Input project (leave blank to use current); OBJECTS=0 selects *Symbols*; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:subdivide — Subdivide ★

group: Vector geometry · **niva verb:** subdivide

This algorithm subdivides the geometry. The returned geometry will be a collection containing subdivided parts from the original geometry, where no part has more than the specified maximum number of nodes.

This is useful for dividing a complex geometry into less complex parts, which are better able to be spatially indexed and faster to perform further operations such as intersects on. The returned geometry parts may not be valid and may contain self-intersections.

Curved geometries will be segmentized before subdivision.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
MAX_NODES	number	no	256	Maximum nodes in parts
OUTPUT	sink	yes	—	Subdivided

Outputs: OUTPUT (outputVector)

Example usage

```
run native:subdivide INPUT=input.gpkg MAX_NODES=256 OUTPUT=output.gpkg
```

This calls **Subdivide**: INPUT=input.gpkg — Input layer; MAX_NODES=256 — Maximum nodes in parts; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:sumlinelengths — Sum line lengths

group: Vector analysis

This algorithm takes a polygon layer and a line layer and measures the total length of lines and the total number of them that cross each polygon.

The resulting layer has the same features as the input polygon layer, but with two additional attributes containing the length and count of the lines across each polygon. The names of these two fields can be configured in the algorithm parameters.

Parameter	Type	Required	Default	Description
POLYGONS	source	yes	—	Polygons
LINES	source	yes	—	Lines
LEN_FIELD	string	no	LENGTH	Lines length field name
COUNT_FIELD	string	no	COUNT	Lines count field name
OUTPUT	sink	yes	—	Line length

Outputs: OUTPUT (outputVector)

Example usage

```
run native:sumlinelengths POLYGONS=input.gpkg LINES=input.gpkg LEN_FIELD=LENGTH
 ↪  COUNT_FIELD=COUNT OUTPUT=output.gpkg
```

This calls **Sum line lengths**: POLYGONS=input.gpkg — Polygons; LINES=input.gpkg — Lines; LEN_FIELD=LENGTH — Lines length field name; COUNT_FIELD=COUNT — Lines count field name; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:surfacetopolygon — Surface to polygon

group: Mesh

This algorithm exports a polygon layer containing a mesh layer's boundary. It may contain holes and it may be a multi-part polygon.

Parameter	Type	Required	Default	Description
INPUT	mesh	yes	—	Input mesh layer
CRS_OUTPUT	crs	no	—	Output coordinate system
OUTPUT	sink	yes	—	Output vector layer

Outputs: OUTPUT (outputVector)

Example usage

```
run native:surfacetopolygon INPUT=mesh.nc CRS_OUTPUT=EPSG:6346 OUTPUT=output.gpkg
```

This calls **Surface to polygon**: INPUT=mesh.nc — Input mesh layer; CRS_OUTPUT=EPSG:6346 — Output coordinate system; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:swapxy — Swap X and Y coordinates ★

group: Vector geometry · **niva verb:** swapxy

This algorithm swaps the X and Y coordinate values in input geometries. It can be used to repair geometries which have accidentally had their latitude and longitude values reversed.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Swapped

Outputs: OUTPUT (outputVector)

Example usage

```
run native:swapxy INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Swap X and Y coordinates**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:symmetricaldifference — Symmetrical difference ★

group: Vector overlay · **niva verb:** `symdifference`

This algorithm extracts the portions of features from both the Input and Overlay layers that do not overlap. Overlapping areas between the two layers are removed. The attribute table of the Symmetrical Difference layer contains original attributes from both the Input and Overlay layers.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OVERLAY	source	yes	—	Overlay layer
OVERLAY_FIELDS_PREFIX *(adv)*	string	no	—	Overlay fields prefix
OUTPUT	sink	yes	—	Symmetrical difference
GRID_SIZE *(adv)*	number	no	—	Grid size

Outputs: OUTPUT (outputVector)

Example usage

```
run native:symmetricaldifference INPUT=input.gpkg OVERLAY=input.gpkg OUTPUT=output.gpkg
```

This calls **Symmetrical difference**: `INPUT=input.gpkg` — Input layer; `OVERLAY=input.gpkg` — Overlay layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:taperedbuffer — Tapered buffers

group: Vector geometry

This algorithm creates tapered buffers along line geometries, using a specified start and end buffer diameter corresponding to the buffer diameter at the start and end of the linestrings.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
START_WIDTH	number	no	0.0	Start width
END_WIDTH	number	no	1	End width
SEGMENTS	number	no	16	Segments
OUTPUT	sink	yes	—	Buffered

Outputs: OUTPUT (outputVector)

Example usage

```
run native:taperedbuffer INPUT=input.gpkg START_WIDTH=0 END_WIDTH=1 SEGMENTS=16
 ↪  OUTPUT=output.gpkg
```

This calls **Tapered buffers**: `INPUT=input.gpkg` — Input layer; `START_WIDTH=0` — Start width; `END_WIDTH=1` — End width; `SEGMENTS=16` — Segments; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:tilesxyzdirectory — Generate XYZ tiles (Directory)

group: Raster tools

Generates XYZ tiles of map canvas content and saves them as individual images in a directory.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Extent

Parameter	Type	Required	Default	Description
ZOOM_MIN	number	no	12	Minimum zoom
ZOOM_MAX	number	no	12	Maximum zoom
DPI	number	no	96	DPI
BACKGROUND_COLOR	color	no	<QColor: RGBA 0, 0, 0, 0>	Background color
ANTIALIAS	boolean	no	True	Enable antialiasing
TILE_FORMAT	enum	no	0	Tile format — options: 0=PNG, 1=JPG
QUALITY	number	no	75	Quality (JPG only)
METATILESIZE	number	no	4	Metatile size
TILE_WIDTH	number	no	256	Tile width
TILE_HEIGHT	number	no	256	Tile height
TMS_CONVENTION	boolean	no	False	Use inverted tile Y axis (TMS convention)
HTML_TITLE *(adv)*	string	no	—	Leaflet HTML output title
HTML_ATTRIBUTION *(adv)*	string	no	—	Leaflet HTML output attribution
HTML_OSM *(adv)*	boolean	no	False	Include OpenStreetMap basemap in Leaflet HTML output
OUTPUT_DIRECTORY	folderDestination	yes	—	Output directory
OUTPUT_HTML	fileDestination	no	—	Output html (Leaflet)

Outputs: OUTPUT_DIRECTORY (outputFolder), OUTPUT_HTML (outputHtml)

Example usage

```
run native:tilesxyzdirectory EXTENT=0,1000,0,1000 ZOOM_MIN=12 ZOOM_MAX=12 DPI=96
 ↪  BACKGROUND_COLOR="#1f78b4" ANTIALIAS=true TILE_FORMAT=0
 ↪  OUTPUT_DIRECTORY=output_directory/
```

This calls **Generate XYZ tiles (Directory)**: EXTENT=0,1000,0,1000 — Extent; ZOOM_MIN=12 — Minimum zoom; ZOOM_MAX=12 — Maximum zoom; DPI=96 — DPI; BACKGROUND_COLOR="#1f78b4" — Background color; ANTIALIAS=true — Enable antialiasing; TILE_FORMAT=0 selects *PNG*; OUTPUT_DIRECTORY=output_directory/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:tilesxyzmbtiles — Generate XYZ tiles (MBTiles)

group: Raster tools

Generates XYZ tiles of map canvas content and saves them as an MBTiles file.

Parameter	Type	Required	Default	Description
EXTENT	extent	yes	—	Extent
ZOOM_MIN	number	no	12	Minimum zoom
ZOOM_MAX	number	no	12	Maximum zoom
DPI	number	no	96	DPI
BACKGROUND_COLOR	color	no	<QColor: RGBA 0, 0, 0, 0>	Background color
ANTIALIAS	boolean	no	True	Enable antialiasing
TILE_FORMAT	enum	no	0	Tile format — options: 0=PNG, 1=JPG
QUALITY	number	no	75	Quality (JPG only)
METATILESIZE	number	no	4	Metatile size
OUTPUT_FILE	fileDestination	yes	—	Output

Outputs: OUTPUT_FILE (outputFile)

Example usage

```
run native:tilesxyzmbtiles EXTENT=0,1000,0,1000 ZOOM_MIN=12 ZOOM_MAX=12 DPI=96
 ↪  BACKGROUND_COLOR="#1f78b4" ANTIALIAS=true TILE_FORMAT=0 OUTPUT_FILE=output_file.txt
```

This calls **Generate XYZ tiles (MBTiles)**: `EXTENT=0,1000,0,1000` — Extent; `ZOOM_MIN=12` — Minimum zoom; `ZOOM_MAX=12` — Maximum zoom; `DPI=96` — DPI; `BACKGROUND_COLOR="#1f78b4"` — Background color; `ANTIALIAS=true` — Enable antialiasing; `TILE_FORMAT=0` selects *PNG*; `OUTPUT_FILE=output_file.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:tinmeshcreation` — TIN Mesh Creation

group: Mesh

This algorithm creates a TIN mesh layer from vector layers.

Parameter	Type	Required	Default	Description
`SOURCE_DATA`	tininputlayers	yes	—	Input layers
`MESH_FORMAT`	enum	no	0	Output format — options: 0=2DM, 1=SELAFIN, 2=PLY, 3=Ugrid, 4=Mike21
`CRS_OUTPUT`	crs	no	—	Output coordinate system
`OUTPUT_MESH`	fileDestination	yes	—	Output file

Outputs: `OUTPUT_MESH` (outputFile)

Example usage

```
run native:tinmeshcreation SOURCE_DATA=… MESH_FORMAT=0 CRS_OUTPUT=EPSG:6346
↪  OUTPUT_MESH=output_mesh.txt
```

This calls **TIN Mesh Creation**: `SOURCE_DATA=…` — Input layers; `MESH_FORMAT=0` selects *2DM*; `CRS_OUTPUT=EPSG:6346` — Output coordinate system; `OUTPUT_MESH=output_mesh.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:totalcurvature` — Total curvature

group: Raster terrain analysis

This algorithm calculates total curvature of the input raster as described by Wilson, Gallant (2000): terrain analysis.

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Elevation layer
`Z_FACTOR`	number	no	1.0	Z factor
`NODATA *(adv)*`	number	no	-9999.0	Output NoData value
`CREATION_OPTIONS *(adv)*`	string	no	—	Creation options
`OUTPUT`	rasterDestination	yes	—	Total curvature

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run native:totalcurvature INPUT=dem.tif Z_FACTOR=1 OUTPUT=output.tif
```

This calls **Total curvature**: `INPUT=dem.tif` — Elevation layer; `Z_FACTOR=1` — Z factor; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:transect` — Transect

group: Vector geometry

This algorithm creates transects on vertices for (multi)linestrings. A transect is a line oriented from an angle (by default perpendicular) to the input polylines (at vertices).

Field(s) from feature(s) are returned in the transect with these new fields:

- TR_FID: ID of the original feature
- TR_ID: ID of the transect. Each transect have an unique ID
- TR_SEGMENT: ID of the segment of the linestring
- TR_ANGLE: Angle in degrees from the original line at the vertex
- TR_LENGTH: Total length of the transect returned
- TR_ORIENT: Side of the transect (only on the left or right of the line, or both side)

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`LENGTH`	distance	no	5.0	Length of the transect
`ANGLE`	number	no	90.0	Angle in degrees from the original line at the vertices
`SIDE`	enum	no	2	Side to create the transects — options: 0=Left, 1=Right, 2=Both
`DIRECTION`	enum	no	0	Direction — options: 0=Right to Left, 1=Left to Right
`OUTPUT`	sink	yes	—	Transect

Outputs: `OUTPUT` (outputVector)

Example usage

```
run native:transect INPUT=input.gpkg LENGTH=5 ANGLE=90 SIDE=2 DIRECTION=0
 ↪  OUTPUT=output.gpkg
```

This calls **Transect**: `INPUT=input.gpkg` — Input layer; `LENGTH=5` — Length of the transect; `ANGLE=90` — Angle in degrees from the original line at the vertices; `SIDE=2` selects *Both*; `DIRECTION=0` selects *Right to Left*; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:transectfixeddistance` — Transect (fixed distance)

group: Vector geometry

This algorithm creates transects at fixed distance intervals along (multi)linestrings. A transect is a line oriented from an angle (by default perpendicular) to the input polylines at regular intervals.

Field(s) from feature(s) are returned in the transect with these new fields:

- TR_FID: ID of the original feature
- TR_ID: ID of the transect. Each transect have an unique ID
- TR_SEGMENT: ID of the segment of the linestring
- TR_ANGLE: Angle in degrees from the original line at the vertex
- TR_LENGTH: Total length of the transect returned
- TR_ORIENT: Side of the transect (only on the left or right of the line, or both side)

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`INTERVAL`	distance	no	10.0	Fixed sampling interval
`INCLUDE_START`	boolean	no	True	Include transect at start of line
`LENGTH`	distance	no	5.0	Length of the transect
`ANGLE`	number	no	90.0	Angle in degrees from the original line at the vertices
`SIDE`	enum	no	2	Side to create the transects — options: 0=Left, 1=Right, 2=Both

Parameter	Type	Required	Default	Description
DIRECTION	enum	no	0	Direction — options: 0=Right to Left, 1=Left to Right
OUTPUT	sink	yes	—	Transect

Outputs: OUTPUT (outputVector)

Example usage

```
run native:transectfixeddistance INPUT=input.gpkg INTERVAL=10 INCLUDE_START=true LENGTH=5
↪  ANGLE=90 SIDE=2 DIRECTION=0 OUTPUT=output.gpkg
```

This calls **Transect (fixed distance)**: `INPUT=input.gpkg` — Input layer; `INTERVAL=10` — Fixed sampling interval; `INCLUDE_START=true` — Include transect at start of line; `LENGTH=5` — Length of the transect; `ANGLE=90` — Angle in degrees from the original line at the vertices; `SIDE=2` selects *Both*; `DIRECTION=0` selects *Right to Left*; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:transferannotationsfrommain` — Transfer annotations from main layer

group: Cartography

This algorithm transfers all annotations from the main annotation layer in a project to a new annotation layer.

Parameter	Type	Required	Default	Description
LAYER_NAME	string	no	Annotations	New layer name

Outputs: OUTPUT (outputLayer)

Example usage

```
run native:transferannotationsfrommain LAYER_NAME=Annotations
```

This calls **Transfer annotations from main layer**: `LAYER_NAME=Annotations` — New layer name. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:translategeometry` — Translate

group: Vector geometry

This algorithm moves the geometries within a layer, by offsetting them with a specified x and y displacement.

Z and M values present in the geometry can also be translated.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
DELTA_X	distance	no	0.0	Offset distance (x-axis)
DELTA_Y	distance	no	0.0	Offset distance (y-axis)
DELTA_Z	number	no	0.0	Offset distance (z-axis)
DELTA_M	number	no	0.0	Offset distance (m values)
OUTPUT	sink	yes	—	Translated

Outputs: OUTPUT (outputVector)

Example usage

```
run native:translategeometry INPUT=input.gpkg DELTA_X=0 DELTA_Y=0 DELTA_Z=0 DELTA_M=0
↪  OUTPUT=output.gpkg
```

This calls **Translate**: INPUT=input.gpkg — Input layer; DELTA_X=0 — Offset distance (x-axis); DELTA_Y=0 — Offset distance (y-axis); DELTA_Z=0 — Offset distance (z-axis); DELTA_M=0 — Offset distance (m values); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:truncatetable — Truncate table

group: Vector general

This algorithm truncates a layer, by deleting all features from within the layer.

Warning — this algorithm modifies the layer in place, and deleted features cannot be restored!

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer

Outputs: OUTPUT (outputVector)

Example usage

```
run native:truncatetable INPUT=input.gpkg
```

This calls **Truncate table**: INPUT=input.gpkg — Input layer. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:union — Union ★

group: Vector overlay · **niva verb:** union

This algorithm checks overlaps between features within the Input layer and creates separate features for overlapping and non-overlapping parts. The area of overlap will create as many identical overlapping features as there are features that participate in that overlap.

An Overlay layer can also be used, in which case features from each layer are split at their overlap with features from the other one, creating a layer containing all the portions from both Input and Overlay layers. The attribute table of the Union layer is filled with attribute values from the respective original layer for non-overlapping features, and attribute values from both layers for overlapping features.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OVERLAY	source	no	—	Overlay layer
OVERLAY_FIELDS_PREFIX *(adv)*	string	no	—	Overlay fields prefix
OUTPUT	sink	yes	—	Union
GRID_SIZE *(adv)*	number	no	—	Grid size

Outputs: OUTPUT (outputVector)

Example usage

```
run native:union INPUT=input.gpkg OVERLAY=input.gpkg OUTPUT=output.gpkg
```

This calls **Union**: INPUT=input.gpkg — Input layer; OVERLAY=input.gpkg — Overlay layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:updatelayermetadata` — Update layer metadata

group: Metadata tools

This algorithm copies all non-empty metadata fields from a source layer to a target layer.

Leaves empty input fields unchanged in the target.

Parameter	Type	Required	Default	Description
SOURCE	layer	yes	—	Source layer
TARGET	layer	yes	—	Target layer

Outputs: OUTPUT (outputLayer)

Example usage

```
run native:updatelayermetadata SOURCE=input.gpkg TARGET=input.gpkg
```

This calls **Update layer metadata**: `SOURCE=input.gpkg` — Source layer; `TARGET=input.gpkg` — Target layer. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:uploadgpsdata` — Upload GPS data to device

group: GPS

This algorithm uses the GPSBabel tool to upload data to a GPS device from the GPX standard format.

Parameter	Type	Required	Default	Description
INPUT	file	yes	—	Input file
DEVICE	string	yes	—	Device
PORT	string	yes	—	Port
FEATURE_TYPE	enum	no	0	Feature type — options: 0=Waypoints, 1=Routes, 2=Tracks

Example usage

```
run native:uploadgpsdata INPUT=input.dat DEVICE=value PORT=value FEATURE_TYPE=0
```

This calls **Upload GPS data to device**: `INPUT=input.dat` — Input file; `DEVICE=value` — Device; `PORT=value` — Port; `FEATURE_TYPE=0` selects *Waypoints*. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`native:validatenetwork` — Validate network

group: Network analysis

This algorithm analyzes a network vector layer to identify data and topology errors that may affect network analysis tools (like shortest path).

Optional checks include:

1. Validating the 'Direction' field to ensure all direction field values in the input layer match the configured forward/backward/both values. Errors will be reported if the direction field value is non-null and does not match one of the configured values.
2. Checking node-to-node separation. This check identifies nodes from the network graph that are closer to other nodes than the specified tolerance distance. This often indicates missed snaps or short segments in the input layer. In the case that a node violates this condition with multiple other nodes, only the closest violation will be reported.

3. Checking node-to-segment separation: This check identifies nodes that are closer to a line segment (e.g. a graph edge) than the specified tolerance distance, without being connected to it. In the case that a node violates this condition with multiple other edges, only the closest violation will be reported.

Topology checks (node-to-node and node-to-segment) can optionally be restricted to only evaluate nodes that are topological dead-ends (connected to only one other distinct node). This is useful for specifically targeting dangles or undershoots.

Two layers are output by this algorithm:

1. An output containing features from the original network layer which failed the direction validation checks.
2. An output representing the problematic node locations with a 'error' field explaining the error. This is a line layer, where the output features join the problematic node to the node or segment which failed the tolerance checks.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Vector layer representing network
TOLERANCE_NODE_NODE	distance	no	—	Minimum separation between nodes
TOLERANCE_NODE_SEGMENT	distance	no	—	Minimum separation between nodes and non-noded segments
ENDPOINTS_ONLY	boolean	no	False	Only check for errors at end points
DIRECTION_FIELD	field	no	—	Direction field
VALUE_FORWARD	string	no	—	Value for forward direction
VALUE_BACKWARD	string	no	—	Value for backward direction
VALUE_BOTH	string	no	—	Value for both directions
TOLERANCE *(adv)*	distance	no	0	Topology tolerance
OUTPUT_INVALID_NETWORK	sink	no	—	Invalid network features
OUTPUT_INVALID_NODES	sink	no	—	Invalid network nodes

Outputs: OUTPUT_INVALID_NETWORK (outputVector), COUNT_INVALID_NETWORK_FEATURES (outputNumber), OUTPUT_INVALID_NODES (outputVector), COUNT_INVALID_NODES (outputNumber)

Example usage

```
run native:validatenetwork INPUT=input.gpkg TOLERANCE_NODE_NODE=10
 ↪  TOLERANCE_NODE_SEGMENT=10 ENDPOINTS_ONLY=false DIRECTION_FIELD=field1
 ↪  VALUE_FORWARD=value VALUE_BACKWARD=value
 ↪  OUTPUT_INVALID_NETWORK=output_invalid_network.gpkg
```

This calls **Validate network**: INPUT=input.gpkg — Vector layer representing network; TOLERANCE_NODE_NODE=10 — Minimum separation between nodes; TOLERANCE_NODE_SEGMENT=10 — Minimum separation between nodes and non-noded segments; ENDPOINTS_ONLY=false — Only check for errors at end points; DIRECTION_FIELD=field1 — Direction field; VALUE_FORWARD=value — Value for forward direction; VALUE_BACKWARD=value — Value for backward direction; OUTPUT_INVALID_NETWORK=output_invalid_network.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:virtualrastercalc — Raster calculator (virtual)

group: Raster analysis

This algorithm performs algebraic operations using raster layers and generates in-memory result.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layers
EXPRESSION	expression	yes	—	Expression
EXTENT	extent	no	—	Output extent

Parameter	Type	Required	Default	Description
CELL_SIZE	number	no	—	Output cell size (leave empty to set automatically)
CRS	crs	no	—	Output CRS
LAYER_NAME	string	no	—	Output layer name

Outputs: OUTPUT (outputRaster)

Example usage

```
run native:virtualrastercalc LAYERS="a.gpkg;b.gpkg" EXPRESSION="value > 0"
 ↪  EXTENT=0,1000,0,1000 CELL_SIZE=10 CRS=EPSG:6346 LAYER_NAME=value
```

This calls **Raster calculator (virtual)**: LAYERS="a.gpkg;b.gpkg" — Input layers; EXPRESSION="value > 0" — Expression; EXTENT=0,1000,0,1000 — Output extent; CELL_SIZE=10 — Output cell size (leave empty to set automatically); CRS=EPSG:6346 — Output CRS; LAYER_NAME=value — Output layer name. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:voronoipolygons — Voronoi polygons ★

group: Vector geometry · **niva verb:** voronoi

This algorithm generates a polygon layer containing the Voronoi diagram corresponding to input points.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
BUFFER	number	no	0	Buffer region (% of extent)
TOLERANCE	number	no	0	Tolerance
COPY_ATTRIBUTES	boolean	no	True	Copy attributes from input features
OUTPUT	sink	yes	—	Voronoi polygons

Outputs: OUTPUT (outputVector)

Example usage

```
run native:voronoipolygons INPUT=input.gpkg BUFFER=0 TOLERANCE=0 COPY_ATTRIBUTES=true
 ↪  OUTPUT=output.gpkg
```

This calls **Voronoi polygons**: INPUT=input.gpkg — Input layer; BUFFER=0 — Buffer region (% of extent); TOLERANCE=0 — Tolerance; COPY_ATTRIBUTES=true — Copy attributes from input features; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:wedgebuffers — Create wedge buffers

group: Vector geometry

This algorithm creates wedge shaped buffers from input points.

The azimuth parameter gives the angle (in degrees) for the middle of the wedge to point. The buffer width (in degrees) is specified by the width parameter. Note that the wedge will extend to half of the angular width either side of the azimuth direction.

The outer radius of the buffer is specified via outer radius, and optionally an inner radius can also be specified.

The native output from this algorithm are CurvePolygon geometries, but these may be automatically segmentized to Polygons depending on the output format.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
AZIMUTH	number	no	0	Azimuth (degrees from North)
WIDTH	number	no	45	Wedge width (in degrees)
OUTER_RADIUS	number	no	1	Outer radius
INNER_RADIUS	number	no	0	Inner radius
OUTPUT	sink	yes	—	Buffers

Outputs: OUTPUT (outputVector)

Example usage

```
run native:wedgebuffers INPUT=input.gpkg AZIMUTH=0 WIDTH=45 OUTER_RADIUS=1 INNER_RADIUS=0
↪  OUTPUT=output.gpkg
```

This calls **Create wedge buffers**: INPUT=input.gpkg — Input layer; AZIMUTH=0 — Azimuth (degrees from North); WIDTH=45 — Wedge width (in degrees); OUTER_RADIUS=1 — Outer radius; INNER_RADIUS=0 — Inner radius; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:writevectortiles_mbtiles — Write Vector Tiles (MBTiles)

group: Vector tiles

This algorithm exports one or more vector layers to vector tiles - a data format optimized for fast map rendering and small data size.

Parameter	Type	Required	Default	Description
OUTPUT	vectorTileDestination	yes	—	Destination MBTiles
LAYERS	vectortilewriterlayers	yes	—	Input layers
MIN_ZOOM	number	no	0	Minimum zoom level
MAX_ZOOM	number	no	3	Maximum zoom level
EXTENT	extent	no	—	Extent
META_NAME	string	no	—	Metadata: Name
META_DESCRIPTION	string	no	—	Metadata: Description
META_ATTRIBUTION	string	no	—	Metadata: Attribution
META_VERSION	string	no	—	Metadata: Version
META_TYPE	string	no	—	Metadata: Type
META_CENTER	string	no	—	Metadata: Center

Outputs: OUTPUT (outputVectorTile)

Example usage

```
run native:writevectortiles_mbtiles LAYERS=… MIN_ZOOM=0 MAX_ZOOM=3 EXTENT=0,1000,0,1000
↪  META_NAME=value META_DESCRIPTION=value META_ATTRIBUTION=value OUTPUT=output.mbtiles
```

This calls **Write Vector Tiles (MBTiles)**: LAYERS=… — Input layers; MIN_ZOOM=0 — Minimum zoom level; MAX_ZOOM=3 — Maximum zoom level; EXTENT=0,1000,0,1000 — Extent; META_NAME=value — Metadata: Name; META_DESCRIPTION=value — Metadata: Description; META_ATTRIBUTION=value — Metadata: Attribution; OUTPUT=output.mbtiles is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:writevectortiles_xyz — Write Vector Tiles (XYZ)

group: Vector tiles

This algorithm exports one or more vector layers to vector tiles - a data format optimized for fast map rendering and small data size.

Parameter	Type	Required	Default	Description
OUTPUT_DIRECTORY	folderDestination	yes	—	Output directory
XYZ_TEMPLATE	string	no	{z}/{x}/{y}.pbf	File template
LAYERS	vectortilewriterlayers	yes	—	Input layers
MIN_ZOOM	number	no	0	Minimum zoom level
MAX_ZOOM	number	no	3	Maximum zoom level
EXTENT	extent	no	—	Extent

Outputs: OUTPUT_DIRECTORY (outputFolder)

Example usage

```
run native:writevectortiles_xyz LAYERS=… XYZ_TEMPLATE={z}/{x}/{y}.pbf MIN_ZOOM=0
↪  MAX_ZOOM=3 EXTENT=0,1000,0,1000 OUTPUT_DIRECTORY=output_directory/
```

This calls **Write Vector Tiles (XYZ)**: LAYERS=… — Input layers; XYZ_TEMPLATE={z}/{x}/{y}.pbf — File template; MIN_ZOOM=0 — Minimum zoom level; MAX_ZOOM=3 — Maximum zoom level; EXTENT=0,1000,0,1000 — Extent; OUTPUT_DIRECTORY=output_directory/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:zonalhistogram — Zonal histogram

group: Raster analysis

This algorithm appends fields representing counts of each unique value from a raster layer contained within zones defined as polygons.

Parameter	Type	Required	Default	Description
INPUT_RASTER	raster	yes	—	Raster layer
RASTER_BAND	band	no	1	Band number
INPUT_VECTOR	source	yes	—	Vector layer containing zones
COLUMN_PREFIX	string	no	HISTO_	Output column prefix
OUTPUT	sink	yes	—	Output zones

Outputs: OUTPUT (outputVector)

Example usage

```
run native:zonalhistogram INPUT_RASTER=dem.tif INPUT_VECTOR=input.gpkg RASTER_BAND=1
↪  COLUMN_PREFIX=HISTO_ OUTPUT=output.gpkg
```

This calls **Zonal histogram**: INPUT_RASTER=dem.tif — Raster layer; INPUT_VECTOR=input.gpkg — Vector layer containing zones; RASTER_BAND=1 — Band number; COLUMN_PREFIX=HISTO_ — Output column prefix; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:zonalminmaxpoint — Zonal minimum/maximum point

group: Raster analysis

This algorithm extracts point features corresponding to the minimum and maximum pixel values contained within polygon zones.

The output will contain one point feature for the minimum and one for the maximum raster value for every individual zonal feature from a polygon layer.

The created point layer will be in the same spatial reference system as the selected raster layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
INPUT_RASTER	raster	yes	—	Raster layer
RASTER_BAND	band	no	1	Raster band
OUTPUT	sink	yes	—	Zonal Extrema

Outputs: OUTPUT (outputVector)

Example usage

```
run native:zonalminmaxpoint INPUT=input.gpkg INPUT_RASTER=dem.tif RASTER_BAND=1
↪  OUTPUT=output.gpkg
```

This calls **Zonal minimum/maximum point**: INPUT=input.gpkg — Input layer; INPUT_RASTER=dem.tif — Raster layer; RASTER_BAND=1 — Raster band; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:zonalstatistics — Zonal statistics (in place)

group: Raster analysis

This algorithm calculates statistics of a raster layer for each feature of an overlapping polygon vector layer. The results will be written in place.

Parameter	Type	Required	Default	Description
INPUT_RASTER	raster	yes	—	Raster layer
RASTER_BAND	band	no	1	Raster band
INPUT_VECTOR	vector	yes	—	Vector layer containing zones
COLUMN_PREFIX	string	no	_	Output column prefix
STATISTICS	enum	no	[0, 1, 2]	Statistics to calculate — options: 0=Count, 1=Sum, 2=Mean, 3=Median, 4=St dev, 5=Minimum, 6=Maximum, 7=Range, 8=Minority, 9=Majority, 10=Variety, 11=Variance

Outputs: INPUT_VECTOR (outputVector)

Example usage

```
run native:zonalstatistics INPUT_RASTER=dem.tif INPUT_VECTOR=input.gpkg RASTER_BAND=1
↪  COLUMN_PREFIX=_ STATISTICS=0
```

This calls **Zonal statistics (in place)**: INPUT_RASTER=dem.tif — Raster layer; INPUT_VECTOR=input.gpkg — Vector layer containing zones; RASTER_BAND=1 — Raster band; COLUMN_PREFIX=_ — Output column prefix; STATISTICS=0 selects *Count*. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

native:zonalstatisticsfb — Zonal statistics ★

group: Raster analysis · **niva verb:** zonalstats

This algorithm calculates statistics of a raster layer for each feature of an overlapping polygon vector layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
INPUT_RASTER	raster	yes	—	Raster layer
RASTER_BAND	band	no	1	Raster band
COLUMN_PREFIX	string	no	_	Output column prefix

Parameter	Type	Required	Default	Description
STATISTICS	enum	no	[0, 1, 2]	Statistics to calculate — options: 0=Count, 1=Sum, 2=Mean, 3=Median, 4=St dev, 5=Minimum, 6=Maximum, 7=Range, 8=Minority, 9=Majority, 10=Variety, 11=Variance
OUTPUT	sink	yes	—	Zonal Statistics

Outputs: OUTPUT (outputVector)

Example usage

```
run native:zonalstatisticsfb INPUT=input.gpkg INPUT_RASTER=dem.tif RASTER_BAND=1
↪  COLUMN_PREFIX=_ STATISTICS=0 OUTPUT=output.gpkg
```

This calls **Zonal statistics**: INPUT=input.gpkg — Input layer; INPUT_RASTER=dem.tif — Raster layer; RASTER_BAND=1 — Raster band; COLUMN_PREFIX=_ — Output column prefix; STATISTICS=0 selects *Count*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal: algorithms

gdal — GDAL/OGR algorithms — 59 algorithms (QGIS 4.0.3). Auto-generated by `scripts/gen_algorithms.py`; see the index.

Call any of these with `run <id> KEY=value …` (the **Name** column is the KEY); each entry includes a worked **Example usage**. A ★ marks an algorithm with a friendly niva alias verb.

gdal:aspect — Aspect ★

group: Raster analysis · **niva verb:** `aspect`

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
TRIG_ANGLE	boolean	no	False	Return trigonometric angle instead of azimuth
ZERO_FLAT	boolean	no	False	Return 0 for flat instead of -9999
COMPUTE_EDGES	boolean	no	False	Compute edges
ZEVENBERGEN	boolean	no	False	Use Zevenbergen&Thorne formula instead of the Horn's one
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Aspect

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:aspect INPUT=dem.tif BAND=1 TRIG_ANGLE=false ZERO_FLAT=false COMPUTE_EDGES=false
↪  ZEVENBERGEN=false OPTIONS=value OUTPUT=output.tif
```

This calls **Aspect**: `INPUT=dem.tif` — Input layer; `BAND=1` — Band number; `TRIG_ANGLE=false` — Return trigonometric angle instead of azimuth; `ZERO_FLAT=false` — Return 0 for flat instead of -9999; `COMPUTE_EDGES=false` — Compute edges; `ZEVENBERGEN=false` — Use Zevenbergen&Thorne formula instead of the Horn's one; `OPTIONS=value` — Additional creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:assignprojection — Assign projection

group: Raster projections

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
CRS	crs	yes	—	Desired CRS

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:assignprojection INPUT=dem.tif CRS=EPSG:6346
```

This calls **Assign projection**: `INPUT=dem.tif` — Input layer; `CRS=EPSG:6346` — Desired CRS. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:buffervectors` — Buffer vectors

group: Vector geoprocessing

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`GEOMETRY`	string	no	geometry	Geometry column name
`DISTANCE`	distance	no	10.0	Buffer distance
`FIELD`	field	no	—	Dissolve by attribute
`DISSOLVE`	boolean	no	False	Dissolve all results
`EXPLODE_COLLEC-TIONS`	boolean	no	False	Produce one feature for each geometry in any kind of geometry collection in the source file
`OPTIONS *(adv)*`	string	no	—	Additional creation options
`OUTPUT`	vectorDestination	yes	—	Buffer

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:buffervectors INPUT=input.gpkg GEOMETRY=geometry DISTANCE=10 FIELD=field1
↪  DISSOLVE=false EXPLODE_COLLECTIONS=false OUTPUT=output.gpkg
```

This calls **Buffer vectors**: `INPUT=input.gpkg` — Input layer; `GEOMETRY=geometry` — Geometry column name; `DISTANCE=10` — Buffer distance; `FIELD=field1` — Dissolve by attribute; `DISSOLVE=false` — Dissolve all results; `EXPLODE_COLLECTIONS=false` — Produce one feature for each geometry in any kind of geometry collection in the…; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:buildvirtualraster` — Build virtual raster

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
`INPUT`	multilayer	yes	—	Input layers
`RESOLUTION`	enum	no	0	Resolution — options: 0=Average, 1=Highest, 2=Lowest
`SEPARATE`	boolean	no	True	Place each input file into a separate band
`PROJ_DIFFERENCE`	boolean	no	False	Allow projection difference
`ADD_ALPHA *(adv)*`	boolean	no	False	Add alpha mask band to VRT when source raster has none
`ASSIGN_CRS *(adv)*`	crs	no	—	Override projection for the output file
`RESAMPLING *(adv)*`	enum	no	0	Resampling algorithm — options: 0=Nearest Neighbour, 1=Bilinear (2x2 Kernel), 2=Cubic (4x4 Kernel), 3=Cubic B-Spline (4x4 Kernel), 4=Lanczos (6x6 Kernel), 5=Average, 6=Mode
`SRC_NODATA *(adv)*`	string	no	—	Nodata value(s) for input bands (space separated)
`EXTRA *(adv)*`	string	no	—	Additional command-line parameters

Parameter	Type	Required	Default	Description
OUTPUT	rasterDestination	yes	—	Virtual

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:buildvirtualraster INPUT="a.gpkg;b.gpkg" RESOLUTION=0 SEPARATE=true
 ↪  PROJ_DIFFERENCE=false OUTPUT=output.tif
```

This calls **Build virtual raster**: INPUT="a.gpkg;b.gpkg" — Input layers; RESOLUTION=0 selects *Average*; SEPARATE=true — Place each input file into a separate band; PROJ_DIFFERENCE=false — Allow projection difference; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:buildvirtualvector — Build virtual vector

group: Vector miscellaneous

This algorithm creates a virtual layer that contains a set of vector layers.

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input datasources
UNIONED	boolean	no	False	Create ”unioned” VRT
OUTPUT	vectorDestination	yes	—	Virtual vector

Outputs: OUTPUT (outputVector), VRT_STRING (outputString)

Example usage

```
run gdal:buildvirtualvector INPUT="a.gpkg;b.gpkg" UNIONED=false OUTPUT=output.gpkg
```

This calls **Build virtual vector**: INPUT="a.gpkg;b.gpkg" — Input datasources; UNIONED=false — Create ”unioned” VRT; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:cliprasterbyextent — Clip raster by extent

group: Raster extraction

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
PROJWIN	extent	yes	—	Clipping extent
OVERCRS	boolean	no	False	Override the projection for the output file
NODATA	number	no	—	Assign a specified NoData value to output bands
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
DATA_TYPE *(adv)*	enum	no	0	Output data type — options: 0=Use Input Layer Data Type, 1=Byte, 2=Int16, 3=UInt16, 4=UInt32, 5=Int32, 6=Float32, 7=Float64, 8=CInt16, 9=CInt32, 10=CFloat32, 11=CFloat64, 12=Int8
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Clipped (extent)

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:cliprasterbyextent INPUT=dem.tif PROJWIN=0,1000,0,1000 OVERCRS=false NODATA=10
↪ OPTIONS=value OUTPUT=output.tif
```

This calls **Clip raster by extent**: `INPUT=dem.tif` — Input layer; `PROJWIN=0,1000,0,1000` — Clipping extent; `OVERCRS=false` — Override the projection for the output file; `NODATA=10` — Assign a specified NoData value to output bands; `OPTIONS=value` — Additional creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:cliprasterbymasklayer` — Clip raster by mask layer ★

group: Raster extraction · **niva verb:** `clipraster`

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input layer
`MASK`	source	yes	—	Mask layer
`SOURCE_CRS`	crs	no	—	Source CRS
`TARGET_CRS`	crs	no	—	Target CRS
`TARGET_EXTENT`	extent	no	—	Target extent
`NODATA`	number	no	—	Assign a specified NoData value to output bands
`ALPHA_BAND`	boolean	no	False	Create an output alpha band
`CROP_TO_CUTLINE`	boolean	no	True	Match the extent of the clipped raster to the extent of the mask layer
`KEEP_RESOLUTION`	boolean	no	False	Keep resolution of input raster
`SET_RESOLUTION`	boolean	no	False	Set output file resolution
`X_RESOLUTION`	number	no	—	X Resolution to output bands
`Y_RESOLUTION`	number	no	—	Y Resolution to output bands
`MULTITHREADING *(adv)*`	boolean	no	False	Use multithreaded warping implementation
`OPTIONS`	string	no	—	Additional creation options
`CREATION_OPTIONS *(adv)*`	string	no	—	Additional creation options
`DATA_TYPE *(adv)*`	enum	no	0	Output data type — options: 0=Use Input Layer Data Type, 1=Byte, 2=Int16, 3=UInt16, 4=UInt32, 5=Int32, 6=Float32, 7=Float64, 8=CInt16, 9=CInt32, 10=CFloat32, 11=CFloat64, 12=Int8
`EXTRA *(adv)*`	string	no	—	Additional command-line parameters
`OUTPUT`	rasterDestination	yes	—	Clipped (mask)

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run gdal:cliprasterbymasklayer INPUT=dem.tif MASK=input.gpkg SOURCE_CRS=EPSG:6346
↪ TARGET_CRS=EPSG:6346 TARGET_EXTENT=0,1000,0,1000 NODATA=10 ALPHA_BAND=false
↪ OUTPUT=output.tif
```

This calls **Clip raster by mask layer**: `INPUT=dem.tif` — Input layer; `MASK=input.gpkg` — Mask layer; `SOURCE_CRS=EPSG:6346` — Source CRS; `TARGET_CRS=EPSG:6346` — Target CRS; `TARGET_EXTENT=0,1000,0,1000` — Target extent; `NODATA=10` — Assign a specified NoData value to output bands; `ALPHA_BAND=false` — Create an output alpha band; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:clipvectorbyextent` — Clip vector by extent

group: Vector geoprocessing

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
EXTENT	extent	yes	—	Clipping extent
OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	vectorDestination	yes	—	Clipped (extent)

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:clipvectorbyextent INPUT=input.gpkg EXTENT=0,1000,0,1000 OUTPUT=output.gpkg
```

This calls **Clip vector by extent**: INPUT=input.gpkg — Input layer; EXTENT=0,1000,0,1000 — Clipping extent; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:clipvectorbypolygon — Clip vector by mask layer

group: Vector geoprocessing

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
MASK	source	yes	—	Mask layer
OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	vectorDestination	yes	—	Clipped (mask)

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:clipvectorbypolygon INPUT=input.gpkg MASK=input.gpkg OUTPUT=output.gpkg
```

This calls **Clip vector by mask layer**: INPUT=input.gpkg — Input layer; MASK=input.gpkg — Mask layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:colorrelief — Color relief

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
COMPUTE_EDGES	boolean	no	False	Compute edges
COLOR_TABLE	file	yes	—	Color configuration file
MATCH_MODE	enum	no	2	Matching mode — options: 0=Use strict color matching, 1=Use closest RGBA quadruplet, 2=Use smoothly blended colors
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Color relief

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:colorrelief INPUT=dem.tif COLOR_TABLE=input.dat BAND=1 COMPUTE_EDGES=false
↪ MATCH_MODE=2 OPTIONS=value OUTPUT=output.tif
```

This calls **Color relief**: `INPUT=dem.tif` — Input layer; `COLOR_TABLE=input.dat` — Color configuration file; `BAND=1` — Band number; `COMPUTE_EDGES=false` — Compute edges; `MATCH_MODE=2` selects *Use smoothly blended colors*; `OPTIONS=value` — Additional creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:contour` — Contour

group: Raster extraction

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input layer
`BAND`	band	no	1	Band number
`INTERVAL`	number	no	10.0	Interval between contour lines
`FIELD_NAME`	string	no	ELEV	Attribute name (if not set, no elevation attribute is attached)
`CREATE_3D *(adv)*`	boolean	no	False	Produce 3D vector
`IGNORE_NODATA *(adv)*`	boolean	no	False	Treat all raster values as valid
`NODATA *(adv)*`	number	no	—	Input pixel value to treat as NoData
`OFFSET`	number	no	0.0	Offset from zero relative to which to interpret intervals
`EXTRA *(adv)*`	string	no	—	Additional command-line parameters
`OPTIONS`	string	no	—	Additional creation options
`OUTPUT`	vectorDestination	yes	—	Contours

Outputs: `OUTPUT` (outputVector)

Example usage

```
run gdal:contour INPUT=dem.tif BAND=1 INTERVAL=10 FIELD_NAME=ELEV OFFSET=0 OPTIONS=value
↪ OUTPUT=output.gpkg
```

This calls **Contour**: `INPUT=dem.tif` — Input layer; `BAND=1` — Band number; `INTERVAL=10` — Interval between contour lines; `FIELD_NAME=ELEV` — Attribute name (if not set, no elevation attribute is attached); `OFFSET=0` — Offset from zero relative to which to interpret intervals; `OPTIONS=value` — Additional creation options; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:contour_polygon` — Contour Polygons

group: Raster extraction

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input layer
`BAND`	band	no	1	Band number
`INTERVAL`	number	no	10.0	Interval between contour lines
`CREATE_3D *(adv)*`	boolean	no	False	Produce 3D vector
`IGNORE_NODATA *(adv)*`	boolean	no	False	Treat all raster values as valid
`NODATA *(adv)*`	number	no	—	Input pixel value to treat as NoData
`OFFSET`	number	no	0.0	Offset from zero relative to which to interpret intervals
`EXTRA *(adv)*`	string	no	—	Additional command-line parameters
`OPTIONS`	string	no	—	Additional creation options

Parameter	Type	Required	Default	Description
FIELD_NAME_MIN	string	no	ELEV_-MIN	Attribute name for minimum elevation of contour polygon
FIELD_NAME_MAX	string	no	ELEV_-MAX	Attribute name for maximum elevation of contour polygon
OUTPUT	vectorDestination	yes	—	Contours

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:contour_polygon INPUT=dem.tif BAND=1 INTERVAL=10 OFFSET=0 OPTIONS=value
 ↪  FIELD_NAME_MIN=ELEV_MIN FIELD_NAME_MAX=ELEV_MAX OUTPUT=output.gpkg
```

This calls **Contour Polygons**: INPUT=dem.tif — Input layer; BAND=1 — Band number; INTERVAL=10 — Interval between contour lines; OFFSET=0 — Offset from zero relative to which to interpret intervals; OPTIONS=value — Additional creation options; FIELD_NAME_MIN=ELEV_MIN — Attribute name for minimum elevation of contour polygon; FIELD_NAME_MAX=ELEV_MAX — Attribute name for maximum elevation of contour polygon; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:convertformat — Convert format

group: Vector conversion

The algorithm converts simple features data between file formats.

Use convert all layers to convert a whole dataset. Supported output formats for this option are:

- GPKG
- GML

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
CONVERT_ALL_LAYERS	boolean	no	False	Convert all layers from dataset
OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	vectorDestination	yes	—	Converted

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:convertformat INPUT=input.gpkg CONVERT_ALL_LAYERS=false OUTPUT=output.gpkg
```

This calls **Convert format**: INPUT=input.gpkg — Input layer; CONVERT_ALL_LAYERS=false — Convert all layers from dataset; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:createcog — Create Cloud Optimized GeoTIFF

group: Raster conversion

This algorithm creates a Cloud Optimized GeoTIFF (COG) from the input raster layers.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layers
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	folderDestination	yes	—	Output directory

Outputs: OUTPUT (outputFolder), OUTPUT_LAYERS (outputMultilayer)

Example usage

```
run gdal:createcog LAYERS="a.gpkg;b.gpkg" OUTPUT=output/
```

This calls **Create Cloud Optimized GeoTIFF**: LAYERS="a.gpkg;b.gpkg" — Input layers; OUTPUT=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:dataset_identify — Dataset identification

group: Miscellaneous

This algorithm reports the name of GDAL drivers that can open files contained in a folder, with optional additional details, and write the result into an output vector layer.

Parameter	Type	Required	Default	Description
INPUT	file	yes	—	Input folder
RECURSIVE	boolean	no	True	Perform recursive exploration of the input folder
DETAILS	boolean	no	True	Add details about identified datasets in the output
OUTPUT	vectorDestination	yes	—	Output file

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:dataset_identify INPUT=input.dat RECURSIVE=true DETAILS=true OUTPUT=output.gpkg
```

This calls **Dataset identification**: INPUT=input.dat — Input folder; RECURSIVE=true — Perform recursive exploration of the input folder; DETAILS=true — Add details about identified datasets in the output; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:dissolve — Dissolve

group: Vector geoprocessing

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	no	—	Dissolve field
GEOMETRY	string	no	geometry	Geometry column name
EXPLODE_COLLECTIONS *(adv)*	boolean	no	False	Produce one feature for each geometry in any kind of geometry collection in the source file
KEEP_ATTRIBUTES *(adv)*	boolean	no	False	Keep input attributes
COUNT_FEATURES *(adv)*	boolean	no	False	Count dissolved features
COMPUTE_AREA *(adv)*	boolean	no	False	Compute area and perimeter of dissolved features
COMPUTE_STATISTICS *(adv)*	boolean	no	False	Compute min/max/sum/mean for attribute
STATISTICS_ATTRIBUTE *(adv)*	field	no	—	Numeric attribute to calculate statistics on
OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	vectorDestination	yes	—	Dissolved

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:dissolve INPUT=input.gpkg FIELD=field1 GEOMETRY=geometry OUTPUT=output.gpkg
```

This calls **Dissolve**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — Dissolve field; `GEOMETRY=geometry` — Geometry column name; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:executesql` — Execute SQL

group: Vector miscellaneous

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`SQL`	string	yes	—	SQL expression
`DIALECT`	enum	no	0	SQL dialect — options: 0=None, 1=OGR SQL, 2=SQLite
`OPTIONS *(adv)*`	string	no	—	Additional creation options
`OUTPUT`	vectorDestination	yes	—	SQL result

Outputs: `OUTPUT` (outputVector)

Example usage

```
run gdal:executesql INPUT=input.gpkg SQL=value DIALECT=0 OUTPUT=output.gpkg
```

This calls **Execute SQL**: `INPUT=input.gpkg` — Input layer; `SQL=value` — SQL expression; `DIALECT=0` selects *None*; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:extractprojection` — Extract projection

group: Raster projections

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input file
`PRJ_FILE_CREATE`	boolean	no	False	Create also .prj file

Outputs: `WORLD_FILE` (outputFile), `PRJ_FILE` (outputFile)

Example usage

```
run gdal:extractprojection INPUT=dem.tif PRJ_FILE_CREATE=false
```

This calls **Extract projection**: `INPUT=dem.tif` — Input file; `PRJ_FILE_CREATE=false` — Create also .prj file. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:fillnodata` — Fill NoData

group: Raster analysis

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input layer
`BAND`	band	no	1	Band number
`DISTANCE`	number	no	10	Maximum distance (in pixels) to search out for values to interpolate
`ITERATIONS`	number	no	0	Number of smoothing iterations to run after the interpolation
`NO_MASK`	boolean	no	False	Do not use the default validity mask for the input band

Parameter	Type	Required	Default	Description
MASK_LAYER	raster	no	—	Validity mask
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Filled

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:fillnodata INPUT=dem.tif BAND=1 DISTANCE=10 ITERATIONS=0 NO_MASK=false
 ↪  MASK_LAYER=dem.tif OPTIONS=value OUTPUT=output.tif
```

This calls **Fill NoData**: INPUT=dem.tif — Input layer; BAND=1 — Band number; DISTANCE=10 — Maximum distance (in pixels) to search out for values to interpolate; ITERATIONS=0 — Number of smoothing iterations to run after the interpolation; NO_MASK=false — Do not use the default validity mask for the input band; MASK_LAYER=dem.tif — Validity mask; OPTIONS=value — Additional creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:gdal2tiles — gdal2tiles

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
PROFILE	enum	no	0	Tile cutting profile — options: 0=Mercator, 1=Geodetic, 2=Raster
ZOOM	string	no	—	Zoom levels to render
VIEWER	enum	no	0	Web viewer to generate — options: 0=All, 1=GoogleMaps, 2=OpenLayers, 3=Leaflet, 4=None
TITLE	string	no	—	Title of the map
COPYRIGHT	string	no	—	Copyright of the map
RESAMPLING *(adv)*	enum	no	0	Resampling method — options: 0=Average, 1=Nearest Neighbour, 2=Bilinear (2x2 Kernel), 3=Cubic (4x4 Kernel), 4=Cubic B-Spline (4x4 Kernel), 5=Lanczos (6x6 Kernel), 6=Antialias
SOURCE_CRS *(adv)*	crs	no	—	The spatial reference system used for the source input data
NODATA *(adv)*	number	no	0	Transparency value to assign to the input data
URL *(adv)*	string	no	—	URL address where the generated tiles are going to be published
GOOGLE_KEY *(adv)*	string	no	—	Google Maps API key (http://code.google.com/apis/maps/signup.html)
BING_KEY *(adv)*	string	no	—	Bing Maps API key (https://www.bingmapsportal.com/)
RESUME *(adv)*	boolean	no	False	Generate only missing files
KML *(adv)*	boolean	no	False	Generate KML for Google Earth
NO_KML *(adv)*	boolean	no	False	Avoid automatic generation of KML files for EPSG:4326
OUTPUT	folderDestination	yes	—	Output directory

Outputs: OUTPUT (outputFolder)

Example usage

```
run gdal:gdal2tiles INPUT=dem.tif PROFILE=0 ZOOM=value VIEWER=0 TITLE=value
 ↪  COPYRIGHT=value OUTPUT=output/
```

This calls **gdal2tiles**: INPUT=dem.tif — Input layer; PROFILE=0 selects *Mercator*; ZOOM=value — Zoom levels to render; VIEWER=0 selects *All*; TITLE=value — Title of the map; COPYRIGHT=value — Copyright of the map; OUTPUT=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:gdal2xyz — gdal2xyz

group: Raster conversion

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
NODATA_INPUT	number	no	—	Input pixel value to treat as NoData
NODATA_OUTPUT	number	no	—	Assign specified NoData value to output
SKIP_NODATA	boolean	no	False	Do not output NoData values
CSV	boolean	no	False	Output comma-separated values
OUTPUT	fileDestination	yes	—	XYZ ASCII file

Outputs: OUTPUT (outputFile)

Example usage

```
run gdal:gdal2xyz INPUT=dem.tif BAND=1 NODATA_INPUT=10 NODATA_OUTPUT=10 SKIP_NODATA=false
↪  CSV=false OUTPUT=output.txt
```

This calls **gdal2xyz**: INPUT=dem.tif — Input layer; BAND=1 — Band number; NODATA_INPUT=10 — Input pixel value to treat as NoData; NODATA_OUTPUT=10 — Assign specified NoData value to output; SKIP_NODATA=false — Do not output NoData values; CSV=false — Output comma-separated values; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:gdalinfo — Raster information

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
MIN_MAX	boolean	no	False	Force computation of the actual min/max values for each band
STATS	boolean	no	False	Read and display image statistics (force computation if necessary)
NOGCP	boolean	no	False	Suppress GCP info
NO_METADATA	boolean	no	False	Suppress metadata info
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	fileDestination	yes	—	Layer information

Outputs: OUTPUT (outputHtml)

Example usage

```
run gdal:gdalinfo INPUT=dem.tif MIN_MAX=false STATS=false NOGCP=false NO_METADATA=false
↪  OUTPUT=output.txt
```

This calls **Raster information**: INPUT=dem.tif — Input layer; MIN_MAX=false — Force computation of the actual min/max values for each band; STATS=false — Read and display image statistics (force computation if necessary); NOGCP=false — Suppress GCP info; NO_METADATA=false — Suppress metadata info; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:gridaverage — Grid (Moving average)

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Point layer
Z_FIELD *(adv)*	field	no	—	Z value from field
RADIUS_1	number	no	0.0	The first radius of search ellipse
RADIUS_2	number	no	0.0	The second radius of search ellipse
ANGLE	number	no	0.0	Angle of search ellipse rotation in degrees (counter clockwise)
MIN_POINTS	number	no	0	Minimum number of data points to use
NODATA	number	no	0.0	NODATA marker to fill empty points
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
OUTPUT	rasterDestination	yes	—	Interpolated (moving average)

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:gridaverage INPUT=input.gpkg RADIUS_1=0 RADIUS_2=0 ANGLE=0 MIN_POINTS=0 NODATA=0
↪  OPTIONS=value OUTPUT=output.tif
```

This calls **Grid (Moving average)**: `INPUT=input.gpkg` — Point layer; `RADIUS_1=0` — The first radius of search ellipse; `RADIUS_2=0` — The second radius of search ellipse; `ANGLE=0` — Angle of search ellipse rotation in degrees (counter clockwise); `MIN_POINTS=0` — Minimum number of data points to use; `NODATA=0` — NODATA marker to fill empty points; `OPTIONS=value` — Additional creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:griddatametrics — Grid (Data metrics)

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Point layer
Z_FIELD *(adv)*	field	no	—	Z value from field
METRIC	enum	no	0	Data metric to use — options: 0=Minimum, 1=Maximum, 2=Range, 3=Count, 4=Average distance, 5=Average distance between points
RADIUS_1	number	no	0.0	The first radius of search ellipse
RADIUS_2	number	no	0.0	The second radius of search ellipse
ANGLE	number	no	0.0	Angle of search ellipse rotation in degrees (counter clockwise)
MIN_POINTS	number	no	0	Minimum number of data points to use
NODATA	number	no	0.0	NODATA marker to fill empty points
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters

Parameter	Type	Required	Default	Description
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
OUTPUT	rasterDestination	yes	—	Interpolated (data metrics)

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:griddatametrics INPUT=input.gpkg METRIC=0 RADIUS_1=0 RADIUS_2=0 ANGLE=0
↪  MIN_POINTS=0 NODATA=0 OUTPUT=output.tif
```

This calls **Grid (Data metrics)**: INPUT=input.gpkg — Point layer; METRIC=0 selects *Minimum*; RADIUS_1=0 — The first radius of search ellipse; RADIUS_2=0 — The second radius of search ellipse; ANGLE=0 — Angle of search ellipse rotation in degrees (counter clockwise); MIN_POINTS=0 — Minimum number of data points to use; NODATA=0 — NODATA marker to fill empty points; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:gridinversedistance — Grid (Inverse distance to a power)

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Point layer
Z_FIELD *(adv)*	field	no	—	Z value from field
POWER	number	no	2.0	Weighting power
SMOOTHING	number	no	0.0	Smoothing
RADIUS_1	number	no	0.0	The first radius of search ellipse
RADIUS_2	number	no	0.0	The second radius of search ellipse
ANGLE	number	no	0.0	Angle of search ellipse rotation in degrees (counter clockwise)
MAX_POINTS	number	no	0	Maximum number of data points to use
MIN_POINTS	number	no	0	Minimum number of data points to use
NODATA	number	no	0.0	NODATA marker to fill empty points
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
OUTPUT	rasterDestination	yes	—	Interpolated (IDW)

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:gridinversedistance INPUT=input.gpkg POWER=2 SMOOTHING=0 RADIUS_1=0 RADIUS_2=0
↪  ANGLE=0 MAX_POINTS=0 OUTPUT=output.tif
```

This calls **Grid (Inverse distance to a power)**: INPUT=input.gpkg — Point layer; POWER=2 — Weighting power; SMOOTHING=0 — Smoothing; RADIUS_1=0 — The first radius of search ellipse; RADIUS_2=0 — The second radius of search ellipse; ANGLE=0 — Angle of search ellipse rotation in degrees (counter clockwise); MAX_POINTS=0 — Maximum number of data points to use; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:gridinversedistancenearestneighbor — Grid (IDW with nearest neighbor searching)

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Point layer
Z_FIELD *(adv)*	field	no	—	Z value from field
POWER	number	no	2.0	Weighting power
SMOOTHING	number	no	0.0	Smoothing
RADIUS	number	no	1.0	The radius of the search circle
MAX_POINTS	number	no	12	Maximum number of data points to use
MIN_POINTS	number	no	0	Minimum number of data points to use
NODATA	number	no	0.0	NODATA marker to fill empty points
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
OUTPUT	rasterDestination	yes	—	Interpolated (IDW with NN search)

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:gridinversedistancenearestneighbor INPUT=input.gpkg POWER=2 SMOOTHING=0 RADIUS=1
↪  MAX_POINTS=12 MIN_POINTS=0 NODATA=0 OUTPUT=output.tif
```

This calls **Grid (IDW with nearest neighbor searching)**: INPUT=input.gpkg — Point layer; POWER=2 — Weighting power; SMOOTHING=0 — Smoothing; RADIUS=1 — The radius of the search circle; MAX_POINTS=12 — Maximum number of data points to use; MIN_POINTS=0 — Minimum number of data points to use; NODATA=0 — NODATA marker to fill empty points; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:gridlinear — Grid (Linear)

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Point layer
Z_FIELD *(adv)*	field	no	—	Z value from field
RADIUS	number	no	-1.0	Search distance
NODATA	number	no	0.0	NODATA marker to fill empty points
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
OUTPUT	rasterDestination	yes	—	Interpolated (Linear)

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:gridlinear INPUT=input.gpkg RADIUS=-1 NODATA=0 OPTIONS=value OUTPUT=output.tif
```

This calls **Grid (Linear)**: `INPUT=input.gpkg` — Point layer; `RADIUS=-1` — Search distance; `NODATA=0` — NODATA marker to fill empty points; `OPTIONS=value` — Additional creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:gridnearestneighbor` — Grid (Nearest neighbor)

group: Raster analysis

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Point layer
`Z_FIELD *(adv)*`	field	no	—	Z value from field
`RADIUS_1`	number	no	0.0	The first radius of search ellipse
`RADIUS_2`	number	no	0.0	The second radius of search ellipse
`ANGLE`	number	no	0.0	Angle of search ellipse rotation in degrees (counter clockwise)
`NODATA`	number	no	0.0	NODATA marker to fill empty points
`OPTIONS`	string	no	—	Additional creation options
`CREATION_OPTIONS *(adv)*`	string	no	—	Additional creation options
`EXTRA *(adv)*`	string	no	—	Additional command-line parameters
`DATA_TYPE *(adv)*`	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
`OUTPUT`	rasterDestination	yes	—	Interpolated (Nearest neighbor)

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run gdal:gridnearestneighbor INPUT=input.gpkg RADIUS_1=0 RADIUS_2=0 ANGLE=0 NODATA=0
↪  OPTIONS=value OUTPUT=output.tif
```

This calls **Grid (Nearest neighbor)**: `INPUT=input.gpkg` — Point layer; `RADIUS_1=0` — The first radius of search ellipse; `RADIUS_2=0` — The second radius of search ellipse; `ANGLE=0` — Angle of search ellipse rotation in degrees (counter clockwise); `NODATA=0` — NODATA marker to fill empty points; `OPTIONS=value` — Additional creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:hillshade` — Hillshade

group: Raster analysis

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input layer
`BAND`	band	no	1	Band number
`Z_FACTOR`	number	no	1.0	Z factor (vertical exaggeration)
`SCALE`	number	no	1.0	Scale (ratio of vertical units to horizontal)
`AZIMUTH`	number	no	315.0	Azimuth of the light
`ALTITUDE`	number	no	45.0	Altitude of the light
`COMPUTE_EDGES`	boolean	no	False	Compute edges
`ZEVENBERGEN`	boolean	no	False	Use Zevenbergen&Thorne formula instead of the Horn's one
`COMBINED`	boolean	no	False	Combined shading
`MULTIDIRECTIONAL`	boolean	no	False	Multidirectional shading

Parameter	Type	Required	Default	Description
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Hillshade

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:hillshade INPUT=dem.tif BAND=1 Z_FACTOR=1 SCALE=1 AZIMUTH=315 ALTITUDE=45
 ↪  COMPUTE_EDGES=false OUTPUT=output.tif
```

This calls **Hillshade**: INPUT=dem.tif — Input layer; BAND=1 — Band number; Z_FACTOR=1 — Z factor (vertical exaggeration); SCALE=1 — Scale (ratio of vertical units to horizontal); AZIMUTH=315 — Azimuth of the light; ALTITUDE=45 — Altitude of the light; COMPUTE_EDGES=false — Compute edges; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:importvectorintopostgisdatabaseavailableconnections — Export to PostgreSQL (available connections)

group: Vector miscellaneous

Exports a vector layer to an existing PostgreSQL database connection

Parameter	Type	Required	Default	Description
DATABASE	providerconnection	yes	—	Database (connection name)
INPUT	source	yes	—	Input layer
SHAPE_ENCODING	string	no	—	Shape encoding
GTYPE	enum	no	0	Output geometry type — options: 0=, 1=NONE, 2=GEOMETRY, 3=POINT, 4=LINESTRING, 5=POLYGON, 6=GEOMETRYCOLLECTION, 7=MULTIPOINT, 8=MULTIPOLYGON, 9=MULTILINESTRING, 10=CIRCULARSTRING, 11=COMPOUNDCURVE, 12=CURVEPOLYGON, 13=MULTICURVE, 14=MULTISURFACE, 15=CONVERT_TO_LINEAR, 16=CONVERT_TO_CURVE
A_SRS	crs	no	—	Assign an output CRS
T_SRS	crs	no	—	Reproject to this CRS on output
S_SRS	crs	no	—	Override source CRS
SCHEMA	databaseschema	no	public	Schema (schema name)
TABLE	databasetable	no	—	Table to import to (leave blank to use layer name)
PK	string	no	id	Primary key (new field)
PRIMARY_KEY	field	no	—	Primary key (existing field, used if the above option is left empty)
GEOCOLUMN	string	no	geom	Geometry column name
DIM	enum	no	0	Vector dimensions — options: 0=2, 1=3, 2=4
SIMPLIFY	string	no	—	Distance tolerance for simplification
SEGMENTIZE	string	no	—	Maximum distance between 2 nodes (densification)
SPAT	extent	no	—	Select features by extent (defined in input layer CRS)
CLIP	boolean	no	False	Clip the input layer using the above (rectangle) extent
WHERE	string	no	—	Select features using a SQL ”WHERE” statement (Ex: column='value')
GT	string	no	—	Group N features per transaction (Default: 20000)
OVERWRITE	boolean	no	True	Overwrite existing table
APPEND	boolean	no	False	Append to existing table
ADDFIELDS	boolean	no	False	Append and add new fields to existing table
LAUNDER	boolean	no	False	Do not launder columns/table names
INDEX	boolean	no	False	Do not create spatial index

Parameter	Type	Required	Default	Description
SKIPFAILURES	boolean	no	False	Continue after a failure, skipping the failed feature
MAKEVALID	boolean	no	False	Validate geometries based on Simple Features specification
PROMOTETOMULTI	boolean	no	True	Promote to Multipart
PRECISION	boolean	no	True	Keep width and precision of input attributes
OPTIONS	string	no	—	Additional creation options

Example usage

```
run gdal:importvectorintopostgisdatabaseavailableconnections DATABASE=… INPUT=input.gpkg
↪  SHAPE_ENCODING=value GTYPE=0 A_SRS=EPSG:6346 T_SRS=EPSG:6346 S_SRS=EPSG:6346
```

This calls **Export to PostgreSQL (available connections)**: DATABASE=… — Database (connection name); INPUT=input.gpkg — Input layer; SHAPE_ENCODING=value — Shape encoding; GTYPE=0 selects **; A_SRS=EPSG:6346 — Assign an output CRS; T_SRS=EPSG:6346 — Reproject to this CRS on output; S_SRS=EPSG:6346 — Override source CRS. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:importvectorintopostgisdatabasenewconnection — Export to PostgreSQL (new connection)

group: Vector miscellaneous

Exports a vector layer to a new PostgreSQL database connection

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
SHAPE_ENCODING	string	no	—	Shape encoding
GTYPE	enum	no	0	Output geometry type — options: 0=, 1=NONE, 2=GEOMETRY, 3=POINT, 4=LINESTRING, 5=POLYGON, 6=GEOMETRYCOLLECTION, 7=MULTIPOINT, 8=MULTIPOLYGON, 9=MULTILINESTRING, 10=CIRCULARSTRING, 11=COMPOUNDCURVE, 12=CURVEPOLYGON, 13=MULTICURVE, 14=MULTISURFACE, 15=CONVERT_TO_LINEAR, 16=CONVERT_TO_CURVE
A_SRS	crs	no	—	Assign an output CRS
T_SRS	crs	no	—	Reproject to this CRS on output
S_SRS	crs	no	—	Override source CRS
HOST	string	no	localhost	Host
PORT	string	no	5432	Port
USER	string	no	—	Username
DBNAME	string	no	—	Database name
PASSWORD	string	no	—	Password
SCHEMA	string	no	public	Schema name
TABLE	string	no	—	Table name, leave blank to use input name
PK	string	no	id	Primary key (new field)
PRIMARY_KEY	field	no	—	Primary key (existing field, used if the above option is left empty)
GEOCOLUMN	string	no	geom	Geometry column name
DIM	enum	no	0	Vector dimensions — options: 0=2, 1=3, 2=4
SIMPLIFY	string	no	—	Distance tolerance for simplification
SEGMENTIZE	string	no	—	Maximum distance between 2 nodes (densification)
SPAT	extent	no	—	Select features by extent (defined in input layer CRS)
CLIP	boolean	no	False	Clip the input layer using the above (rectangle) extent
FIELDS	field	no	—	Fields to include (leave empty to use all fields)
WHERE	string	no	—	Select features using a SQL ”WHERE” statement (Ex: column='value')
GT	string	no	—	Group N features per transaction (Default: 20000)
OVERWRITE	boolean	no	True	Overwrite existing table

Parameter	Type	Required	Default	Description
APPEND	boolean	no	False	Append to existing table
ADDFIELDS	boolean	no	False	Append and add new fields to existing table
LAUNDER	boolean	no	False	Do not launder columns/table names
INDEX	boolean	no	False	Do not create spatial index
SKIPFAILURES	boolean	no	False	Continue after a failure, skipping the failed feature
MAKEVALID	boolean	no	False	Validate geometries based on Simple Features specification
PROMOTETOMULTI	boolean	no	True	Promote to Multipart
PRECISION	boolean	no	True	Keep width and precision of input attributes
OPTIONS	string	no	—	Additional creation options

Example usage

```
run gdal:importvectorintopostgisdatabasenewconnection INPUT=input.gpkg
 ↪  SHAPE_ENCODING=value GTYPE=0 A_SRS=EPSG:6346 T_SRS=EPSG:6346 S_SRS=EPSG:6346
 ↪  HOST=localhost
```

This calls **Export to PostgreSQL (new connection)**: INPUT=input.gpkg — Input layer; SHAPE_ENCODING=value — Shape encoding; GTYPE=0 selects **; A_SRS=EPSG:6346 — Assign an output CRS; T_SRS=EPSG:6346 — Reproject to this CRS on output; S_SRS=EPSG:6346 — Override source CRS; HOST=localhost — Host. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:merge — Merge

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input layers
PCT	boolean	no	False	Grab pseudocolor table from first layer
SEPARATE	boolean	no	False	Place each input file into a separate band
NODATA_INPUT *(adv)*	number	no	—	Input pixel value to treat as NoData
NODATA_OUTPUT *(adv)*	number	no	—	Assign specified NoData value to output
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
DATA_TYPE	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
OUTPUT	rasterDestination	yes	—	Merged

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:merge INPUT="a.gpkg;b.gpkg" PCT=false SEPARATE=false OPTIONS=value DATA_TYPE=5
 ↪  OUTPUT=output.tif
```

This calls **Merge**: INPUT="a.gpkg;b.gpkg" — Input layers; PCT=false — Grab pseudocolor table from first layer; SEPARATE=false — Place each input file into a separate band; OPTIONS=value — Additional creation options; DATA_TYPE=5 selects *Float32*; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:nearblack — Near black

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
NEAR	number	no	15	How far from black (white)
WHITE	boolean	no	False	Search for nearly white pixels instead of nearly black
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Nearblack

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:nearblack INPUT=dem.tif NEAR=15 WHITE=false OPTIONS=value OUTPUT=output.tif
```

This calls **Near black**: INPUT=dem.tif — Input layer; NEAR=15 — How far from black (white); WHITE=false — Search for nearly white pixels instead of nearly black; OPTIONS=value — Additional creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:offsetcurve — Offset curve

group: Vector geoprocessing

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
GEOMETRY	string	no	geometry	Geometry column name
DISTANCE	distance	no	10	Offset distance (left-sided: positive, right-sided: negative)
OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	vectorDestination	yes	—	Offset curve

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:offsetcurve INPUT=input.gpkg GEOMETRY=geometry DISTANCE=10 OUTPUT=output.gpkg
```

This calls **Offset curve**: INPUT=input.gpkg — Input layer; GEOMETRY=geometry — Geometry column name; DISTANCE=10 — Offset distance (left-sided: positive, right-sided: negative); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:ogrinfo — Vector information

group: Vector miscellaneous

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
ALL_LAYERS	boolean	no	False	Enable listing of all layers in the dataset
SUMMARY_ONLY	boolean	no	True	Summary output only
NO_METADATA	boolean	no	False	Suppress metadata info
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	fileDestination	yes	—	Layer information

Outputs: OUTPUT (outputHtml)

Example usage

```
run gdal:ogrinfo INPUT=input.gpkg ALL_LAYERS=false SUMMARY_ONLY=true NO_METADATA=false
↪ OUTPUT=output.txt
```

This calls **Vector information**: INPUT=input.gpkg — Input layer; ALL_LAYERS=false — Enable listing of all layers in the dataset; SUMMARY_ONLY=true — Summary output only; NO_METADATA=false — Suppress metadata info; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:ogrinfojson — Vector information (JSON)

group: Vector miscellaneous

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
ALL_LAYERS	boolean	no	False	Enable listing of all layers in the dataset
FEATURES	boolean	no	False	Enable listing of features
NO_METADATA	boolean	no	False	Suppress metadata info
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	fileDestination	yes	—	Layer information

Outputs: OUTPUT (outputFile)

Example usage

```
run gdal:ogrinfojson INPUT=input.gpkg ALL_LAYERS=false FEATURES=false NO_METADATA=false
↪ OUTPUT=output.txt
```

This calls **Vector information (JSON)**: INPUT=input.gpkg — Input layer; ALL_LAYERS=false — Enable listing of all layers in the dataset; FEATURES=false — Enable listing of features; NO_METADATA=false — Suppress metadata info; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:onesidebuffer — One side buffer

group: Vector geoprocessing

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
GEOMETRY	string	no	geometry	Geometry column name
DISTANCE	distance	no	10	Buffer distance
BUFFER_SIDE	enum	no	0	Buffer side — options: 0=Right, 1=Left
FIELD	field	no	—	Dissolve by attribute
DISSOLVE	boolean	no	False	Dissolve all results
EXPLODE_COLLECTIONS	boolean	no	False	Produce one feature for each geometry in any kind of geometry collection in the source file
OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	vectorDestination	yes	—	One-sided buffer

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:onesidebuffer INPUT=input.gpkg GEOMETRY=geometry DISTANCE=10 BUFFER_SIDE=0
↪ FIELD=field1 DISSOLVE=false EXPLODE_COLLECTIONS=false OUTPUT=output.gpkg
```

This calls **One side buffer**: INPUT=input.gpkg — Input layer; GEOMETRY=geometry — Geometry column name; DISTANCE=10 — Buffer distance; BUFFER_SIDE=0 selects *Right*; FIELD=field1 — Dissolve by attribute; DISSOLVE=false — Dissolve all results; EXPLODE_COLLECTIONS=false —

Produce one feature for each geometry in any kind of geometry collection in the…; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:overviews — Build overviews (pyramids)

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
CLEAN	boolean	no	False	Remove all existing overviews
LEVELS *(adv)*	string	no	—	Overview levels (e.g. 2 4 8 16)
RESAMPLING *(adv)*	enum	no	—	Resampling method — options: 0=Nearest Neighbour (default), 1=Average, 2=Gaussian, 3=Cubic (4x4 Kernel), 4=Cubic B-Spline (4x4 Kernel), 5=Lanczos (6x6 Kernel), 6=Average MP, 7=Average in Mag/Phase Space, 8=Mode
FORMAT *(adv)*	enum	no	0	Overviews format — options: 0=Internal (if possible), 1=External (GTiff .ovr), 2=External (ERDAS Imagine .aux)
EXTRA *(adv)*	string	no	—	Additional command-line parameters

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:overviews INPUT=dem.tif CLEAN=false
```

This calls **Build overviews (pyramids)**: INPUT=dem.tif — Input layer; CLEAN=false — Remove all existing overviews. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:pansharp — Pansharpening

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
SPECTRAL	raster	yes	—	Spectral dataset
PANCHROMATIC	raster	yes	—	Panchromatic dataset
RESAMPLING *(adv)*	enum	no	2	Resampling algorithm — options: 0=Nearest Neighbour, 1=Bilinear (2x2 Kernel), 2=Cubic (4x4 Kernel), 3=Cubic B-Spline (4x4 Kernel), 4=Lanczos (6x6 Kernel), 5=Average
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Output

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:pansharp SPECTRAL=dem.tif PANCHROMATIC=dem.tif OPTIONS=value OUTPUT=output.tif
```

This calls **Pansharpening**: SPECTRAL=dem.tif — Spectral dataset; PANCHROMATIC=dem.tif — Panchromatic dataset; OPTIONS=value — Additional creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:pcttorgb — PCT to RGB

group: Raster conversion

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
RGBA	boolean	no	False	Generate a RGBA file
OUTPUT	rasterDestination	yes	—	PCT to RGB

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:pcttorgb INPUT=dem.tif BAND=1 RGBA=false OUTPUT=output.tif
```

This calls **PCT to RGB**: INPUT=dem.tif — Input layer; BAND=1 — Band number; RGBA=false — Generate a RGBA file; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:pointsalonglines — Points along lines

group: Vector geoprocessing

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
GEOMETRY	string	no	geometry	Geometry column name
DISTANCE	number	no	0.5	Distance from line start represented as fraction of line length
OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	vectorDestination	yes	—	Points along lines

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:pointsalonglines INPUT=input.gpkg GEOMETRY=geometry DISTANCE=0.5
↪  OUTPUT=output.gpkg
```

This calls **Points along lines**: INPUT=input.gpkg — Input layer; GEOMETRY=geometry — Geometry column name; DISTANCE=0.5 — Distance from line start represented as fraction of line length; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:polygonize — Polygonize (raster to vector) ★

group: Raster conversion · **niva verb:** polygonize

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
FIELD	string	no	DN	Name of the field to create
EIGHT_CONNECTEDNESS	boolean	no	False	Use 8-connectedness
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	vectorDestination	yes	—	Vectorized

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:polygonize INPUT=dem.tif BAND=1 FIELD=DN EIGHT_CONNECTEDNESS=false
↪ OUTPUT=output.gpkg
```

This calls **Polygonize (raster to vector)**: `INPUT=dem.tif` — Input layer; `BAND=1` — Band number; `FIELD=DN` — Name of the field to create; `EIGHT_CONNECTEDNESS=false` — Use 8-connectedness; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:proximity` — Proximity (raster distance)

group: Raster analysis

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input layer
`BAND`	band	no	1	Band number
`VALUES`	string	no	—	List of target pixels
`UNITS`	enum	no	1	Distance units — options: 0=Georeferenced coordinates, 1=Pixel coordinates
`MAX_DISTANCE`	number	no	0.0	The maximum distance to generate
`REPLACE`	number	no	0.0	Value to apply to pixels within the maximum distance of target pixels
`NODATA`	number	no	0.0	Nodata value to use for the destination proximity raster
`OPTIONS`	string	no	—	Additional creation options
`CREATION_OPTIONS *(adv)*`	string	no	—	Additional creation options
`EXTRA *(adv)*`	string	no	—	Additional command-line parameters
`DATA_TYPE *(adv)*`	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
`OUTPUT`	rasterDestination	yes	—	Proximity map

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run gdal:proximity INPUT=dem.tif BAND=1 VALUES=value UNITS=1 MAX_DISTANCE=0 REPLACE=0
↪ NODATA=0 OUTPUT=output.tif
```

This calls **Proximity (raster distance)**: `INPUT=dem.tif` — Input layer; `BAND=1` — Band number; `VALUES=value` — List of target pixels; `UNITS=1` selects *Pixel coordinates*; `MAX_DISTANCE=0` — The maximum distance to generate; `REPLACE=0` — Value to apply to pixels within the maximum distance of target pixels; `NODATA=0` — Nodata value to use for the destination proximity raster; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:rastercalculator` — Raster calculator

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
`INPUT_A`	raster	yes	—	Input layer A
`BAND_A`	band	yes	—	Number of raster band for A
`INPUT_B`	raster	no	—	Input layer B
`BAND_B`	band	no	—	Number of raster band for B
`INPUT_C`	raster	no	—	Input layer C
`BAND_C`	band	no	—	Number of raster band for C
`INPUT_D`	raster	no	—	Input layer D

Parameter	Type	Required	Default	Description
BAND_D	band	no	—	Number of raster band for D
INPUT_E	raster	no	—	Input layer E
BAND_E	band	no	—	Number of raster band for E
INPUT_F	raster	no	—	Input layer F
BAND_F	band	no	—	Number of raster band for F
FORMULA	string	no	A*2	Calculation in gdalnumeric syntax using +-/* or any numpy array functions (i.e. logical_and())
NO_DATA	number	no	—	Set output NoData value
EXTENT_OPT	enum	no	0	Handling of extent differences — options: 0=Ignore, 1=Fail, 2=Union, 3=Intersect
PROJWIN	extent	no	—	Output extent
RTYPE	enum	no	5	Output raster type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Calculated

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:rastercalculator INPUT_A=dem.tif BAND_A=1 INPUT_B=dem.tif BAND_B=1
 ↪  INPUT_C=dem.tif BAND_C=1 INPUT_D=dem.tif OUTPUT=output.tif
```

This calls **Raster calculator**: INPUT_A=dem.tif — Input layer A; BAND_A=1 — Number of raster band for A; INPUT_B=dem.tif — Input layer B; BAND_B=1 — Number of raster band for B; INPUT_C=dem.tif — Input layer C; BAND_C=1 — Number of raster band for C; INPUT_D=dem.tif — Input layer D; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:rasterize — Rasterize (vector to raster)

group: Vector conversion

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	no	—	Field to use for a burn-in value
BURN	number	no	0.0	A fixed value to burn
USE_Z	boolean	no	False	Burn value extracted from the "Z" values of the feature
UNITS	enum	yes	—	Output raster size units — options: 0=Pixels, 1=Georeferenced units
WIDTH	number	no	0.0	Width/Horizontal resolution
HEIGHT	number	no	0.0	Height/Vertical resolution
EXTENT	extent	no	—	Output extent
NODATA	number	no	—	Assign a specified NoData value to output bands
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
INIT *(adv)*	number	no	—	Pre-initialize the output image with value
INVERT *(adv)*	boolean	no	False	Invert rasterization

Parameter	Type	Required	Default	Description
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Rasterized

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:rasterize INPUT=input.gpkg UNITS=0 FIELD=field1 BURN=0 USE_Z=false WIDTH=0
↪  HEIGHT=0 OUTPUT=output.tif
```

This calls **Rasterize (vector to raster)**: INPUT=input.gpkg — Input layer; UNITS=0 selects *Pixels*; FIELD=field1 — Field to use for a burn-in value; BURN=0 — A fixed value to burn; USE_Z=false — Burn value extracted from the ”Z” values of the feature; WIDTH=0 — Width/Horizontal resolution; HEIGHT=0 — Height/Vertical resolution; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:rasterize_over — Rasterize (overwrite with attribute)

group: Vector conversion

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input vector layer
INPUT_RASTER	raster	yes	—	Input raster layer
FIELD	field	yes	—	Field to use for burn in value
ADD *(adv)*	boolean	no	False	Add burn in values to existing raster values
EXTRA *(adv)*	string	no	—	Additional command-line parameters

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:rasterize_over INPUT=input.gpkg INPUT_RASTER=dem.tif FIELD=field1
```

This calls **Rasterize (overwrite with attribute)**: INPUT=input.gpkg — Input vector layer; INPUT_RASTER=dem.tif — Input raster layer; FIELD=field1 — Field to use for burn in value. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:rasterize_over_fixed_value — Rasterize (overwrite with fixed value)

group: Vector conversion

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input vector layer
INPUT_RASTER	raster	yes	—	Input raster layer
BURN	number	no	0.0	A fixed value to burn
ADD *(adv)*	boolean	no	False	Add burn in values to existing raster values
EXTRA *(adv)*	string	no	—	Additional command-line parameters

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:rasterize_over_fixed_value INPUT=input.gpkg INPUT_RASTER=dem.tif BURN=0
```

This calls **Rasterize (overwrite with fixed value)**: INPUT=input.gpkg — Input vector layer; INPUT_RASTER=dem.tif — Input raster layer; BURN=0 — A fixed value to burn. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:rearrange_bands — Rearrange bands

group: Raster conversion

This algorithm creates a new raster using selected band(s) from a given raster layer.

The algorithm also makes it possible to reorder the bands for the newly-created raster.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BANDS	band	yes	—	Selected band(s)
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
DATA_TYPE *(adv)*	enum	no	0	Output data type — options: 0=Use Input Layer Data Type, 1=Byte, 2=Int16, 3=UInt16, 4=UInt32, 5=Int32, 6=Float32, 7=Float64, 8=CInt16, 9=CInt32, 10=CFloat32, 11=CFloat64, 12=Int8
OUTPUT	rasterDestination	yes	—	Converted

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:rearrange_bands INPUT=dem.tif BANDS=1 OPTIONS=value OUTPUT=output.tif
```

This calls **Rearrange bands**: INPUT=dem.tif — Input layer; BANDS=1 — Selected band(s); OPTIONS=value — Additional creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:retile — Retile

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
INPUT	multilayer	yes	—	Input files
TILE_SIZE_X	number	no	256	Tile width
TILE_SIZE_Y	number	no	256	Tile height
OVERLAP	number	no	0	Overlap in pixels between consecutive tiles
LEVELS	number	no	1	Number of pyramids levels to build
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
SOURCE_CRS *(adv)*	crs	no	—	Source coordinate reference system
RESAMPLING *(adv)*	enum	no	0	Resampling method — options: 0=Nearest Neighbour, 1=Bilinear (2x2 Kernel), 2=Cubic (4x4 Kernel), 3=Cubic B-Spline (4x4 Kernel), 4=Lanczos (6x6 Kernel)
DELIMITER *(adv)*	string	no	;	Column delimiter used in the CSV file
EXTRA *(adv)*	string	no	—	Additional command-line parameters
DATA_TYPE *(adv)*	enum	no	5	Output data type — options: 0=Byte, 1=Int16, 2=UInt16, 3=UInt32, 4=Int32, 5=Float32, 6=Float64, 7=CInt16, 8=CInt32, 9=CFloat32, 10=CFloat64, 11=Int8
ONLY_PYRAMIDS *(adv)*	boolean	no	False	Build only the pyramids
DIR_FOR_ROW *(adv)*	boolean	no	False	Use separate directory for each tiles row
OUTPUT	folderDestination	yes	—	Output directory
OUTPUT_CSV	fileDestination	no	—	CSV file containing the tile(s) georeferencing information

Outputs: OUTPUT (outputFolder), OUTPUT_CSV (outputFile)

Example usage

```
run gdal:retile INPUT="a.gpkg;b.gpkg" TILE_SIZE_X=256 TILE_SIZE_Y=256 OVERLAP=0 LEVELS=1
↪ OPTIONS=value OUTPUT=output/
```

This calls **Retile**: `INPUT="a.gpkg;b.gpkg"` — Input files; `TILE_SIZE_X=256` — Tile width; `TILE_SIZE_Y=256` — Tile height; `OVERLAP=0` — Overlap in pixels between consecutive tiles; `LEVELS=1` — Number of pyramids levels to build; `OPTIONS=value` — Additional creation options; `OUTPUT=output/` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:rgbtopct` — RGB to PCT

group: Raster conversion

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input layer
`NCOLORS`	number	no	2	Number of colors
`OUTPUT`	rasterDestination	yes	—	RGB to PCT

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run gdal:rgbtopct INPUT=dem.tif NCOLORS=2 OUTPUT=output.tif
```

This calls **RGB to PCT**: `INPUT=dem.tif` — Input layer; `NCOLORS=2` — Number of colors; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:roughness` — Roughness

group: Raster analysis

Parameter	Type	Required	Default	Description
`INPUT`	raster	yes	—	Input layer
`BAND`	band	no	1	Band number
`COMPUTE_EDGES`	boolean	no	False	Compute edges
`OPTIONS`	string	no	—	Additional creation options
`CREATION_OPTIONS *(adv)*`	string	no	—	Additional creation options
`OUTPUT`	rasterDestination	yes	—	Roughness

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run gdal:roughness INPUT=dem.tif BAND=1 COMPUTE_EDGES=false OPTIONS=value
↪ OUTPUT=output.tif
```

This calls **Roughness**: `INPUT=dem.tif` — Input layer; `BAND=1` — Band number; `COMPUTE_EDGES=false` — Compute edges; `OPTIONS=value` — Additional creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`gdal:sieve` — Sieve

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
THRESHOLD	number	no	10	Threshold
EIGHT_CONNECTED-NESS	boolean	no	False	Use 8-connectedness
NO_MASK	boolean	no	False	Do not use the default validity mask for the input band
MASK_LAYER	raster	no	—	Validity mask
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Sieved

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:sieve INPUT=dem.tif THRESHOLD=10 EIGHT_CONNECTEDNESS=false NO_MASK=false
↪  MASK_LAYER=dem.tif OUTPUT=output.tif
```

This calls **Sieve**: INPUT=dem.tif — Input layer; THRESHOLD=10 — Threshold; EIGHT_CONNECTEDNESS=false — Use 8-connectedness; NO_MASK=false — Do not use the default validity mask for the input band; MASK_LAYER=dem.tif — Validity mask; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:slope — Slope ★

group: Raster analysis · **niva verb:** slope

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
SCALE	number	no	1.0	Ratio of vertical units to horizontal
AS_PERCENT	boolean	no	False	Express slope as percent instead of degrees
COMPUTE_EDGES	boolean	no	False	Compute edges
ZEVENBERGEN	boolean	no	False	Use Zevenbergen&Thorne formula instead of the Horn's one
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Slope

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:slope INPUT=dem.tif BAND=1 SCALE=1 AS_PERCENT=false COMPUTE_EDGES=false
↪  ZEVENBERGEN=false OPTIONS=value OUTPUT=output.tif
```

This calls **Slope**: INPUT=dem.tif — Input layer; BAND=1 — Band number; SCALE=1 — Ratio of vertical units to horizontal; AS_PERCENT=false — Express slope as percent instead of degrees; COMPUTE_EDGES=false — Compute edges; ZEVENBERGEN=false — Use Zevenbergen&Thorne formula instead of the Horn's one; OPTIONS=value — Additional creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:tileindex — Tile index

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input files
PATH_FIELD_NAME	string	no	location	Field name to hold the file path to the indexed rasters
ABSOLUTE_PATH	boolean	no	False	Store absolute path to the indexed rasters
PROJ_DIFFERENCE	boolean	no	False	Skip files with different projection reference
TARGET_CRS *(adv)*	crs	no	—	Transform geometries to the given CRS
CRS_FIELD_NAME *(adv)*	string	no	—	The name of the field to store the SRS of each tile
CRS_FORMAT *(adv)*	enum	no	0	The format in which the CRS of each tile must be written — options: 0=Auto, 1=Well-known text (WKT), 2=EPSG, 3=Proj.4
OUTPUT	vectorDestination	yes	—	Tile index

Outputs: OUTPUT (outputVector)

Example usage

```
run gdal:tileindex LAYERS="a.gpkg;b.gpkg" PATH_FIELD_NAME=location ABSOLUTE_PATH=false
↪  PROJ_DIFFERENCE=false OUTPUT=output.gpkg
```

This calls **Tile index**: LAYERS="a.gpkg;b.gpkg" — Input files; PATH_FIELD_NAME=location — Field name to hold the file path to the indexed rasters; ABSOLUTE_PATH=false — Store absolute path to the indexed rasters; PROJ_DIFFERENCE=false — Skip files with different projection reference; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:tpitopographicpositionindex — Topographic Position Index (TPI)

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
COMPUTE_EDGES	boolean	no	False	Compute edges
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	rasterDestination	yes	—	Topographic Position Index

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:tpitopographicpositionindex INPUT=dem.tif BAND=1 COMPUTE_EDGES=false
↪  OPTIONS=value OUTPUT=output.tif
```

This calls **Topographic Position Index (TPI)**: INPUT=dem.tif — Input layer; BAND=1 — Band number; COMPUTE_EDGES=false — Compute edges; OPTIONS=value — Additional creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:translate — Translate (convert format)

group: Raster conversion

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
TARGET_CRS	crs	no	—	Override the projection for the output file
NODATA	number	no	—	Assign a specified NoData value to output bands

Parameter	Type	Required	Default	Description
COPY_SUBDATASETS	boolean	no	False	Copy all subdatasets of this file to individual output files
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
DATA_TYPE *(adv)*	enum	no	0	Output data type — options: 0=Use Input Layer Data Type, 1=Byte, 2=Int16, 3=UInt16, 4=UInt32, 5=Int32, 6=Float32, 7=Float64, 8=CInt16, 9=CInt32, 10=CFloat32, 11=CFloat64, 12=Int8
OUTPUT	rasterDestination	yes	—	Converted

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:translate INPUT=dem.tif TARGET_CRS=EPSG:6346 NODATA=10 COPY_SUBDATASETS=false
↪  OPTIONS=value OUTPUT=output.tif
```

This calls **Translate (convert format)**: INPUT=dem.tif — Input layer; TARGET_CRS=EPSG:6346 — Override the projection for the output file; NODATA=10 — Assign a specified NoData value to output bands; COPY_SUBDATASETS=false — Copy all subdatasets of this file to individual output files; OPTIONS=value — Additional creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:triterrainruggednessindex — Terrain Ruggedness Index (TRI)

group: Raster analysis

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
COMPUTE_EDGES	boolean	no	False	Compute edges
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
OUTPUT	rasterDestination	yes	—	Terrain Ruggedness Index

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:triterrainruggednessindex INPUT=dem.tif BAND=1 COMPUTE_EDGES=false OPTIONS=value
↪  OUTPUT=output.tif
```

This calls **Terrain Ruggedness Index (TRI)**: INPUT=dem.tif — Input layer; BAND=1 — Band number; COMPUTE_EDGES=false — Compute edges; OPTIONS=value — Additional creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:viewshed — Viewshed

group: Raster miscellaneous

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
OBSERVER	point	yes	—	Observer location

Parameter	Type	Required	Default	Description
OBSERVER_HEIGHT	number	no	1.0	Observer height, DEM units
TARGET_HEIGHT	number	no	1.0	Target height, DEM units
MAX_DISTANCE	distance	no	100.0	Maximum distance from observer to compute visibility
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Output

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:viewshed INPUT=dem.tif OBSERVER=100,100 BAND=1 OBSERVER_HEIGHT=1 TARGET_HEIGHT=1
↪ MAX_DISTANCE=100 OPTIONS=value OUTPUT=output.tif
```

This calls **Viewshed**: INPUT=dem.tif — Input layer; OBSERVER=100,100 — Observer location; BAND=1 — Band number; OBSERVER_HEIGHT=1 — Observer height, DEM units; TARGET_HEIGHT=1 — Target height, DEM units; MAX_DISTANCE=100 — Maximum distance from observer to compute visibility; OPTIONS=value — Additional creation options; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

gdal:warpreproject — Warp (reproject) ★

group: Raster projections · **niva verb:** warp

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
SOURCE_CRS	crs	no	—	Source CRS
TARGET_CRS	crs	no	—	Target CRS
RESAMPLING	enum	no	0	Resampling method to use — options: 0=Nearest Neighbour, 1=Bilinear (2x2 Kernel), 2=Cubic (4x4 Kernel), 3=Cubic B-Spline (4x4 Kernel), 4=Lanczos (6x6 Kernel), 5=Average, 6=Mode, 7=Maximum, 8=Minimum, 9=Median, 10=First Quartile (Q1), 11=Third Quartile (Q3)
NODATA	number	no	—	Nodata value for output bands
TARGET_RESOLUTION	number	no	—	Output file resolution in target georeferenced units
OPTIONS	string	no	—	Additional creation options
CREATION_OPTIONS *(adv)*	string	no	—	Additional creation options
DATA_TYPE *(adv)*	enum	no	0	Output data type — options: 0=Use Input Layer Data Type, 1=Byte, 2=Int16, 3=UInt16, 4=UInt32, 5=Int32, 6=Float32, 7=Float64, 8=CInt16, 9=CInt32, 10=CFloat32, 11=CFloat64, 12=Int8
TARGET_EXTENT *(adv)*	extent	no	—	Georeferenced extents of output file to be created
TARGET_EXTENT_CRS *(adv)*	crs	no	—	CRS of the target raster extent
MULTITHREADING *(adv)*	boolean	no	False	Use multithreaded warping implementation
EXTRA *(adv)*	string	no	—	Additional command-line parameters
OUTPUT	rasterDestination	yes	—	Reprojected

Outputs: OUTPUT (outputRaster)

Example usage

```
run gdal:warpreproject INPUT=dem.tif SOURCE_CRS=EPSG:6346 TARGET_CRS=EPSG:6346
↪ RESAMPLING=0 NODATA=10 TARGET_RESOLUTION=10 OPTIONS=value OUTPUT=output.tif
```

This calls **Warp (reproject)**: `INPUT=dem.tif` — Input layer; `SOURCE_CRS=EPSG:6346` — Source CRS; `TARGET_CRS=EPSG:6346` — Target CRS; `RESAMPLING=0` selects *Nearest Neighbour*; `NODATA=10` — Nodata value for output bands; `TARGET_RESOLUTION=10` — Output file resolution in target georeferenced units; `OPTIONS=value` — Additional creation options; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis: algorithms

qgis — QGIS (Python) algorithms — 39 algorithms (QGIS 4.0.3). Auto-generated by `scripts/gen_algorithms.py`; see the index.

Call any of these with `run <id> KEY=value …` (the **Name** column is the KEY); each entry includes a worked **Example usage**. A ★ marks an algorithm with a friendly niva alias verb.

qgis:advancedpythonfieldcalculator — Advanced Python field calculator

group: Vector table

This algorithm adds a new attribute to a vector layer, with values calculated by applying an expression to each feature. The expression is defined as a Python function.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD_NAME	string	no	NewField	Result field name
FIELD_TYPE	enum	yes	—	Field type — options: 0=Integer (32 bit), 1=Decimal (double), 2=Text (string), 3=Boolean, 4=Date, 5=Time, 6=Date & Time, 7=Binary Object (BLOB), 8=String List, 9=Integer List, 10=Decimal (double) List
FIELD_LENGTH	number	no	10	Field length
FIELD_PRECISION	number	no	3	Field precision
GLOBAL	string	no	—	Global expression
FORMULA	string	no	value =	Formula
OUTPUT	sink	yes	—	Calculated

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:advancedpythonfieldcalculator INPUT=input.gpkg FIELD_TYPE=0 FIELD_NAME=NewField
 ↪  FIELD_LENGTH=10 FIELD_PRECISION=3 GLOBAL=value FORMULA="value = " OUTPUT=output.gpkg
```

This calls **Advanced Python field calculator**: `INPUT=input.gpkg` — Input layer; `FIELD_TYPE=0` selects *Integer (32 bit)*; `FIELD_NAME=NewField` — Result field name; `FIELD_LENGTH=10` — Field length; `FIELD_PRECISION=3` — Field precision; `GLOBAL=value` — Global expression; `FORMULA="value = "` — Formula; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:barplot — Bar plot

group: Plots

This algorithm creates a bar plot from a category and a layer field.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
NAME_FIELD	field	yes	—	Category name field
VALUE_FIELD	field	yes	—	Value field
OUTPUT	fileDestination	yes	—	Bar plot
TITLE	string	no	—	Title
XAXIS_TITLE	string	no	—	X-axis title
YAXIS_TITLE	string	no	—	Y-axis title

Outputs: OUTPUT (outputHtml)

Example usage

```
run qgis:barplot INPUT=input.gpkg NAME_FIELD=field1 VALUE_FIELD=field1 TITLE=value
 ↪  XAXIS_TITLE=value YAXIS_TITLE=value OUTPUT=output.txt
```

This calls **Bar plot**: `INPUT=input.gpkg` — Input layer; `NAME_FIELD=field1` — Category name field; `VALUE_FIELD=field1` — Value field; `TITLE=value` — Title; `XAXIS_TITLE=value` — X-axis title; `YAXIS_TITLE=value` — Y-axis title; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:boxplot` — Box plot

group: Plots

This algorithm creates a box plot from a category and a layer field.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
NAME_FIELD	field	yes	—	Category name field
VALUE_FIELD	field	yes	—	Value field
MSD	enum	no	0	Additional Statistic Lines — options: 0=Show Mean, 1=Show Standard Deviation, 2=Don't show Mean and Standard Deviation
TITLE	string	no	—	Title
XAXIS_TITLE	string	no	—	X-axis title
YAXIS_TITLE	string	no	—	Y-axis title
OUTPUT	fileDestination	yes	—	Box plot

Outputs: OUTPUT (outputHtml)

Example usage

```
run qgis:boxplot INPUT=input.gpkg NAME_FIELD=field1 VALUE_FIELD=field1 MSD=0 TITLE=value
 ↪  XAXIS_TITLE=value YAXIS_TITLE=value OUTPUT=output.txt
```

This calls **Box plot**: `INPUT=input.gpkg` — Input layer; `NAME_FIELD=field1` — Category name field; `VALUE_FIELD=field1` — Value field; `MSD=0` selects *Show Mean*; `TITLE=value` — Title; `XAXIS_TITLE=value` — X-axis title; `YAXIS_TITLE=value` — Y-axis title; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:distancematrix` — Distance matrix

group: Vector analysis

This algorithm creates a table containing a distance matrix, with distances between all the points in a points layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input point layer
INPUT_FIELD	field	yes	—	Input unique ID field
TARGET	source	yes	—	Target point layer
TARGET_FIELD	field	yes	—	Target unique ID field
MATRIX_TYPE	enum	no	0	Output matrix type — options: 0=Linear (N*k x 3) distance matrix, 1=Standard (N x T) distance matrix, 2=Summary distance matrix (mean, std. dev., min, max)
NEAREST_POINTS	number	no	0	Use only the nearest (k) target points
OUTPUT	sink	yes	—	Distance matrix

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:distancematrix INPUT=input.gpkg INPUT_FIELD=field1 TARGET=input.gpkg
↪  TARGET_FIELD=field1 MATRIX_TYPE=0 NEAREST_POINTS=0 OUTPUT=output.gpkg
```

This calls **Distance matrix**: INPUT=input.gpkg — Input point layer; INPUT_FIELD=field1 — Input unique ID field; TARGET=input.gpkg — Target point layer; TARGET_FIELD=field1 — Target unique ID field; MATRIX_TYPE=0 selects *Linear (N*k x 3) distance matrix*; NEAREST_POINTS=0 — Use only the nearest (k) target points; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:distancetonearesthublinetohub — Distance to nearest hub (line to hub)

group: Vector analysis

Given an origin and a destination layers, this algorithm computes the distance between origin features and their closest destination one. Distance calculations are based on the feature's center. The resulting layer contains lines linking each origin point with its nearest destination feature.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Source points layer
HUBS	source	yes	—	Destination hubs layer
FIELD	field	yes	—	Hub layer name attribute
UNIT	enum	yes	—	Measurement unit — options: 0=Meters, 1=Feet, 2=Miles, 3=Kilometers, 4=Layer units
OUTPUT	sink	yes	—	Hub distance

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:distancetonearesthublinetohub INPUT=input.gpkg HUBS=input.gpkg FIELD=field1
↪  UNIT=0 OUTPUT=output.gpkg
```

This calls **Distance to nearest hub (line to hub)**: INPUT=input.gpkg — Source points layer; HUBS=input.gpkg — Destination hubs layer; FIELD=field1 — Hub layer name attribute; UNIT=0 selects *Meters*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:distancetonearesthubpoints — Distance to nearest hub (points)

group: Vector analysis

Given an origin and a destination layers, this algorithm computes the distance between origin features and their closest destination one. Distance calculations are based on the feature's center.

The resulting layer contains origin features center point with an additional field indicating the identifier of the nearest destination feature and the distance to it.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Source points layer
HUBS	source	yes	—	Destination hubs layer
FIELD	field	yes	—	Hub layer name attribute
UNIT	enum	yes	—	Measurement unit — options: 0=Meters, 1=Feet, 2=Miles, 3=Kilometers, 4=Layer units
OUTPUT	sink	yes	—	Hub distance

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:distancetonearesthubpoints INPUT=input.gpkg HUBS=input.gpkg FIELD=field1 UNIT=0
↪ OUTPUT=output.gpkg
```

This calls **Distance to nearest hub (points)**: INPUT=input.gpkg — Source points layer; HUBS=input.gpkg — Destination hubs layer; FIELD=field1 — Hub layer name attribute; UNIT=0 selects *Meters*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:eliminateselectedpolygons — Eliminate selected polygons

group: Vector geometry

This algorithm combines selected polygons from the input layer with certain adjacent polygons by erasing their common boundary. The adjacent polygon can be either the one with the largest or smallest area or the one sharing the largest common boundary with the polygon to be eliminated. Eliminate is normally used to get rid of sliver polygons, i.e. tiny polygons that are a result of polygon intersection processes where boundaries of the inputs are similar but not identical.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
MODE	enum	yes	—	Merge selection with the neighbouring polygon with the — options: 0=Largest Area, 1=Smallest Area, 2=Largest Common Boundary
OUTPUT	sink	yes	—	Eliminated

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:eliminateselectedpolygons INPUT=input.gpkg MODE=0 OUTPUT=output.gpkg
```

This calls **Eliminate selected polygons**: INPUT=input.gpkg — Input layer; MODE=0 selects *Largest Area*; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:executesql — Execute SQL

group: Vector general

This algorithm runs a query with SQL syntax. Input data sources are identified with input1, input2, ..., inputN and a simple query will look like: 'SELECT * FROM input1'. The result of the query will be added as a new layer.

Parameter	Type	Required	Default	Description
INPUT_DATASOURCES	multilayer	no	—	Input data sources (called input1, .., inputN in the query)
INPUT_QUERY	execute_sql	yes	—	SQL query
INPUT_UID_FIELD	string	no	—	Unique identifier field
INPUT_GEOMETRY_FIELD	string	no	—	Geometry field
INPUT_GEOMETRY_TYPE	enum	no	0	Geometry type — options: 0=Autodetect, 1=No geometry, 2=Point, 3=LineString, 4=Polygon, 5=MultiPoint, 6=MultiLineString, 7=MultiPolygon
INPUT_GEOMETRY_CRS	crs	no	—	CRS
OUTPUT	sink	yes	—	SQL Output

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:executesql INPUT_QUERY=value INPUT_DATASOURCES="a.gpkg;b.gpkg"
 ↪  INPUT_UID_FIELD=value INPUT_GEOMETRY_FIELD=value INPUT_GEOMETRY_TYPE=0
 ↪  INPUT_GEOMETRY_CRS=EPSG:6346 OUTPUT=output.gpkg
```

This calls **Execute SQL**: INPUT_QUERY=value — SQL query; INPUT_DATASOURCES="a.gpkg;b.gpkg" — Input data sources (called input1, .., inputN in the query); INPUT_UID_FIELD=value — Unique identifier field; INPUT_GEOMETRY_FIELD=value — Geometry field; INPUT_GEOMETRY_TYPE=0 selects *Autodetect*; INPUT_GEOMETRY_CRS=EPSG:6346 — CRS; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:generatepointspixelcentroidsalongline — Generate points (pixel centroids) along line

group: Vector creation

This algorithm generates a point vector layer from an input raster and line layer. The points correspond to the pixel centroids that intersect the line layer.

Parameter	Type	Required	Default	Description
INPUT_RASTER	raster	yes	—	Raster layer
INPUT_VECTOR	source	yes	—	Vector layer
OUTPUT	sink	yes	—	Points along lines

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:generatepointspixelcentroidsalongline INPUT_RASTER=dem.tif
 ↪  INPUT_VECTOR=input.gpkg OUTPUT=output.gpkg
```

This calls **Generate points (pixel centroids) along line**: INPUT_RASTER=dem.tif — Raster layer; INPUT_VECTOR=input.gpkg — Vector layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:heatmapkerneldensityestimation — Heatmap (Kernel Density Estimation)

group: Interpolation

This algorithm creates a density (heatmap) raster of an input point vector layer using kernel density estimation. Heatmaps allow easy identification of hotspots and clustering of points. The density

is calculated based on the number of points in a location, with larger numbers of clustered points resulting in larger values.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Point layer
RADIUS	distance	no	100.0	Radius
RADIUS_FIELD *(adv)*	field	no	—	Radius from field
PIXEL_SIZE	number	no	0.1	Output raster size
WEIGHT_FIELD *(adv)*	field	no	—	Weight from field
KERNEL *(adv)*	enum	no	0	Kernel shape — options: 0=Quartic, 1=Triangular, 2=Uniform, 3=Triweight, 4=Epanechnikov
DECAY *(adv)*	number	no	0.0	Decay ratio (Triangular kernels only)
OUTPUT_VALUE *(adv)*	enum	no	0	Output value scaling — options: 0=Raw, 1=Scaled
OUTPUT	rasterDestination	yes	—	Heatmap

Outputs: OUTPUT (outputRaster)

Example usage

```
run qgis:heatmapkerneldensityestimation INPUT=input.gpkg RADIUS=100 PIXEL_SIZE=0.1
↪  OUTPUT=output.tif
```

This calls **Heatmap (Kernel Density Estimation)**: INPUT=input.gpkg — Point layer; RADIUS=100 — Radius; PIXEL_SIZE=0.1 — Output raster size; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:hypsometriccurves — Hypsometric curves

group: Raster terrain analysis

This algorithm computes hypsometric curves for an input Digital Elevation Model (DEM). Curves are produced as table files in an output folder specified by the user.

Parameter	Type	Required	Default	Description
INPUT_DEM	raster	yes	—	DEM to analyze
BOUNDARY_LAYER	source	yes	—	Boundary layer
STEP	number	no	100.0	Step
USE_PERCENTAGE	boolean	no	False	Use % of area instead of absolute value
OUTPUT_DIRECTORY	folderDestination	yes	—	Hypsometric curves

Outputs: OUTPUT_DIRECTORY (outputFolder)

Example usage

```
run qgis:hypsometriccurves INPUT_DEM=dem.tif BOUNDARY_LAYER=input.gpkg STEP=100
↪  USE_PERCENTAGE=false OUTPUT_DIRECTORY=output_directory/
```

This calls **Hypsometric curves**: INPUT_DEM=dem.tif — DEM to analyze; BOUNDARY_LAYER=input.gpkg — Boundary layer; STEP=100 — Step; USE_PERCENTAGE=false — Use % of area instead of absolute value; OUTPUT_DIRECTORY=output_directory/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:idwinterpolation — IDW interpolation

group: Interpolation

This algorithm generates an Inverse Distance Weighted (IDW) interpolation of a point vector layer. Sample points are weighted during interpolation such that the influence of one point relative to another declines with distance from the unknown point you want to create.

Parameter	Type	Required	Default	Description
INTERPOLATION_DATA	idw_interpolation_data	yes	—	Input layer(s)
DISTANCE_COEFFICIENT	number	no	2.0	Distance coefficient P
EXTENT	extent	yes	—	Extent
PIXEL_SIZE	number	no	0.1	Output raster size
COLUMNS	number	no	—	Number of columns
ROWS	number	no	—	Number of rows
OUTPUT	rasterDestination	yes	—	Interpolated

Outputs: OUTPUT (outputRaster)

Example usage

```
run qgis:idwinterpolation INTERPOLATION_DATA=… EXTENT=0,1000,0,1000 DISTANCE_COEFFICIENT=2
 ↪  PIXEL_SIZE=0.1 COLUMNS=10 ROWS=10 OUTPUT=output.tif
```

This calls **IDW interpolation**: INTERPOLATION_DATA=… — Input layer(s); EXTENT=0,1000,0,1000 — Extent; DISTANCE_COEFFICIENT=2 — Distance coefficient P; PIXEL_SIZE=0.1 — Output raster size; COLUMNS=10 — Number of columns; ROWS=10 — Number of rows; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:knearestconcavehull — Concave hull (k-nearest neighbor)

group: Vector geometry

This algorithm generates a concave hull polygon from a set of points. If the input layer is a line or polygon layer, it will use the nodes. The number of neighbours to consider determines the concaveness of the output polygon. A lower number will result in a concave hull that follows the points very closely, while a higher number will have a smoother shape. The minimum number of neighbour points to consider is 3. A value equal to or greater than the number of points will result in a convex hull. If a field is selected, the algorithm will group the features in the input layer using unique values in that field and generate individual polygons in the output layer for each group.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
KNEIGHBORS	number	no	3	Number of neighboring points to consider (a lower number is more concave, a higher number is smoother)
FIELD	field	no	—	Field (set if creating concave hulls by class)
OUTPUT	sink	yes	—	Concave hull

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:knearestconcavehull INPUT=input.gpkg KNEIGHBORS=3 FIELD=field1 OUTPUT=output.gpkg
```

This calls **Concave hull (k-nearest neighbor)**: INPUT=input.gpkg — Input layer; KNEIGHBORS=3 — Number of neighboring points to consider (a lower number is more concave, a…; FIELD=field1 — Field (set if creating concave hulls by class); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:linestopolygons — Lines to polygons

group: Vector geometry

This algorithm generates a polygon layer using as polygon rings the lines from an input line layer. All attributes from the input line layer will be copied to the output layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
OUTPUT	sink	yes	—	Polygons

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:linestopolygons INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Lines to polygons**: INPUT=input.gpkg — Input layer; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:meanandstandarddeviationplot — Mean and standard deviation plot

group: Plots

This algorithm creates a box plot with mean and standard deviation values.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input table
NAME_FIELD	field	yes	—	Category name field
VALUE_FIELD	field	yes	—	Value field
OUTPUT	fileDestination	yes	—	Plot

Outputs: OUTPUT (outputHtml)

Example usage

```
run qgis:meanandstandarddeviationplot INPUT=input.gpkg NAME_FIELD=field1
↪  VALUE_FIELD=field1 OUTPUT=output.txt
```

This calls **Mean and standard deviation plot**: INPUT=input.gpkg — Input table; NAME_FIELD=field1 — Category name field; VALUE_FIELD=field1 — Value field; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:pointsdisplacement — Points displacement

group: Vector geometry

This algorithm offsets point features by moving nearby points by a preset amount to minimize overlapping features.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
PROXIMITY	distance	no	1.0	Minimum distance to other points
DISTANCE	distance	no	1.0	Displacement distance
HORIZONTAL	boolean	yes	—	Horizontal distribution for two point case
OUTPUT	sink	yes	—	Displaced

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:pointsdisplacement INPUT=input.gpkg HORIZONTAL=false PROXIMITY=1 DISTANCE=1
 ↪  OUTPUT=output.gpkg
```

This calls **Points displacement**: `INPUT=input.gpkg` — Input layer; `HORIZONTAL=false` — Horizontal distribution for two point case; `PROXIMITY=1` — Minimum distance to other points; `DISTANCE=1` — Displacement distance; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:polarplot` — Polar plot

group: Plots

This algorithm generates a polar plot based on the value of an input vector layer.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`VALUE_FIELD`	field	yes	—	Value field
`OUTPUT`	fileDestination	yes	—	Polar plot

Outputs: OUTPUT (outputHtml)

Example usage

```
run qgis:polarplot INPUT=input.gpkg VALUE_FIELD=field1 OUTPUT=output.txt
```

This calls **Polar plot**: `INPUT=input.gpkg` — Input layer; `VALUE_FIELD=field1` — Value field; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:randomextractwithinsubsets` — Random extract within subsets

group: Vector selection

This algorithm takes a vector layer and generates a new one that contains only a subset of the features in the input layer. The subset is defined randomly, using a percentage or count value to define the total number of features in the subset. The percentage/count value is not applied to the whole layer, but instead to each category. Categories are defined according to a given attribute, which is also specified as an input parameter for the algorithm.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`FIELD`	field	yes	—	ID field
`METHOD`	enum	no	0	Method — options: 0=Number of selected features, 1=Percentage of selected features
`NUMBER`	number	no	10	Number/percentage of selected features
`OUTPUT`	sink	yes	—	Extracted (random stratified)

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:randomextractwithinsubsets INPUT=input.gpkg FIELD=field1 METHOD=0 NUMBER=10
 ↪  OUTPUT=output.gpkg
```

This calls **Random extract within subsets**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — ID field; `METHOD=0` selects *Number of selected features*; `NUMBER=10` — Number/percentage of selected features; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:randompointsalongline` — Random points along line

group: Vector creation

This algorithm creates a point layer with a given number of points placed on the lines of the input layer. The location of each point is determined by randomly selecting a feature, then a segment of the line geometry of that feature, and finally a random position on that segment. A minimum distance between the points can be specified (Euclidean distance).

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`POINTS_NUMBER`	number	no	1	Number of points
`MIN_DISTANCE`	distance	no	0	Minimum distance between points
`OUTPUT`	sink	yes	—	Random points

Outputs: `OUTPUT` (outputVector)

Example usage

```
run qgis:randompointsalongline INPUT=input.gpkg POINTS_NUMBER=1 MIN_DISTANCE=0
 ↪  OUTPUT=output.gpkg
```

This calls **Random points along line**: `INPUT=input.gpkg` — Input layer; `POINTS_NUMBER=1` — Number of points; `MIN_DISTANCE=0` — Minimum distance between points; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:randompointsinlayerbounds` — Random points in layer bounds

group: Vector creation

This algorithm creates a new point layer with a given number of random points, all of them within the extent of a given layer. A distance factor can be specified, to avoid points being too close to each other.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`POINTS_NUMBER`	number	no	1	Number of points
`MIN_DISTANCE`	distance	no	0	Minimum distance between points
`OUTPUT`	sink	yes	—	Random points

Outputs: `OUTPUT` (outputVector)

Example usage

```
run qgis:randompointsinlayerbounds INPUT=input.gpkg POINTS_NUMBER=1 MIN_DISTANCE=0
 ↪  OUTPUT=output.gpkg
```

This calls **Random points in layer bounds**: `INPUT=input.gpkg` — Input layer; `POINTS_NUMBER=1` — Number of points; `MIN_DISTANCE=0` — Minimum distance between points; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:randompointsinsidepolygons — Random points inside polygons

group: Vector creation

This algorithm creates a new point layer with random points inside the polygons of a given layer.The number of points in each polygon can be defined as a fixed count or as a density value. The count/density value could also be taken from an attribute or an expression specified using the 'Data defined override' functionality, so it can be different for each polygon in the input layer.A minimum distance can be specified, to avoid points being too close to each other.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
STRATEGY	enum	no	0	Sampling strategy — options: 0=Points count, 1=Points density
VALUE	number	no	1	Point count or density
EXPRESSION	expression	no	—	Expression
MIN_DISTANCE	distance	no	—	Minimum distance between points
OUTPUT	sink	yes	—	Random points

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:randompointsinsidepolygons INPUT=input.gpkg STRATEGY=0 VALUE=1 EXPRESSION="value
↪  > 0" MIN_DISTANCE=10 OUTPUT=output.gpkg
```

This calls **Random points inside polygons**: INPUT=input.gpkg — Input layer; STRATEGY=0 selects *Points count*; VALUE=1 — Point count or density; EXPRESSION="value > 0" — Expression; MIN_DISTANCE=10 — Minimum distance between points; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:randomselectionwithinsubsets — Random selection within subsets

group: Vector selection

This algorithm takes a vector layer and selects a subset of its features. No new layer is generated by this algorithm. The subset is defined randomly, using a percentage or count value to define the total number of features in the subset. The percentage/count value is not applied to the whole layer, but instead to each category. Categories are defined according to a given attribute, which is also specified as an input parameter for the algorithm.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
FIELD	field	yes	—	ID field
METHOD	enum	no	0	Method — options: 0=Number of selected features, 1=Percentage of selected features
NUMBER	number	no	10	Number/percentage of selected features

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:randomselectionwithinsubsets INPUT=input.gpkg FIELD=field1 METHOD=0 NUMBER=10
```

This calls **Random selection within subsets**: INPUT=input.gpkg — Input layer; FIELD=field1 — ID field; METHOD=0 selects *Number of selected features*; NUMBER=10 — Number/percentage of selected features. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:rastercalculator — Raster calculator

group: Raster analysis

This algorithm allows performing algebraic operations using raster layers. The resulting layer will have its values computed according to an expression. The expression can contain numerical values, operators and references to any of the layers in the current project. The following functions are also supported:

- sin(), cos(), tan(), atan2(), ln(), log10() The extent, cell size, and output CRS can be defined by the user. If the extent is not specified, the minimum extent that covers selected reference layer(s) will be used. If the cell size is not specified, the minimum cell size of selected reference layer(s) will be used. If the output CRS is not specified, the CRS of the first reference layer will be used. The cell size is assumed to be the same in both X and Y axes. Layers are referred by their name as displayed in the layer list and the number of the band to use (based on 1), using the pattern 'layer_name@band number'. For instance, the first band from a layer named DEM will be referred as DEM@1. When using the calculator in the batch interface or from the console, the files to use have to be specified. The corresponding layers are referred using the base name of the file (without the full path). For instance, if using a layer at path/to/my/rasterfile.tif, the first band of that layer will be referred as rasterfile.tif@1.

Parameter	Type	Required	Default	Description
EXPRESSION	raster_calc_expression	yes	—	Expression
LAYERS	multilayer	no	—	Reference layer(s) (used for automated extent, cellsize, and CRS)
CELLSIZE	number	no	0.0	Cell size (use 0 or empty to set it automatically)
EXTENT	extent	no	—	Output extent
CRS	crs	no	—	Output CRS
OUTPUT	rasterDestination	yes	—	Output

Outputs: OUTPUT (outputRaster)

Example usage

```
run qgis:rastercalculator EXPRESSION=… LAYERS="a.gpkg;b.gpkg" CELLSIZE=0
 ↪  EXTENT=0,1000,0,1000 CRS=EPSG:6346 OUTPUT=output.tif
```

This calls **Raster calculator**: EXPRESSION=… — Expression; LAYERS="a.gpkg;b.gpkg" — Reference layer(s) (used for automated extent, cellsize, and CRS); CELLSIZE=0 — Cell size (use 0 or empty to set it automatically); EXTENT=0,1000,0,1000 — Output extent; CRS=EPSG:6346 — Output CRS; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:rasterlayerhistogram — Raster layer histogram

group: Plots

This algorithm generates a histogram with the values of a raster layer. The raster layer must have a single band.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Input layer
BAND	band	no	1	Band number
BINS	number	no	10	number of bins
OUTPUT	fileDestination	yes	—	Histogram

Outputs: OUTPUT (outputHtml)

Example usage

```
run qgis:rasterlayerhistogram INPUT=dem.tif BAND=1 BINS=10 OUTPUT=output.txt
```

This calls **Raster layer histogram**: `INPUT=dem.tif` — Input layer; `BAND=1` — Band number; `BINS=10` — number of bins; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:rectanglesovalsdiamondsvariable` — Rectangles, ovals, diamonds (variable)

group: Vector geometry

This algorithm creates rectangle, oval or diamond-shaped polygons from the input point layer using specified width, height and (optional) rotation values. Multipart inputs should be promoted to singleparts first.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`SHAPE`	enum	yes	—	Buffer shape — options: 0=Rectangles, 1=Diamonds, 2=Ovals
`WIDTH`	field	yes	—	Width field
`HEIGHT`	field	yes	—	Height field
`ROTATION`	field	no	—	Rotation field
`SEGMENTS`	number	no	36	Number of segments
`OUTPUT`	sink	yes	—	Output

Outputs: `OUTPUT` (outputVector)

Example usage

```
run qgis:rectanglesovalsdiamondsvariable INPUT=input.gpkg SHAPE=0 WIDTH=field1
↪  HEIGHT=field1 ROTATION=field1 SEGMENTS=36 OUTPUT=output.gpkg
```

This calls **Rectangles, ovals, diamonds (variable)**: `INPUT=input.gpkg` — Input layer; `SHAPE=0` selects *Rectangles*; `WIDTH=field1` — Width field; `HEIGHT=field1` — Height field; `ROTATION=field1` — Rotation field; `SEGMENTS=36` — Number of segments; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:regularpoints` — Regular points

group: Vector creation

This algorithm creates a point layer with a given number of regular points, all of them within a given extent.

Parameter	Type	Required	Default	Description
`EXTENT`	extent	yes	—	Input extent
`SPACING`	distance	no	100	Point spacing/count
`INSET`	distance	no	0.0	Initial inset from corner (LH side)
`RANDOMIZE`	boolean	no	False	Apply random offset to point spacing
`IS_SPACING`	boolean	no	True	Use point spacing
`CRS`	crs	no	ProjectCrs	Output layer CRS
`OUTPUT`	sink	yes	—	Regular points

Outputs: `OUTPUT` (outputVector)

Example usage

```
run qgis:regularpoints EXTENT=0,1000,0,1000 SPACING=100 INSET=0 RANDOMIZE=false
↪  IS_SPACING=true CRS=EPSG:6346 OUTPUT=output.gpkg
```

This calls **Regular points**: EXTENT=0,1000,0,1000 — Input extent; SPACING=100 — Point spacing/count; INSET=0 — Initial inset from corner (LH side); RANDOMIZE=false — Apply random offset to point spacing; IS_SPACING=true — Use point spacing; CRS=EPSG:6346 — Output layer CRS; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:relief — Relief

group: Raster terrain analysis

This algorithm creates a shaded relief layer from digital elevation data.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Elevation layer
Z_FACTOR	number	no	1.0	Z factor
AUTO_COLORS	boolean	no	False	Generate relief classes automatically
COLORS	relief_colors	no	—	Relief colors
OUTPUT	rasterDestination	yes	—	Relief
FREQUENCY_DISTRIBUTION	fileDestination	no	—	Frequency distribution

Outputs: OUTPUT (outputRaster), FREQUENCY_DISTRIBUTION (outputFile)

Example usage

```
run qgis:relief INPUT=dem.tif Z_FACTOR=1 AUTO_COLORS=false OUTPUT=output.tif
```

This calls **Relief**: INPUT=dem.tif — Elevation layer; Z_FACTOR=1 — Z factor; AUTO_COLORS=false — Generate relief classes automatically; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:scatter3dplot — Vector layer scatterplot 3D

group: Plots

This algorithm creates a 3D scatter plot for a vector layer.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
XFIELD	field	yes	—	X attribute
YFIELD	field	yes	—	Y attribute
ZFIELD	field	yes	—	Z attribute
TITLE	string	no	—	Title
XAXIS_TITLE	string	no	—	X-axis title
YAXIS_TITLE	string	no	—	Y-axis title
ZAXIS_TITLE	string	no	—	Z-axis title
OUTPUT	fileDestination	yes	—	Scatterplot 3D

Outputs: OUTPUT (outputHtml)

Example usage

```
run qgis:scatter3dplot INPUT=input.gpkg XFIELD=field1 YFIELD=field1 ZFIELD=field1
↪  TITLE=value XAXIS_TITLE=value YAXIS_TITLE=value OUTPUT=output.txt
```

This calls **Vector layer scatterplot 3D**: INPUT=input.gpkg — Input layer; XFIELD=field1 — X attribute; YFIELD=field1 — Y attribute; ZFIELD=field1 — Z attribute; TITLE=value — Title; XAXIS_TITLE=value — X-axis title; YAXIS_TITLE=value — Y-axis title; OUTPUT=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:selectbyattribute — Select by attribute

group: Vector selection

This algorithm creates a selection in a vector layer. The criteria for selected features is defined based on the values of an attribute from the input layer.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
FIELD	field	yes	—	Selection attribute
OPERATOR	enum	no	0	Operator — options: 0==, 1=≠, 2=>, 3=≥, 4=<, 5=≤, 6=begins with, 7=contains, 8=is null, 9=is not null, 10=does not contain
VALUE	string	no	—	Value
METHOD	enum	no	0	Modify current selection by — options: 0=creating new selection, 1=adding to current selection, 2=removing from current selection, 3=selecting within current selection

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:selectbyattribute INPUT=input.gpkg FIELD=field1 OPERATOR=0 VALUE=value METHOD=0
```

This calls **Select by attribute**: INPUT=input.gpkg — Input layer; FIELD=field1 — Selection attribute; OPERATOR=0 selects =; VALUE=value — Value; METHOD=0 selects *creating new selection*. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:selectbyexpression — Select by expression

group: Vector selection

This algorithm creates a selection in a vector layer. The criteria for selecting features is based on a QGIS expression. For help with QGIS expression functions, see the inbuilt help for specific functions which is available in the expression builder.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Input layer
EXPRESSION	expression	yes	—	Expression
METHOD	enum	no	0	Modify current selection by — options: 0=creating new selection, 1=adding to current selection, 2=removing from current selection, 3=selecting within current selection

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:selectbyexpression INPUT=input.gpkg EXPRESSION="value > 0" METHOD=0
```

This calls **Select by expression**: INPUT=input.gpkg — Input layer; EXPRESSION="value > 0" — Expression; METHOD=0 selects *creating new selection*. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:setstyleforrasterlayer — Set style for raster layer

group: Raster tools

This algorithm sets the style of a raster layer. The style must be defined in a QML file.

Parameter	Type	Required	Default	Description
INPUT	raster	yes	—	Raster layer

Parameter	Type	Required	Default	Description
STYLE	file	yes	—	Style file

Outputs: INPUT (outputRaster)

Example usage

```
run qgis:setstyleforrasterlayer INPUT=dem.tif STYLE=input.dat
```

This calls **Set style for raster layer**: INPUT=dem.tif — Raster layer; STYLE=input.dat — Style file. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:setstyleforvectorlayer — Set style for vector layer

group: Vector general

This algorithm sets the style of a vector layer. The style must be defined in a QML file.

Parameter	Type	Required	Default	Description
INPUT	vector	yes	—	Vector layer
STYLE	file	yes	—	Style file

Outputs: INPUT (outputVector)

Example usage

```
run qgis:setstyleforvectorlayer INPUT=input.gpkg STYLE=input.dat
```

This calls **Set style for vector layer**: INPUT=input.gpkg — Vector layer; STYLE=input.dat — Style file. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:statisticsbycategories — Statistics by categories

group: Vector analysis

This algorithm calculates statistics of fields depending on a parent class.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input vector layer
VALUES_FIELD_NAME	field	no	—	Field to calculate statistics on (if empty, only count is calculated)
CATEGORIES_FIELD_NAME	field	yes	—	Field(s) with categories
OUTPUT	sink	yes	—	Statistics by category

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:statisticsbycategories INPUT=input.gpkg CATEGORIES_FIELD_NAME=field1
↪  VALUES_FIELD_NAME=field1 OUTPUT=output.gpkg
```

This calls **Statistics by categories**: INPUT=input.gpkg — Input vector layer; CATEGORIES_FIELD_NAME=field1 — Field(s) with categories; VALUES_FIELD_NAME=field1 — Field to calculate statistics on (if empty, only count is calculated); OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:texttofloat — Text to float

group: Vector table

This algorithm modifies the type of a given attribute in a vector layer, converting a text attribute containing numeric strings into a numeric attribute.

Parameter	Type	Required	Default	Description
INPUT	source	yes	—	Input layer
FIELD	field	yes	—	Text attribute to convert to float
OUTPUT	sink	yes	—	Float from text

Outputs: OUTPUT (outputVector)

Example usage

```
run qgis:texttofloat INPUT=input.gpkg FIELD=field1 OUTPUT=output.gpkg
```

This calls **Text to float**: INPUT=input.gpkg — Input layer; FIELD=field1 — Text attribute to convert to float; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

qgis:tininterpolation — TIN interpolation

group: Interpolation

This algorithm generates a Triangulated Irregular Network (TIN) interpolation of a point vector layer. With the TIN method you can create a surface formed by triangles of nearest neighbor points.To do this, circumcircles around selected sample points are created and their intersections are connected to a network of non overlapping and as compact as possible triangles.The resulting surfaces are not smooth. The algorithm creates both the raster layer of the interpolated values and the vector line layer with the triangulation boundaries.

Parameter	Type	Required	Default	Description
INTERPOLATION_-DATA	idw_interpolation_data	yes	—	Input layer(s)
METHOD	enum	no	0	Interpolation method — options: 0=Linear, 1=Clough-Toucher (cubic)
EXTENT	extent	yes	—	Extent
PIXEL_SIZE	number	no	0.1	Output raster size
COLUMNS	number	no	—	Number of columns
ROWS	number	no	—	Number of rows
OUTPUT	rasterDestination	yes	—	Interpolated
TRIANGULATION	sink	no	—	Triangulation

Outputs: OUTPUT (outputRaster), TRIANGULATION (outputVector)

Example usage

```
run qgis:tininterpolation INTERPOLATION_DATA=… EXTENT=0,1000,0,1000 METHOD=0
 ↪  PIXEL_SIZE=0.1 COLUMNS=10 ROWS=10 OUTPUT=output.tif
```

This calls **TIN interpolation**: INTERPOLATION_DATA=… — Input layer(s); EXTENT=0,1000,0,1000 — Extent; METHOD=0 selects *Linear*; PIXEL_SIZE=0.1 — Output raster size; COLUMNS=10 — Number of columns; ROWS=10 — Number of rows; OUTPUT=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:topologicalcoloring` — Topological coloring

group: Cartography

This algorithm assigns a color index to polygon features in such a way that no adjacent polygons share the same color index, whilst minimizing the number of colors required. An optional minimum distance between features assigned the same color can be set to prevent nearby (but non-touching) features from being assigned equal colors. The algorithm allows choice of method to use when assigning colors. The default method attempts to assign colors so that the count of features assigned to each individual color index is balanced. The 'by assigned area' mode instead assigns colors so that the total area of features assigned to each color is balanced. This mode can be useful to help avoid large features resulting in one of the colors appearing more dominant on a colored map. The 'by distance between colors' mode will assign colors in order to maximize the distance between features of the same color. This mode helps to create a more uniform distribution of colors across a map. A minimum number of colors can be specified if desired. The color index is saved to a new attribute named color_id.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`MIN_COLORS`	number	no	4	Minimum number of colors
`MIN_DISTANCE`	distance	no	0.0	Minimum distance between features
`BALANCE`	enum	no	0	Balance color assignment — options: 0=By feature count, 1=By assigned area, 2=By distance between colors
`OUTPUT`	sink	yes	—	Colored

Outputs: `OUTPUT` (outputVector)

Example usage

```
run qgis:topologicalcoloring INPUT=input.gpkg MIN_COLORS=4 MIN_DISTANCE=0 BALANCE=0
 ↪  OUTPUT=output.gpkg
```

This calls **Topological coloring**: `INPUT=input.gpkg` — Input layer; `MIN_COLORS=4` — Minimum number of colors; `MIN_DISTANCE=0` — Minimum distance between features; `BALANCE=0` selects *By feature count*; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:variabledistancebuffer` — Variable distance buffer

group: Vector geometry

This algorithm computes a buffer area for all the features in an input layer. The size of the buffer for a given feature is defined by an attribute, so it allows different features to have different buffer sizes.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`FIELD`	field	yes	—	Distance field
`SEGMENTS`	number	no	5	Segments
`DISSOLVE`	boolean	no	False	Dissolve result
`END_CAP_STYLE`	enum	no	0	End cap style — options: 0=Round, 1=Flat, 2=Square
`JOIN_STYLE`	enum	no	0	Join style — options: 0=Round, 1=Miter, 2=Bevel
`MITER_LIMIT`	number	no	2	Miter limit
`OUTPUT`	sink	yes	—	Buffer

Outputs: `OUTPUT` (outputVector)

Example usage

```
run qgis:variabledistancebuffer INPUT=input.gpkg FIELD=field1 SEGMENTS=5 DISSOLVE=false
 ↪  END_CAP_STYLE=0 JOIN_STYLE=0 MITER_LIMIT=2 OUTPUT=output.gpkg
```

This calls **Variable distance buffer**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — Distance field; `SEGMENTS=5` — Segments; `DISSOLVE=false` — Dissolve result; `END_CAP_STYLE=0` selects *Round*; `JOIN_STYLE=0` selects *Round*; `MITER_LIMIT=2` — Miter limit; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:vectorlayerhistogram` — Vector layer histogram

group: Plots

This algorithm generates a histogram with the values of the attribute of a vector layer. The attribute to use for computing the histogram must be a numeric attribute.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`FIELD`	field	yes	—	Attribute
`BINS`	number	no	10	number of bins
`OUTPUT`	fileDestination	yes	—	Histogram

Outputs: `OUTPUT` (outputHtml)

Example usage

```
run qgis:vectorlayerhistogram INPUT=input.gpkg FIELD=field1 BINS=10 OUTPUT=output.txt
```

This calls **Vector layer histogram**: `INPUT=input.gpkg` — Input layer; `FIELD=field1` — Attribute; `BINS=10` — number of bins; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`qgis:vectorlayerscatterplot` — Vector layer scatterplot

group: Plots

This algorithm creates a simple X - Y scatter plot for a vector layer.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`XFIELD`	field	yes	—	X attribute
`YFIELD`	field	yes	—	Y attribute
`HOVERTEXT`	expression	no	—	Hover text
`TITLE`	string	no	—	Title
`XAXIS_TITLE`	string	no	—	X-axis title
`YAXIS_TITLE`	string	no	—	Y-axis title
`XAXIS_LOG`	boolean	no	False	Use logarithmic scale for x-axis
`YAXIS_LOG`	boolean	no	False	Use logarithmic scale for y-axis
`OUTPUT`	fileDestination	yes	—	Scatterplot

Outputs: `OUTPUT` (outputHtml)

Example usage

```
run qgis:vectorlayerscatterplot INPUT=input.gpkg XFIELD=field1 YFIELD=field1
 ↪  HOVERTEXT="value > 0" TITLE=value XAXIS_TITLE=value YAXIS_TITLE=value
 ↪  OUTPUT=output.txt
```

This calls **Vector layer scatterplot**: `INPUT=input.gpkg` — Input layer; `XFIELD=field1` — X attribute; `YFIELD=field1` — Y attribute; `HOVERTEXT="value > 0"` — Hover text; `TITLE=value` — Title; `XAXIS_TITLE=value` — X-axis title; `YAXIS_TITLE=value` — Y-axis title; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal: algorithms

pdal — PDAL point-cloud algorithms — 24 algorithms (QGIS 4.0.3). Auto-generated by `scripts/gen_algorithms.py`; see the index.

Call any of these with `run <id> KEY=value …` (the **Name** column is the KEY); each entry includes a worked **Example usage**. A ★ marks an algorithm with a friendly niva alias verb.

pdal:assignprojection — Assign projection

group: Point cloud data management

This algorithm assigns point cloud CRS if it is not present or wrong.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
CRS	crs	no	EPSG:4326	Desired CRS
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	pointCloudDestination	yes	—	Output layer

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:assignprojection INPUT=cloud.copc.laz CRS=EPSG:6346 OUTPUT=output.laz
```

This calls **Assign projection**: INPUT=cloud.copc.laz — Input layer; CRS=EPSG:6346 — Desired CRS; OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:boundary — Boundary

group: Point cloud extraction

This algorithm exports a polygon file containing point cloud layer boundary. It may contain holes and it may be a multi-part polygon.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
RESOLUTION	number	no	—	Resolution of cells used to calculate boundary
THRESHOLD	number	no	—	Minimal number of points in a cell to consider cell occupied
FILTER_EXPRESSION *(adv)*	expression	no	—	Filter expression
FILTER_EXTENT *(adv)*	extent	no	—	Cropping extent
OUTPUT	vectorDestination	yes	—	Boundary

Outputs: OUTPUT (outputVector)

Example usage

```
run pdal:boundary INPUT=cloud.copc.laz RESOLUTION=10 THRESHOLD=10 OUTPUT=output.gpkg
```

This calls **Boundary**: INPUT=cloud.copc.laz — Input layer; RESOLUTION=10 — Resolution of cells used to calculate boundary; THRESHOLD=10 — Minimal number of points in a cell to consider cell occupied; OUTPUT=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:classifyground — Classify ground points

group: Point cloud data management

This algorithm classifies ground points using the Simple Morphological Filter (SMRF) algorithm.

Cell Size is the cell size for processing grid in map units, where smaller values give finer detail but may increase noise. Scalar is the threshold for steeper slopes; a higher value is needed if the terrain is rough, otherwise real ground might be misclassified. Slope is the slope threshold measured as rise over run, indicating how much slope is tolerated as ground and should be higher for steep terrain. Threshold is the elevation threshold for separating ground from objects; higher values allow larger deviations from the ground. Window Size is the maximum filter window size, where higher values better identify large buildings or objects while smaller values protect smaller features.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
CELL_SIZE	number	no	1.0	Grid Cell Size
SCALAR	number	no	1.25	Scalar
SLOPE	number	no	0.15	Slope
THRESHOLD	number	no	0.5	Threshold
WINDOW_SIZE	number	no	18.0	Window Size
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	pointCloudDestination	yes	—	Classified Ground

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:classifyground INPUT=cloud.copc.laz CELL_SIZE=1 SCALAR=1.25 SLOPE=0.15
↪  THRESHOLD=0.5 WINDOW_SIZE=18 OUTPUT=output.laz
```

This calls **Classify ground points**: INPUT=cloud.copc.laz — Input layer; CELL_SIZE=1 — Grid Cell Size; SCALAR=1.25 — Scalar; SLOPE=0.15 — Slope; THRESHOLD=0.5 — Threshold; WINDOW_SIZE=18 — Window Size; OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:clip — Clip point cloud

group: Point cloud data management

This algorithm clips point cloud with clipping polygons, the resulting point cloud contains points that are inside these polygons.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
OVERLAY	vector	yes	—	Clipping polygons
FILTER_EXPRESSION *(adv)*	expression	no	—	Filter expression
FILTER_EXTENT *(adv)*	extent	no	—	Cropping extent
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	pointCloudDestination	yes	—	Clipped

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:clip INPUT=cloud.copc.laz OVERLAY=input.gpkg OUTPUT=output.laz
```

This calls **Clip point cloud**: `INPUT=cloud.copc.laz` — Input layer; `OVERLAY=input.gpkg` — Clipping polygons; `OUTPUT=output.laz` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:compare` — Compare point clouds

group: Point cloud data management

This algorithm compares two point clouds using an M3C2 (Multiscale Model-to-Model Cloud Comparison) algorithm and outputs a subset (filtered using Poisson sampling, based on Subsampling cell size parameter) of the original point cloud.

The output point cloud will have several new dimensions: m3c2_distance, m3c2_uncertainty, m3c2_significant, m3c2_std_dev1, m3c2_std_dev2, m3c2_count1 and m3c2_count2.

The M3C2 algorithm calculates the distance between two point clouds by considering the local orientation of the surface. It estimates surface normals at a user-defined scale and measures the average distance between clouds within a cylindrical projection along those normals.

The approach is highly robust against sensor noise and surface roughness, making it the standard for detecting change in complex natural environments. It also provides a sign (indicating whether a surface has moved in or out) and a statistically significant level of detection to distinguish real change from measurement error.

References: Lague, Dimitri, Nicolas Brodu, and Jérôme Leroux. Accurate 3D Comparison of Complex Topography with Terrestrial Laser Scanner: Application to the Rangitikei Canyon (N-Z). arXiv, 2013, https://arxiv.org/abs/1302.1183.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
INPUT_COMPARE	pointcloud	yes	—	Compare layer
SUBSAMPLING_CELL_SIZE	distance	no	0.0	Subsampling cell size
NORMAL_RADIUS	distance	no	2.0	Normal Radius
CYLINDER_RADIUS	distance	no	2.0	Cylinder Radius
CYLINDER_HALF_LENGTH	distance	no	5.0	Cylinder Half-Length
REGISTRATION_ERROR	number	no	0.0	Registration Error
CYLINDER_ORIENTATION	enum	no	0	Cylinder Orientation — options: 0=Up, 1=Origin, 2=None
FILTER_EXPRESSION *(adv)*	expression	no	—	Filter expression
FILTER_EXTENT *(adv)*	extent	no	—	Cropping extent
OUTPUT	pointCloudDestination	yes	—	Comparison Point Cloud

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:compare INPUT=cloud.copc.laz INPUT_COMPARE=cloud.copc.laz SUBSAMPLING_CELL_SIZE=0
↪ NORMAL_RADIUS=2 CYLINDER_RADIUS=2 CYLINDER_HALF_LENGTH=5 REGISTRATION_ERROR=0
↪ OUTPUT=output.laz
```

This calls **Compare point clouds**: `INPUT=cloud.copc.laz` — Input layer; `INPUT_COMPARE=cloud.copc.laz` — Compare layer; `SUBSAMPLING_CELL_SIZE=0` — Subsampling cell size; `NORMAL_RADIUS=2` — Normal Radius; `CYLINDER_RADIUS=2` — Cylinder Radius; `CYLINDER_HALF_LENGTH=5` — Cylinder Half-Length; `REGISTRATION_ERROR=0` — Registration Error; `OUTPUT=output.laz` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:convertformat` — Convert point cloud format

group: Point cloud conversion

This algorithm converts point cloud to a different file format, e. g. creates compressed LAZ.

Parameter	Type	Required	Default	Description
`INPUT`	pointcloud	yes	—	Input layer
`VPC_OUTPUT_FORMAT *(adv)*`	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
`OUTPUT`	pointCloudDestination	yes	—	Converted

Outputs: `OUTPUT` (outputPointCloud)

Example usage

```
run pdal:convertformat INPUT=cloud.copc.laz OUTPUT=output.laz
```

This calls **Convert point cloud format**: `INPUT=cloud.copc.laz` — Input layer; `OUTPUT=output.laz` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:createcopc` — Create COPC

group: Point cloud data management

This algorithm creates a COPC file for each input point cloud file.

Parameter	Type	Required	Default	Description
`LAYERS`	multilayer	yes	—	Input layers
`OUTPUT`	folderDestination	no	—	Output directory

Outputs: `OUTPUT` (outputFolder), `OUTPUT_LAYERS` (outputMultilayer)

Example usage

```
run pdal:createcopc LAYERS="a.gpkg;b.gpkg" OUTPUT=output/
```

This calls **Create COPC**: `LAYERS="a.gpkg;b.gpkg"` — Input layers; `OUTPUT=output/` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:density` — Point cloud density

group: Point cloud extraction

This algorithm exports a raster file where each cell contains number of points that are in that cell's area.

Parameter	Type	Required	Default	Description
`INPUT`	pointcloud	yes	—	Input layer
`RESOLUTION`	number	no	1	Resolution of the density raster
`TILE_SIZE`	number	no	1000	Tile size for parallel runs
`FILTER_EXPRESSION *(adv)*`	expression	no	—	Filter expression
`FILTER_EXTENT *(adv)*`	extent	no	—	Cropping extent
`ORIGIN_X *(adv)*`	number	no	—	X origin of a tile for parallel runs
`ORIGIN_Y *(adv)*`	number	no	—	Y origin of a tile for parallel runs
`OUTPUT`	rasterDestination	yes	—	Density

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run pdal:density INPUT=cloud.copc.laz RESOLUTION=1 TILE_SIZE=1000 OUTPUT=output.tif
```

This calls **Point cloud density**: `INPUT=cloud.copc.laz` — Input layer; `RESOLUTION=1` — Resolution of the density raster; `TILE_SIZE=1000` — Tile size for parallel runs; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:exportraster` — Export point cloud to raster

group: Point cloud conversion

This algorithm exports point cloud data to a 2D raster grid having cell size of given resolution, writing values from the specified attribute.

Parameter	Type	Required	Default	Description
`INPUT`	pointcloud	yes	—	Input layer
`ATTRIBUTE`	attribute	no	Z	Attribute
`RESOLUTION`	number	no	1	Resolution of the density raster
`TILE_SIZE`	number	no	1000	Tile size for parallel runs
`FILTER_EXPRESSION *(adv)*`	expression	no	—	Filter expression
`FILTER_EXTENT *(adv)*`	extent	no	—	Cropping extent
`ORIGIN_X *(adv)*`	number	no	—	X origin of a tile for parallel runs
`ORIGIN_Y *(adv)*`	number	no	—	Y origin of a tile for parallel runs
`OUTPUT`	rasterDestination	yes	—	Exported

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run pdal:exportraster INPUT=cloud.copc.laz ATTRIBUTE=Z RESOLUTION=1 TILE_SIZE=1000
 ↪  OUTPUT=output.tif
```

This calls **Export point cloud to raster**: `INPUT=cloud.copc.laz` — Input layer; `ATTRIBUTE=Z` — Attribute; `RESOLUTION=1` — Resolution of the density raster; `TILE_SIZE=1000` — Tile size for parallel runs; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:exportrastertin` — Export point cloud to raster (using triangulation)

group: Point cloud conversion

This algorithm exports point cloud data to a 2D raster grid using a triangulation of points and then interpolating cell values from triangles.

Parameter	Type	Required	Default	Description
`INPUT`	pointcloud	yes	—	Input layer
`RESOLUTION`	number	no	1	Resolution of the density raster
`TILE_SIZE`	number	no	1000	Tile size for parallel runs
`FILTER_EXPRESSION *(adv)*`	expression	no	—	Filter expression
`FILTER_EXTENT *(adv)*`	extent	no	—	Cropping extent
`ORIGIN_X *(adv)*`	number	no	—	X origin of a tile for parallel runs
`ORIGIN_Y *(adv)*`	number	no	—	Y origin of a tile for parallel runs
`OUTPUT`	rasterDestination	yes	—	Exported (using triangulation)

Outputs: `OUTPUT` (outputRaster)

Example usage

```
run pdal:exportrastertin INPUT=cloud.copc.laz RESOLUTION=1 TILE_SIZE=1000
 ↪  OUTPUT=output.tif
```

This calls **Export point cloud to raster (using triangulation)**: `INPUT=cloud.copc.laz` — Input layer; `RESOLUTION=1` — Resolution of the density raster; `TILE_SIZE=1000` — Tile size for parallel runs; `OUTPUT=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:exportvector` — Export point cloud to vector

group: Point cloud conversion

This algorithm exports point cloud data to a vector layer with 3D points (a GeoPackage), optionally with extra attributes.

Parameter	Type	Required	Default	Description
`INPUT`	pointcloud	yes	—	Input layer
`ATTRIBUTE`	attribute	no	—	Attribute
`FILTER_EXPRESSION *(adv)*`	expression	no	—	Filter expression
`FILTER_EXTENT *(adv)*`	extent	no	—	Cropping extent
`OUTPUT`	vectorDestination	yes	—	Exported

Outputs: `OUTPUT` (outputVector)

Example usage

```
run pdal:exportvector INPUT=cloud.copc.laz ATTRIBUTE=field1 OUTPUT=output.gpkg
```

This calls **Export point cloud to vector**: `INPUT=cloud.copc.laz` — Input layer; `ATTRIBUTE=field1` — Attribute; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:filter` — Filter point cloud

group: Point cloud extraction

This algorithm extracts point from the input point cloud which match PDAL expression and/or are inside of a cropping rectangle.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
FILTER_EXPRESSION	expression	yes	—	Filter expression
FILTER_EXTENT	extent	no	—	Cropping extent
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	pointCloudDestination	yes	—	Filtered

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:filter INPUT=cloud.copc.laz FILTER_EXPRESSION="value > 0"
 ↪  FILTER_EXTENT=0,1000,0,1000 OUTPUT=output.laz
```

This calls **Filter point cloud**: INPUT=cloud.copc.laz — Input layer; FILTER_EXPRESSION="value > 0" — Filter expression; FILTER_EXTENT=0,1000,0,1000 — Cropping extent; OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:filternoiseradius — Filter noise (using radius)

group: Point cloud data management

This algorithm filters noise in a point cloud using radius algorithm.

Points are marked as noise if they have fewer than the minimum number of neighbors within the specified radius.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
REMOVE_NOISE_POINTS	boolean	no	False	Remove noise points
MIN_K	number	no	2	Minimum number of neighbors in radius
RADIUS	number	no	1.0	Radius
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	pointCloudDestination	yes	—	Filtered noise

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:filternoiseradius INPUT=cloud.copc.laz REMOVE_NOISE_POINTS=false MIN_K=2 RADIUS=1
 ↪  OUTPUT=output.laz
```

This calls **Filter noise (using radius)**: INPUT=cloud.copc.laz — Input layer; REMOVE_NOISE_POINTS=false — Remove noise points; MIN_K=2 — Minimum number of neighbors in radius; RADIUS=1 — Radius; OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:filternoisestatistical — Filter noise

group: Point cloud data management

This algorithm filters noise in a point cloud using a statistical outlier removal algorithm.

For each point, the algorithm computes the mean distance to its K nearest neighbors. Points whose mean distance exceeds a threshold (mean distance + multiplier × standard deviation) are classified as noise.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
REMOVE_NOISE_POINTS	boolean	no	False	Remove noise points
MEAN_K	number	no	8	Mean number of neighbors
MULTIPLIER	number	no	2.0	Standard deviation multiplier
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	pointCloudDestination	yes	—	Filtered noise

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:filternoisestatistical INPUT=cloud.copc.laz REMOVE_NOISE_POINTS=false MEAN_K=8
↪  MULTIPLIER=2 OUTPUT=output.laz
```

This calls **Filter noise**: INPUT=cloud.copc.laz — Input layer; REMOVE_NOISE_POINTS=false — Remove noise points; MEAN_K=8 — Mean number of neighbors; MULTIPLIER=2 — Standard deviation multiplier; OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:heightabovegroundbynearestneighbor — Height above ground

group: Point cloud data management

This algorithm calculates the height of points above the ground surface in a point cloud using a nearest neighbor algorithm.

For each point, the algorithm finds the specified number of nearest ground-classified points (classification value 2) and interpolates the ground elevation from them using inverse distance weighting.

The output adds a HeightAboveGround dimension to the point cloud. If 'Replace Z values' is enabled, the Z coordinate will be replaced with the height above ground value.

Maximum Distance parameter can limit the search radius (0 = no limit).

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
REPLACE_Z	boolean	no	True	Replace Z values with height above ground
COUNT	number	no	1	Number of neighbors for terrain interpolation
MAX_DISTANCE	number	no	0.0	Maximum search distance
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	pointCloudDestination	yes	—	Height above ground

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:heightabovegroundbynearestneighbor INPUT=cloud.copc.laz REPLACE_Z=true COUNT=1
↪  MAX_DISTANCE=0 OUTPUT=output.laz
```

This calls **Height above ground**: INPUT=cloud.copc.laz — Input layer; REPLACE_Z=true — Replace Z values with height above ground; COUNT=1 — Number of neighbors for terrain interpolation; MAX_DISTANCE=0 — Maximum search distance; OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:heightabovegroundtriangulation` — Height above ground (using triangulation)

group: Point cloud data management

This algorithm calculates the height of points above the ground surface in a point cloud using a Delaunay triangulation algorithm.

The algorithm uses ground-classified points (classification value 2) to create a triangulated irregular network (TIN) from specified number of neighbors, then computes the height above this surface for all points.

The output adds a HeightAboveGround dimension to the point cloud. If 'Replace Z values' is enabled, the Z coordinate will be replaced with the height above ground value.

Parameter	Type	Required	Default	Description
`INPUT`	pointcloud	yes	—	Input layer
`REPLACE_Z`	boolean	no	True	Replace Z values with height above ground
`COUNT`	number	no	10	Number of ground neighbors for terrain construction
`VPC_OUTPUT_FORMAT *(adv)*`	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
`OUTPUT`	pointCloudDestination	yes	—	Height above ground

Outputs: `OUTPUT` (outputPointCloud)

Example usage

```
run pdal:heightabovegroundtriangulation INPUT=cloud.copc.laz REPLACE_Z=true COUNT=10
↪  OUTPUT=output.laz
```

This calls **Height above ground (using triangulation)**: `INPUT=cloud.copc.laz` — Input layer; `REPLACE_Z=true` — Replace Z values with height above ground; `COUNT=10` — Number of ground neighbors for terrain construction; `OUTPUT=output.laz` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:info` — Point cloud information

group: Point cloud data management

This algorithm outputs basic metadata from the point cloud file.

Parameter	Type	Required	Default	Description
`INPUT`	pointcloud	yes	—	Input layer
`OUTPUT`	fileDestination	yes	—	Layer information

Outputs: `OUTPUT` (outputHtml)

Example usage

```
run pdal:info INPUT=cloud.copc.laz OUTPUT=output.txt
```

This calls **Point cloud information**: `INPUT=cloud.copc.laz` — Input layer; `OUTPUT=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:merge — Merge point cloud

group: Point cloud data management

This algorithm merges multiple point cloud files to a single one.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layers
FILTER_EXPRESSION *(adv)*	expression	no	—	Filter expression
FILTER_EXTENT *(adv)*	extent	no	—	Cropping extent
OUTPUT	pointCloudDestination	yes	—	Merged

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:merge LAYERS="a.gpkg;b.gpkg" OUTPUT=output.laz
```

This calls **Merge point cloud**: LAYERS="a.gpkg;b.gpkg" — Input layers; OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:reproject — Reproject point cloud

group: Point cloud data management

This algorithm reprojects point cloud to a different coordinate reference system.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
CRS	crs	no	EPSG:4326	Target CRS
OPERATION *(adv)*	coordinateoperation	no	—	Coordinate operation
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	pointCloudDestination	yes	—	Reprojected

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:reproject INPUT=cloud.copc.laz CRS=EPSG:6346 OUTPUT=output.laz
```

This calls **Reproject point cloud**: INPUT=cloud.copc.laz — Input layer; CRS=EPSG:6346 — Target CRS; OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:thinbydecimate — Thin (by skipping points)

group: Point cloud data management

This algorithm creates a thinned version of the point cloud by keeping only every N-th point.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
POINTS_NUMBER	number	no	1	Number of points to skip
FILTER_EXPRESSION *(adv)*	expression	no	—	Filter expression
FILTER_EXTENT *(adv)*	extent	no	—	Cropping extent
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS

Parameter	Type	Required	Default	Description
OUTPUT	pointCloudDestination	yes	—	Thinned (by decimation)

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:thinbydecimate INPUT=cloud.copc.laz POINTS_NUMBER=1 OUTPUT=output.laz
```

This calls **Thin (by skipping points)**: INPUT=cloud.copc.laz — Input layer; POINTS_NUMBER=1 — Number of points to skip; OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:thinbyradius — Thin (by sampling radius)

group: Point cloud data management

This algorithm creates a thinned version of the point cloud by performing sampling by distance point.

Parameter	Type	Required	Default	Description
INPUT	pointcloud	yes	—	Input layer
SAMPLING_RADIUS	number	no	1.0	Sampling radius (in map units)
FILTER_EXPRESSION *(adv)*	expression	no	—	Filter expression
FILTER_EXTENT *(adv)*	extent	no	—	Cropping extent
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	pointCloudDestination	yes	—	Thinned (by radius)

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:thinbyradius INPUT=cloud.copc.laz SAMPLING_RADIUS=1 OUTPUT=output.laz
```

This calls **Thin (by sampling radius)**: INPUT=cloud.copc.laz — Input layer; SAMPLING_RADIUS=1 — Sampling radius (in map units); OUTPUT=output.laz is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

pdal:tile — Create tiles from point cloud

group: Point cloud data management

This algorithm creates tiles from an input point cloud.

Parameter	Type	Required	Default	Description
LAYERS	multilayer	yes	—	Input layers
LENGTH	number	no	1000.0	Tile length
CRS *(adv)*	crs	no	—	Assign CRS
VPC_OUTPUT_FORMAT *(adv)*	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
OUTPUT	folderDestination	yes	—	Output directory

Outputs: OUTPUT (outputFolder)

Example usage

```
run pdal:tile LAYERS="a.gpkg;b.gpkg" LENGTH=1000 OUTPUT=output/
```

This calls **Create tiles from point cloud**: `LAYERS="a.gpkg;b.gpkg"` — Input layers; `LENGTH=1000` — Tile length; `OUTPUT=output/` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:transformpointcloud` — Transform point cloud

group: Point cloud data management

This algorithm transforms point cloud coordinates using translation, rotation, and scaling operations using a 4x4 transformation matrix.

The algorithm applies transformations in the following order: scaling, rotation (using Euler angles), then translation.

Rotation angles are specified in degrees around the X, Y, and Z axes.

All parameters are combined into a 4×4 transformation matrix that is passed to PDAL Wrench.

Parameter	Type	Required	Default	Description
`INPUT`	pointcloud	yes	—	Input layer
`TRANSLATE_X`	number	no	0.0	X Translation
`TRANSLATE_Y`	number	no	0.0	Y Translation
`TRANSLATE_Z`	number	no	0.0	Z Translation
`SCALE_X`	number	no	1.0	X Scale
`SCALE_Y`	number	no	1.0	Y Scale
`SCALE_Z`	number	no	1.0	Z Scale
`ROTATE_X`	number	no	0.0	X Rotation
`ROTATE_Y`	number	no	0.0	Y Rotation
`ROTATE_Z`	number	no	0.0	Z Rotation
`VPC_OUTPUT_FORMAT *(adv)*`	enum	no	COPC	VPC Output Format — options: 0=COPC, 1=LAZ, 2=LAS
`OUTPUT`	pointCloudDestination	yes	—	Transformed

Outputs: `OUTPUT` (outputPointCloud)

Example usage

```
run pdal:transformpointcloud INPUT=cloud.copc.laz TRANSLATE_X=0 TRANSLATE_Y=0
 ↪  TRANSLATE_Z=0 SCALE_X=1 SCALE_Y=1 SCALE_Z=1 OUTPUT=output.laz
```

This calls **Transform point cloud**: `INPUT=cloud.copc.laz` — Input layer; `TRANSLATE_X=0` — X Translation; `TRANSLATE_Y=0` — Y Translation; `TRANSLATE_Z=0` — Z Translation; `SCALE_X=1` — X Scale; `SCALE_Y=1` — Y Scale; `SCALE_Z=1` — Z Scale; `OUTPUT=output.laz` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`pdal:virtualpointcloud` — Build virtual point cloud (VPC)

group: Point cloud data management

This algorithm creates a virtual point cloud from input data.

Parameter	Type	Required	Default	Description
`LAYERS`	multilayer	yes	—	Input layers
`BOUNDARY`	boolean	no	False	Calculate boundary polygons
`STATISTICS`	boolean	no	False	Calculate statistics
`OVERVIEW`	boolean	no	False	Build overview point cloud
`CONVERT_COPC`	boolean	no	False	Convert individual files to COPC format
`OUTPUT`	pointCloudDestination	yes	—	Virtual point cloud

Outputs: OUTPUT (outputPointCloud)

Example usage

```
run pdal:virtualpointcloud LAYERS="a.gpkg;b.gpkg" BOUNDARY=false STATISTICS=false
 ↪  OVERVIEW=false CONVERT_COPC=false OUTPUT=output.laz
```

This calls **Build virtual point cloud (VPC)**: `LAYERS="a.gpkg;b.gpkg"` — Input layers; `BOUNDARY=false` — Calculate boundary polygons; `STATISTICS=false` — Calculate statistics; `OVERVIEW=false` — Build overview point cloud; `CONVERT_COPC=false` — Convert individual files to COPC format; `OUTPUT=output.laz` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

3d: algorithms

3d — 3D algorithms — 1 algorithms (QGIS 4.0.3). Auto-generated by `scripts/gen_algorithms.py`; see the index.

Call any of these with `run <id> KEY=value …` (the **Name** column is the KEY); each entry includes a worked **Example usage**. A ★ marks an algorithm with a friendly niva alias verb.

`3d:tessellate` — Tessellate

group: Vector geometry

This algorithm tessellates a polygon geometry layer, dividing the geometries into triangular components.

The output layer consists of multipolygon geometries for each input feature, with each multipolygon consisting of multiple triangle component polygons.

Parameter	Type	Required	Default	Description
`INPUT`	source	yes	—	Input layer
`OUTPUT`	sink	yes	—	Tessellated

Outputs: `OUTPUT` (outputVector)

Example usage

```
run 3d:tessellate INPUT=input.gpkg OUTPUT=output.gpkg
```

This calls **Tessellate**: `INPUT=input.gpkg` — Input layer; `OUTPUT=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass: algorithms

grass — GRASS GIS algorithms — 307 algorithms (QGIS 4.0.3). Auto-generated by `scripts/gen_algorithms.py`; see the index.

Call any of these with `run <id> KEY=value …` (the **Name** column is the KEY); each entry includes a worked **Example usage**. A ★ marks an algorithm with a friendly niva alias verb.

`grass:g.extension.list` — g.extension.list

group: General (g.*)

g.extension.list - List GRASS addons.

Parameter	Type	Required	Default	Description
`list`	enum	no	0	List — options: 0=Locally installed extensions, 1=Extensions available in the official GRASS GIS Addons repository, 2=Extensions available in the official GRASS GIS Addons repository including module description
`html`	fileDestination	no	addons_list.html	List of addons
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Outputs: `html` (outputHtml)

Example usage

```
run grass:g.extension.list list=0 html=html.txt
```

This calls **g.extension.list**: `list=0` selects *Locally installed extensions*; `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:g.extension.manage` — g.extension.manage

group: General (g.*)

g.extension.manage - Install or uninstall GRASS addons.

Parameter	Type	Required	Default	Description
`extension`	string	yes	—	Name of Extension
`operation`	enum	yes	—	Operation — options: 0=add, 1=remove
`-f`	boolean	no	False	Force (required for removal)
`-t`	boolean	no	False	Operate on toolboxes instead of single modules (experimental)
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Example usage

```
run grass:g.extension.manage extension=value operation=0
```

This calls **g.extension.manage**: `extension=value` — Name of Extension; `operation=0` selects *add*. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:g.version` — g.version

group: General (g.*)

g.version - Display GRASS GIS version info. Prints only version if run with no options checked.

Parameter	Type	Required	Default	Description
`-c`	boolean	no	False	Print copyright message
`-x`	boolean	no	False	Print citation options
`-b`	boolean	no	False	Print build information
`-r`	boolean	no	True	Print GIS library revision number and date
`-e`	boolean	no	True	Print info for additional libraries
`-g`	boolean	no	False	Print in shell script style (with Git commit reference)
`--verbose`	boolean	no	False	Print verbose output
`html`	fileDestination	no	grass_version_-info.html	Output file
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Outputs: `html` (outputHtml)

Example usage

```
run grass:g.version html=html.txt
```

This calls **g.version**: `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.albedo` — i.albedo

group: Imagery (i.*)

Computes broad band albedo from surface reflectance.

Parameter	Type	Required	Default	Description
`input`	multilayer	yes	—	Name of input raster maps
`-m`	boolean	no	False	MODIS (7 input bands:1,2,3,4,5,6,7)
`-n`	boolean	no	False	NOAA AVHRR (2 input bands:1,2)
`-l`	boolean	no	False	Landsat 5+7 (6 input bands:1,2,3,4,5,7)
`-8`	boolean	no	False	Landsat 8 (7 input bands:1,2,3,4,5,6,7)
`-a`	boolean	no	False	ASTER (6 input bands:1,3,5,6,8,9)
`-c`	boolean	no	False	Aggressive mode (Landsat)
`-d`	boolean	no	False	Soft mode (MODIS)
`output`	rasterDestination	yes	—	Albedo
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)

Parameter	Type	Required	Default	Description
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.albedo input="a.gpkg;b.gpkg" output=output.tif
```

This calls **i.albedo**: input="a.gpkg;b.gpkg" — Name of input raster maps; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.aster.toar — i.aster.toar

group: Imagery (i.*)

Calculates Top of Atmosphere Radiance/Reflectance/Brightness Temperature from ASTER DN.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Names of ASTER DN layers (15 layers)
dayofyear	number	no	0.0	Day of Year of satellite overpass [0-366]
sun_elevation	number	yes	—	Sun elevation angle (degrees, < 90.0)
-r	boolean	no	False	Output is radiance (W/m2)
-a	boolean	no	False	VNIR is High Gain
-b	boolean	no	False	SWIR is High Gain
-c	boolean	no	False	VNIR is Low Gain 1
-d	boolean	no	False	SWIR is Low Gain 1
-e	boolean	no	False	SWIR is Low Gain 2
output	folderDestination	yes	—	Output Directory
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFolder)

Example usage

```
run grass:i.aster.toar input="a.gpkg;b.gpkg" sun_elevation=10 dayofyear=0 output=output/
```

This calls **i.aster.toar**: input="a.gpkg;b.gpkg" — Names of ASTER DN layers (15 layers); sun_elevation=10 — Sun elevation angle (degrees, < 90.0); dayofyear=0 — Day of Year of satellite overpass [0-366]; output=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.atcorr — i.atcorr

group: Imagery (i.*)

Performs atmospheric correction using the 6S algorithm.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
range	range	no	0,255	Input imagery range [0,255]
elevation	raster	no	—	Input altitude raster map in m (optional)
visibility	raster	no	—	Input visibility raster map in km (optional)
parameters	file	yes	—	Name of input text file
rescale	range	no	0,255	Rescale output raster map [0,255]

Parameter	Type	Required	Default	Description
output	rasterDestination	yes	—	Atmospheric correction
-i *(adv)*	boolean	no	False	Output raster map as integer
-r *(adv)*	boolean	no	False	Input raster map converted to reflectance (default is radiance)
-a *(adv)*	boolean	no	False	Input from ETM+ image taken after July 1, 2000
-b *(adv)*	boolean	no	False	Input from ETM+ image taken before July 1, 2000
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.atcorr input=dem.tif parameters=input.dat range=0,255 elevation=dem.tif
 ↪  visibility=dem.tif rescale=0,255 output=output.tif
```

This calls **i.atcorr**: input=dem.tif — Name of input raster map; parameters=input.dat — Name of input text file; range=0,255 — Input imagery range [0,255]; elevation=dem.tif — Input altitude raster map in m (optional); visibility=dem.tif — Input visibility raster map in km (optional); rescale=0,255 — Rescale output raster map [0,255]; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.biomass — i.biomass

group: Imagery (i.*)

Computes biomass growth, precursor of crop yield calculation.

Parameter	Type	Required	Default	Description
fpar	raster	yes	—	Name of fPAR raster map
lightuse_efficiency	raster	yes	—	Name of light use efficiency raster map (UZB:cotton=1.9)
latitude	raster	yes	—	Name of degree latitude raster map [dd.ddd]
dayofyear	raster	yes	—	Name of Day of Year raster map [1-366]
transmissivity_singleway	raster	yes	—	Name of single-way transmissivity raster map [0.0-1.0]
water_availability	raster	yes	—	Value of water availability raster map [0.0-1.0]
output	rasterDestination	yes	—	Biomass
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.biomass fpar=dem.tif lightuse_efficiency=dem.tif latitude=dem.tif
 ↪  dayofyear=dem.tif transmissivity_singleway=dem.tif water_availability=dem.tif
 ↪  output=output.tif
```

This calls **i.biomass**: `fpar=dem.tif` — Name of fPAR raster map; `lightuse_efficiency=dem.tif` — Name of light use efficiency raster map (UZB:cotton=1.9); `latitude=dem.tif` — Name of degree latitude raster map [dd.ddd]; `dayofyear=dem.tif` — Name of Day of Year raster map [1-366]; `transmissivity_singleway=dem.tif` — Name of single-way transmissivity raster map [0.0-1.0]; `water_availability=dem.tif` — Value of water availability raster map [0.0-1.0]; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.cca` — i.cca

group: Imagery (i.*)

Canonical components analysis (CCA) program for image processing.

Parameter	Type	Required	Default	Description
`input`	multilayer	yes	—	Input rasters (2 to 8)
`signature`	file	yes	—	File containing spectral signatures
`output`	folderDestination	yes	—	Output Directory
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output` (outputFolder)

Example usage

```
run grass:i.cca input="a.gpkg;b.gpkg" signature=input.dat output=output/
```

This calls **i.cca**: `input="a.gpkg;b.gpkg"` — Input rasters (2 to 8); `signature=input.dat` — File containing spectral signatures; `output=output/` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.cluster` — i.cluster

group: Imagery (i.*)

Generates spectral signatures for land cover types in an image using a clustering algorithm.

Parameter	Type	Required	Default	Description
`input`	multilayer	yes	—	Input rasters
`classes`	number	yes	—	Initial number of classes (1-255)
`seed`	file	no	—	Name of file containing initial signatures
`sample`	string	no	—	Sampling intervals (by row and col)
`iterations`	number	no	30.0	Maximum number of iterations
`convergence`	number	no	98.0	Percent convergence
`separation`	number	no	0.0	Cluster separation
`min_size`	number	no	17.0	Minimum number of pixels in a class
`signaturefile`	fileDestination	yes	—	Signature File
`reportfile`	fileDestination	no	—	Final Report File
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `signaturefile` (outputFile), `reportfile` (outputFile)

Example usage

```
run grass:i.cluster input="a.gpkg;b.gpkg" classes=10 seed=input.dat sample=value
 ↪  iterations=30 convergence=98 separation=0 signaturefile=signaturefile.txt
```

This calls **i.cluster**: `input="a.gpkg;b.gpkg"` — Input rasters; `classes=10` — Initial number of classes (1-255); `seed=input.dat` — Name of file containing initial signatures; `sample=value` — Sampling intervals (by row and col); `iterations=30` — Maximum number of iterations; `convergence=98` — Percent convergence; `separation=0` — Cluster separation; `signaturefile=signatu refile.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.colors.enhance` — i.colors.enhance

group: Imagery (i.*)

Performs auto-balancing of colors for RGB images.

Parameter	Type	Required	Default	Description
`red`	raster	yes	—	Name of red channel
`green`	raster	yes	—	Name of green channel
`blue`	raster	yes	—	Name of blue channel
`strength`	number	no	98.0	Cropping intensity (upper brightness level)
`-f *(adv)*`	boolean	no	False	Extend colors to full range of data on each channel
`-p *(adv)*`	boolean	no	False	Preserve relative colors, adjust brightness only
`-r *(adv)*`	boolean	no	False	Reset to standard color range
`-s *(adv)*`	boolean	no	False	Process bands serially (default: run in parallel)
`redoutput`	rasterDestination	yes	—	Enhanced Red
`greenoutput`	rasterDestination	yes	—	Enhanced Green
`blueoutput`	rasterDestination	yes	—	Enhanced Blue
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `redoutput` (outputRaster), `greenoutput` (outputRaster), `blueoutput` (outputRaster)

Example usage

```
run grass:i.colors.enhance red=dem.tif green=dem.tif blue=dem.tif strength=98
 ↪  redoutput=redoutput.tif
```

This calls **i.colors.enhance**: `red=dem.tif` — Name of red channel; `green=dem.tif` — Name of green channel; `blue=dem.tif` — Name of blue channel; `strength=98` — Cropping intensity (upper brightness level); `redoutput=redoutput.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.eb.eta` — i.eb.eta

group: Imagery (i.*)

Actual evapotranspiration for diurnal period (Bastiaanssen, 1995).

Parameter	Type	Required	Default	Description
`netradiationdiurnal`	raster	yes	—	Name of the diurnal net radiation map [W/m2]
`evaporativefraction`	raster	yes	—	Name of the evaporative fraction map
`temperature`	raster	yes	—	Name of the surface skin temperature [K]
`output`	rasterDestination	yes	—	Evapotranspiration

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.eb.eta netradiationdiurnal=dem.tif evaporativefraction=dem.tif
↪  temperature=dem.tif output=output.tif
```

This calls **i.eb.eta**: netradiationdiurnal=dem.tif — Name of the diurnal net radiation map [W/m2]; evaporativefraction=dem.tif — Name of the evaporative fraction map; temperature=dem.tif — Name of the surface skin temperature [K]; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.eb.evapfr — i.eb.evapfr

group: Imagery (i.*)

Computes evaporative fraction (Bastiaanssen, 1995) and root zone soil moisture (Makin, Molden and Bastiaanssen, 2001).

Parameter	Type	Required	Default	Description
netradiation	raster	yes	—	Name of Net Radiation raster map [W/m2]
soilheatflux	raster	yes	—	Name of soil heat flux raster map [W/m2]
sensibleheatflux	raster	yes	—	Name of sensible heat flux raster map [W/m2]
evaporativefraction	rasterDestination	yes	—	Evaporative Fraction
soilmoisture	rasterDestination	no	—	Root Zone Soil Moisture
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: evaporativefraction (outputRaster), soilmoisture (outputRaster)

Example usage

```
run grass:i.eb.evapfr netradiation=dem.tif soilheatflux=dem.tif sensibleheatflux=dem.tif
↪  evaporativefraction=evaporativefraction.tif
```

This calls **i.eb.evapfr**: netradiation=dem.tif — Name of Net Radiation raster map [W/m2]; soilheatflux=dem.tif — Name of soil heat flux raster map [W/m2]; sensibleheatflux=dem.tif — Name of sensible heat flux raster map [W/m2]; evaporativefraction=evaporativefraction.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.eb.hsebal01.coords — i.eb.hsebal01.coords

group: Imagery (i.*)

i.eb.hsebal01.coords - Computes sensible heat flux iteration SEBAL 01. Inline coordinates

Parameter	Type	Required	Default	Description
netradiation	raster	yes	—	Name of instantaneous net radiation raster map [W/m2]
soilheatflux	raster	yes	—	Name of instantaneous soil heat flux raster map [W/m2]
aerodynresistance	raster	yes	—	Name of aerodynamic resistance to heat momentum raster map [s/m]
temperaturemeansealevel	raster	yes	—	Name of altitude corrected surface temperature raster map [K]
frictionvelocitystar	number	no	0.32407	Value of the height independent friction velocity (u*) [m/s]
vapourpressureactual	raster	yes	—	Name of the actual vapour pressure (e_act) map [KPa]
row_wet_pixel	number	no	—	Row value of the wet pixel
column_wet_pixel	number	no	—	Column value of the wet pixel
row_dry_pixel	number	no	—	Row value of the dry pixel
column_dry_pixel	number	no	—	Column value of the dry pixel
-a *(adv)*	boolean	no	False	Automatic wet/dry pixel (careful!)
-c *(adv)*	boolean	no	False	Dry/Wet pixels coordinates are in image projection, not row/col
output	rasterDestination	yes	—	Sensible Heat Flux
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.eb.hsebal01.coords netradiation=dem.tif soilheatflux=dem.tif
 ↪  aerodynresistance=dem.tif temperaturemeansealevel=dem.tif vapourpressureactual=dem.tif
 ↪  frictionvelocitystar=0.32407 row_wet_pixel=10 output=output.tif
```

This calls **i.eb.hsebal01.coords**: netradiation=dem.tif — Name of instantaneous net radiation raster map [W/m2]; soilheatflux=dem.tif — Name of instantaneous soil heat flux raster map [W/m2]; aerodynresistance=dem.tif — Name of aerodynamic resistance to heat momentum raster map [s/m]; temperaturemeansealevel=dem.tif — Name of altitude corrected surface temperature raster map [K]; vapourpressureactual=dem.tif — Name of the actual vapour pressure (e_act) map [KPa]; frictionvelocitystar=0.32407 — Value of the height independent friction velocity (u*) [m/s]; row_wet_pixel=10 — Row value of the wet pixel; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.eb.netrad — i.eb.netrad

group: Imagery (i.*)

Net radiation approximation (Bastiaanssen, 1995).

Parameter	Type	Required	Default	Description
albedo	raster	yes	—	Name of albedo raster map [0.0;1.0]
ndvi	raster	yes	—	Name of NDVI raster map [-1.0;+1.0]
temperature	raster	yes	—	Name of surface temperature raster map [K]
localutctime	raster	yes	—	Name of time of satellite overpass raster map [local time in UTC]
temperaturedifference2m	raster	yes	—	Name of the difference map of temperature from surface skin to about 2 m height [K]

Parameter	Type	Required	Default	Description
emissivity	raster	yes	—	Name of the emissivity map [-]
transmissivity_singleway	raster	yes	—	Name of the single-way atmospheric transmissivitymap [-]
dayofyear	raster	yes	—	Name of the Day Of Year (DOY) map [-]
sunzenithangle	raster	yes	—	Name of the sun zenith angle map [degrees]
output	rasterDestination	yes	—	Net Radiation
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.eb.netrad albedo=dem.tif ndvi=dem.tif temperature=dem.tif localutctime=dem.tif
 ↪  temperaturedifference2m=dem.tif emissivity=dem.tif transmissivity_singleway=dem.tif
 ↪  output=output.tif
```

This calls **i.eb.netrad**: albedo=dem.tif — Name of albedo raster map [0.0;1.0]; ndvi=dem.tif — Name of NDVI raster map [-1.0;+1.0]; temperature=dem.tif — Name of surface temperature raster map [K]; localutctime=dem.tif — Name of time of satellite overpass raster map [local time in UTC]; temperaturedifference2m=dem.tif — Name of the difference map of temperature from surface skin to about 2 m height...; emissivity=dem.tif — Name of the emissivity map [-]; transmissivity_singleway=dem.tif — Name of the single-way atmospheric transmissivitymap [-]; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.eb.soilheatflux — i.eb.soilheatflux

group: Imagery (i.*)

Soil heat flux approximation (Bastiaanssen, 1995).

Parameter	Type	Required	Default	Description
albedo	raster	yes	—	Name of albedo raster map [0.0;1.0]
ndvi	raster	yes	—	Name of NDVI raster map [-1.0;+1.0]
temperature	raster	yes	—	Name of Surface temperature raster map [K]
netradiation	raster	yes	—	Name of Net Radiation raster map [W/m2]
localutctime	raster	yes	—	Name of time of satellite overpass raster map [local time in UTC]
-r	boolean	no	False	HAPEX-Sahel empirical correction (Roerink, 1995)
output	rasterDestination	yes	—	Soil Heat Flux
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.eb.soilheatflux albedo=dem.tif ndvi=dem.tif temperature=dem.tif
 ↪  netradiation=dem.tif localutctime=dem.tif output=output.tif
```

This calls **i.eb.soilheatflux**: `albedo=dem.tif` — Name of albedo raster map [0.0;1.0]; `ndvi=dem.tif` — Name of NDVI raster map [-1.0;+1.0]; `temperature=dem.tif` — Name of Surface temperature raster map [K]; `netradiation=dem.tif` — Name of Net Radiation raster map [W/m2]; `localutctime=dem.tif` — Name of time of satellite overpass raster map [local time in UTC]; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.emissivity` — i.emissivity

group: Imagery (i.*)

Computes emissivity from NDVI, generic method for sparse land.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Name of NDVI raster map [-]
`output`	rasterDestination	yes	—	Emissivity
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:i.emissivity input=dem.tif output=output.tif
```

This calls **i.emissivity**: `input=dem.tif` — Name of NDVI raster map [-]; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.evapo.mh` — i.evapo.mh

group: Imagery (i.*)

Computes evapotranspiration calculation modified or original Hargreaves formulation, 2001.

Parameter	Type	Required	Default	Description
`netradiation_diurnal`	raster	yes	—	Name of input diurnal net radiation raster map [W/m2/d]
`average_temperature`	raster	yes	—	Name of input average air temperature raster map [C]
`minimum_temperature`	raster	yes	—	Name of input minimum air temperature raster map [C]
`maximum_temperature`	raster	yes	—	Name of input maximum air temperature raster map [C]
`precipitation`	raster	no	—	Name of precipitation raster map [mm/month]
`-z *(adv)*`	boolean	no	False	Set negative ETa to zero
`-h *(adv)*`	boolean	no	False	Use original Hargreaves (1985)
`-s *(adv)*`	boolean	no	False	Use Hargreaves-Samani (1985)
`output`	rasterDestination	yes	—	Evapotranspiration
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:i.evapo.mh netradiation_diurnal=dem.tif average_temperature=dem.tif
 ↪  minimum_temperature=dem.tif maximum_temperature=dem.tif precipitation=dem.tif
 ↪  output=output.tif
```

This calls **i.evapo.mh**: `netradiation_diurnal=dem.tif` — Name of input diurnal net radiation raster map [W/m2/d]; `average_temperature=dem.tif` — Name of input average air temperature raster map [C]; `minimum_temperature=dem.tif` — Name of input minimum air temperature raster map [C]; `maximum_temperature=dem.tif` — Name of input maximum air temperature raster map [C]; `precipitation=dem.tif` — Name of precipitation raster map [mm/month]; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.evapo.pm` — i.evapo.pm

group: Imagery (i.*)

Computes potential evapotranspiration calculation with hourly Penman-Monteith.

Parameter	Type	Required	Default	Description
`elevation`	raster	yes	—	Name of input elevation raster map [m a.s.l.]
`temperature`	raster	yes	—	Name of input temperature raster map [C]
`relativehumidity`	raster	yes	—	Name of input relative humidity raster map [%]
`windspeed`	raster	yes	—	Name of input wind speed raster map [m/s]
`netradiation`	raster	yes	—	Name of input net solar radiation raster map [MJ/m2/h]
`cropheight`	raster	yes	—	Name of input crop height raster map [m]
`-z *(adv)*`	boolean	no	False	Set negative ETa to zero
`-n *(adv)*`	boolean	no	False	Use Night-time
`output`	rasterDestination	yes	—	Evapotranspiration
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:i.evapo.pm elevation=dem.tif temperature=dem.tif relativehumidity=dem.tif
 ↪  windspeed=dem.tif netradiation=dem.tif cropheight=dem.tif output=output.tif
```

This calls **i.evapo.pm**: `elevation=dem.tif` — Name of input elevation raster map [m a.s.l.]; `temperature=dem.tif` — Name of input temperature raster map [C]; `relativehumidity=dem.tif` — Name of input relative humidity raster map [%]; `windspeed=dem.tif` — Name of input wind speed raster map [m/s]; `netradiation=dem.tif` — Name of input net solar radiation raster map [MJ/m2/h]; `cropheight=dem.tif` — Name of input crop height raster map [m]; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.evapo.pt — i.evapo.pt

group: Imagery (i.*)

Computes evapotranspiration calculation Priestley and Taylor formulation, 1972.

Parameter	Type	Required	Default	Description
net_radiation	raster	yes	—	Name of input net radiation raster map [W/m2]
soil_heatflux	raster	yes	—	Name of input soil heat flux raster map [W/m2]
air_temperature	raster	yes	—	Name of input air temperature raster map [K]
atmospheric_pressure	raster	yes	—	Name of input atmospheric pressure raster map [millibars]
priestley_taylor_coeff	number	no	1.26	Priestley-Taylor coefficient
-z *(adv)*	boolean	no	False	Set negative ETa to zero
output	rasterDestination	yes	—	Evapotranspiration
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.evapo.pt net_radiation=dem.tif soil_heatflux=dem.tif air_temperature=dem.tif
 ↪  atmospheric_pressure=dem.tif priestley_taylor_coeff=1.26 output=output.tif
```

This calls **i.evapo.pt**: net_radiation=dem.tif — Name of input net radiation raster map [W/m2]; soil_heatflux=dem.tif — Name of input soil heat flux raster map [W/m2]; air_temperature=dem.tif — Name of input air temperature raster map [K]; atmospheric_pressure=dem.tif — Name of input atmospheric pressure raster map [millibars]; priestley_taylor_coeff=1.26 — Priestley-Taylor coefficient; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.evapo.time — i.evapo.time

group: Imagery (i.*)

Computes temporal integration of satellite ET actual (ETa) following the daily ET reference (ETo) from meteorological station(s).

Parameter	Type	Required	Default	Description
eta	multilayer	yes	—	Names of satellite ETa raster maps [mm/d or cm/d]
eta_doy	multilayer	yes	—	Names of satellite ETa Day of Year (DOY) raster maps [0-400] [-]
eto	multilayer	yes	—	Names of meteorological station ETo raster maps [0-400] [mm/d or cm/d]
eto_doy_min	number	no	1.0	Value of DOY for ETo first day
start_period	number	no	1.0	Value of DOY for the first day of the period studied
end_period	number	no	1.0	Value of DOY for the last day of the period studied
output	rasterDestination	yes	—	Temporal integration
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.evapo.time eta="a.gpkg;b.gpkg" eta_doy="a.gpkg;b.gpkg" eto="a.gpkg;b.gpkg"
↪  eto_doy_min=1 start_period=1 end_period=1 output=output.tif
```

This calls **i.evapo.time**: eta="a.gpkg;b.gpkg" — Names of satellite ETa raster maps [mm/d or cm/d]; eta_doy="a.gpkg;b.gpkg" — Names of satellite ETa Day of Year (DOY) raster maps [0-400] [-]; eto="a.gpkg;b.gpkg" — Names of meteorological station ETo raster maps [0-400] [mm/d or cm/d]; eto_doy_min=1 — Value of DOY for ETo first day; start_period=1 — Value of DOY for the first day of the period studied; end_period=1 — Value of DOY for the last day of the period studied; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.fft — i.fft

group: Imagery (i.*)

Fast Fourier Transform (FFT) for image processing.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
real	rasterDestination	yes	—	Real part arrays
imaginary	rasterDestination	yes	—	Imaginary part arrays
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: real (outputRaster), imaginary (outputRaster)

Example usage

```
run grass:i.fft input=dem.tif real=real.tif
```

This calls **i.fft**: input=dem.tif — Name of input raster map; real=real.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.gensig — i.gensig

group: Imagery (i.*)

Generates statistics for i.maxlik from raster map.

Parameter	Type	Required	Default	Description
trainingmap	raster	yes	—	Ground truth training map
input	multilayer	yes	—	Input rasters
signaturefile	fileDestination	yes	—	Signature File

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: signaturefile (outputFile)

Example usage

```
run grass:i.gensig trainingmap=dem.tif input="a.gpkg;b.gpkg"
 ↪  signaturefile=signaturefile.txt
```

This calls **i.gensig**: trainingmap=dem.tif — Ground truth training map; input="a.gpkg;b.gpkg" — Input rasters; signaturefile=signaturefile.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.gensigset — i.gensigset

group: Imagery (i.*)

Generates statistics for i.smap from raster map.

Parameter	Type	Required	Default	Description
trainingmap	raster	yes	—	Ground truth training map
input	multilayer	yes	—	Input rasters
maxsig	number	no	5.0	Maximum number of sub-signatures in any class
signaturefile	fileDestination	yes	—	Signature File
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: signaturefile (outputFile)

Example usage

```
run grass:i.gensigset trainingmap=dem.tif input="a.gpkg;b.gpkg" maxsig=5
 ↪  signaturefile=signaturefile.txt
```

This calls **i.gensigset**: trainingmap=dem.tif — Ground truth training map; input="a.gpkg;b.gpkg" — Input rasters; maxsig=5 — Maximum number of sub-signatures in any class; signaturefile=signaturefile.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.group — i.group

group: Imagery (i.*)

Regroup multiple mono-band rasters into a single multiband raster.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Input rasters
group	rasterDestination	yes	—	Multiband raster
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: group (outputRaster)

Example usage

```
run grass:i.group input="a.gpkg;b.gpkg" group=group.tif
```

This calls **i.group**: input="a.gpkg;b.gpkg" — Input rasters; group=group.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.his.rgb — i.his.rgb

group: Imagery (i.*)

Transforms raster maps from HIS (Hue-Intensity-Saturation) color space to RGB (Red-Green-Blue) color space.

Parameter	Type	Required	Default	Description
hue	raster	yes	—	Name of input raster map (hue)
intensity	raster	yes	—	Name of input raster map (intensity)
saturation	raster	yes	—	Name of input raster map (saturation)
red	rasterDestination	yes	—	Red
green	rasterDestination	yes	—	Green
blue	rasterDestination	yes	—	Blue
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: red (outputRaster), green (outputRaster), blue (outputRaster)

Example usage

```
run grass:i.his.rgb hue=dem.tif intensity=dem.tif saturation=dem.tif red=red.tif
```

This calls **i.his.rgb**: hue=dem.tif — Name of input raster map (hue); intensity=dem.tif — Name of input raster map (intensity); saturation=dem.tif — Name of input raster map (saturation); red=red.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.ifft — i.ifft

group: Imagery (i.*)

Inverse Fast Fourier Transform (IFFT) for image processing.

Parameter	Type	Required	Default	Description
real	raster	yes	—	Name of input raster map (image fft, real part)
imaginary	raster	yes	—	Name of input raster map (image fft, imaginary part)
output	rasterDestination	yes	—	Inverse Fast Fourier Transform
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)

Parameter	Type	Required	Default	Description
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.ifft real=dem.tif imaginary=dem.tif output=output.tif
```

This calls **i.ifft**: real=dem.tif — Name of input raster map (image fft, real part); imaginary=dem.tif — Name of input raster map (image fft, imaginary part); output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.image.mosaic — i.image.mosaic

group: Imagery (i.*)

Mosaics several images and extends colormap.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Input rasters
output	rasterDestination	yes	—	Mosaic Raster
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.image.mosaic input="a.gpkg;b.gpkg" output=output.tif
```

This calls **i.image.mosaic**: input="a.gpkg;b.gpkg" — Input rasters; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.in.spotvgt — i.in.spotvgt

group: Imagery (i.*)

Imports SPOT VGT NDVI data into a raster map.

Parameter	Type	Required	Default	Description
input	file	yes	—	Name of input SPOT VGT NDVI HDF file
-a *(adv)*	boolean	no	False	Also import quality map (SM status map layer) and filter NDVI map
output	rasterDestination	yes	—	SPOT NDVI Raster
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:i.in.spotvgt input=input.dat output=output.tif
```

This calls **i.in.spotvgt**: `input=input.dat` — Name of input SPOT VGT NDVI HDF file; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.landsat.acca` — i.landsat.acca

group: Imagery (i.*)

Performs Landsat TM/ETM+ Automatic Cloud Cover Assessment (ACCA).

Parameter	Type	Required	Default	Description
`rasters`	multilayer	yes	—	Landsat input rasters
`b56composite`	number	no	225.0	B56composite (step 6)
`b45ratio`	number	no	1.0	B45ratio: Desert detection (step 10)
`histogram`	number	no	100.0	Number of classes in the cloud temperature histogram
`-5 *(adv)*`	boolean	no	False	Data is Landsat-5 TM
`-f *(adv)*`	boolean	no	False	Apply post-processing filter to remove small holes
`-x *(adv)*`	boolean	no	False	Always use cloud signature (step 14)
`-2 *(adv)*`	boolean	no	False	Bypass second-pass processing, and merge warm (not ambiguous) and cold clouds
`-s *(adv)*`	boolean	no	False	Include a category for cloud shadows
`output`	rasterDestination	yes	—	ACCA Raster
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:i.landsat.acca rasters="a.gpkg;b.gpkg" b56composite=225 b45ratio=1 histogram=100
 ↪  output=output.tif
```

This calls **i.landsat.acca**: `rasters="a.gpkg;b.gpkg"` — Landsat input rasters; `b56composite=225` — B56composite (step 6); `b45ratio=1` — B45ratio: Desert detection (step 10); `histogram=100` — Number of classes in the cloud temperature histogram; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.landsat.toar` — i.landsat.toar

group: Imagery (i.*)

Calculates top-of-atmosphere radiance or reflectance and temperature for Landsat MSS/TM/ETM+/OLI

Parameter	Type	Required	Default	Description
rasters	multilayer	yes	—	Landsat input rasters
metfile	file	no	—	Name of Landsat metadata file (.met or MTL.txt)
sensor	enum	no	7	Spacecraft sensor — options: 0=mss1, 1=mss2, 2=mss3, 3=mss4, 4=mss5, 5=tm4, 6=tm5, 7=tm7, 8=oli8
method	enum	no	0	Atmospheric correction method — options: 0=uncorrected, 1=dos1, 2=dos2, 3=dos2b, 4=dos3, 5=dos4
date	string	no	—	Image acquisition date (yyyy-mm-dd)
sun_elevation	number	no	—	Sun elevation in degrees
product_date	string	no	—	Image creation date (yyyy-mm-dd)
gain	string	no	—	Gain (H/L) of all Landsat ETM+ bands (1-5,61,62,7,8)
percent	number	no	0.01	Percent of solar radiance in path radiance
pixel	number	no	1000.0	Minimum pixels to consider digital number as dark object
rayleigh	number	no	0.0	Rayleigh atmosphere (diffuse sky irradiance)
scale	number	no	1.0	Scale factor for output
-r *(adv)*	boolean	no	False	Output at-sensor radiance instead of reflectance for all bands
-n *(adv)*	boolean	no	False	Input raster maps use as extension the number of the band instead the code
output	folderDestination	yes	—	Output Directory
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFolder)

Example usage

```
run grass:i.landsat.toar rasters="a.gpkg;b.gpkg" metfile=input.dat sensor=7 method=0
 ↪  date=value sun_elevation=10 product_date=value output=output/
```

This calls **i.landsat.toar**: rasters="a.gpkg;b.gpkg" — Landsat input rasters; metfile=input.dat — Name of Landsat metadata file (.met or MTL.txt); sensor=7 selects *tm7*; method=0 selects *uncorrected*; date=value — Image acquisition date (yyyy-mm-dd); sun_elevation=10 — Sun elevation in degrees; product_date=value — Image creation date (yyyy-mm-dd); output=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.maxlik — i.maxlik

group: Imagery (i.*)

Classifies the cell spectral reflectances in imagery data.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Input rasters
signaturefile	file	yes	—	Name of input file containing signatures
output	rasterDestination	yes	—	Classification
reject	rasterDestination	no	—	Reject Threshold
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster), `reject` (outputRaster)

Example usage

```
run grass:i.maxlik input="a.gpkg;b.gpkg" signaturefile=input.dat output=output.tif
```

This calls **i.maxlik**: `input="a.gpkg;b.gpkg"` — Input rasters; `signaturefile=input.dat` — Name of input file containing signatures; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.modis.qc` — i.modis.qc

group: Imagery (i.*)

Extracts quality control parameters from MODIS QC layers.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Name of input surface reflectance QC layer [bit array]
`productname`	enum	no	8	Name of MODIS product type — options: 0=mod09Q1, 1=mod09A1, 2=mod09A1s, 3=mod09CMG, 4=mod09CMGs, 5=mod09CMGi, 6=mod11A1, 7=mod11A2, 8=mod13A2, 9=mcd43B2, 10=mcd43B2q, 11=mod13Q1
`qcname`	enum	no	5	Name of QC type to extract — options: 0=adjcorr, 1=atcorr, 2=cloud, 3=data_quality, 4=diff_orbit_from_500m, 5=modland_qa, 6=mandatory_qa_11A1, 7=data_quality_flag_11A1, 8=emis_error_11A1, 9=lst_error_11A1, 10=data_quality_flag_11A2, 11=emis_error_11A2, 12=mandatory_qa_11A2, 13=lst_error_11A2, 14=aerosol_quantity, 15=brdf_correction_performed, 16=cirrus_detected, 17=cloud_shadow, 18=cloud_state, 19=internal_cloud_algorithm, 20=internal_fire_algorithm, 21=internal_snow_mask, 22=land_water, 23=mod35_snow_ice, 24=pixel_adjacent_to_cloud, 25=icm_cloudy, 26=icm_clear, 27=icm_high_clouds, 28=icm_low_clouds, 29=icm_snow, 30=icm_fire, 31=icm_sun_glint, 32=icm_dust, 33=icm_cloud_shadow, 34=icm_pixel_is_adjacent_to_cloud, 35=icm_cirrus, 36=icm_pan_flag, 37=icm_criteria_for_aerosol_retrieval, 38=icm_aot_has_clim_val, 39=modland_qa, 40=vi_usefulness, 41=aerosol_quantity, 42=pixel_adjacent_to_cloud, 43=brdf_correction_performed, 44=mixed_clouds, 45=land_water, 46=possible_snow_ice, 47=possible_shadow, 48=platform, 49=land_water, 50=sun_z_angle_at_local_noon, 51=brdf_correction_performed
`band`	enum	no	[0, 1]	Band number of MODIS product (mod09Q1=[1,2],mod09A1=[1-7],m[o/y]d09CMG=[1-7], mcd43B2q=[1-7]) — options: 0=1, 1=2, 2=3, 3=4, 4=5, 5=6, 6=7
`output`	rasterDestination	yes	—	QC Classification
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Parameter	Type	Required	Default	Description
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.modis.qc input=dem.tif productname=8 qcname=5 band=0 output=output.tif
```

This calls **i.modis.qc**: input=dem.tif — Name of input surface reflectance QC layer [bit array]; productname=8 selects *mod13A2*; qcname=5 selects *modland_qa*; band=0 selects *1*; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.oif — i.oif

group: Imagery (i.*)

Calculates Optimum-Index-Factor table for spectral bands

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Name of input raster map(s)
-g *(adv)*	boolean	no	False	Print in shell script style
-s *(adv)*	boolean	no	False	Process bands serially (default: run in parallel)
output	fileDestination	yes	—	OIF File
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:i.oif input="a.gpkg;b.gpkg" output=output.txt
```

This calls **i.oif**: input="a.gpkg;b.gpkg" — Name of input raster map(s); output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.pansharpen — i.pansharpen

group: Imagery (i.*)

Image fusion algorithms to sharpen multispectral with high-res panchromatic channels

Parameter	Type	Required	Default	Description
red	raster	yes	—	Name of red channel
green	raster	yes	—	Name of green channel
blue	raster	yes	—	Name of blue channel
pan	raster	yes	—	Name of raster map to be used for high resolution panchromatic channel
method	enum	no	1	Method — options: 0=brovey, 1=ihs, 2=pca
-l *(adv)*	boolean	no	False	Rebalance blue channel for LANDSAT
-s *(adv)*	boolean	no	False	Process bands serially (default: run in parallel)
redoutput	rasterDestination	yes	—	Enhanced Red
greenoutput	rasterDestination	yes	—	Enhanced Green

Parameter	Type	Required	Default	Description
blueoutput	rasterDestination	yes	—	Enhanced Blue
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: redoutput (outputRaster), greenoutput (outputRaster), blueoutput (outputRaster)

Example usage

```
run grass:i.pansharpen red=dem.tif green=dem.tif blue=dem.tif pan=dem.tif method=1
↪  redoutput=redoutput.tif
```

This calls **i.pansharpen**: red=dem.tif — Name of red channel; green=dem.tif — Name of green channel; blue=dem.tif — Name of blue channel; pan=dem.tif — Name of raster map to be used for high resolution panchromatic channel; method=1 selects *ihs*; redoutput=redoutput.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.pca — i.pca

group: Imagery (i.*)

Principal components analysis (PCA) for image processing.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Name of two or more input raster maps
rescale	range	no	0,255	Rescaling range for output maps. For no rescaling use 0,0
percent	number	no	99.0	Cumulative percent importance for filtering
-n *(adv)*	boolean	no	False	Normalize (center and scale) input maps
-f *(adv)*	boolean	no	False	Output will be filtered input bands
output	folderDestination	yes	—	Output Directory
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFolder)

Example usage

```
run grass:i.pca input="a.gpkg;b.gpkg" rescale=0,255 percent=99 output=output/
```

This calls **i.pca**: input="a.gpkg;b.gpkg" — Name of two or more input raster maps; rescale=0,255 — Rescaling range for output maps. For no rescaling use 0,0; percent=99 — Cumulative percent importance for filtering; output=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.rgb.his — i.rgb.his

group: Imagery (i.*)

Transforms raster maps from RGB (Red-Green-Blue) color space to HIS (Hue-Intensity-Saturation) color space.

Parameter	Type	Required	Default	Description
red	raster	no	—	Name for input raster map (red)
green	raster	no	—	Name for input raster map (green)
blue	raster	no	—	Name for input raster map (blue)
hue	rasterDestination	no	False	Hue
intensity	rasterDestination	no	False	Intensity
saturation	rasterDestination	no	False	Saturation
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: hue (outputRaster), intensity (outputRaster), saturation (outputRaster)

Example usage

```
run grass:i.rgb.his red=dem.tif green=dem.tif blue=dem.tif hue=hue.tif
```

This calls **i.rgb.his**: red=dem.tif — Name for input raster map (red); green=dem.tif — Name for input raster map (green); blue=dem.tif — Name for input raster map (blue); hue=hue.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.segment — i.segment

group: Imagery (i.*)

Identifies segments (objects) from imagery data.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Input rasters
threshold	number	no	0.5	Difference threshold between 0 and 1
method	enum	no	0	Segmentation method — options: 0=region_growing
similarity	enum	no	0	Similarity calculation method — options: 0=euclidean, 1=manhattan
minsize	number	no	1.0	Minimum number of cells in a segment
memory	number	no	300.0	Amount of memory to use in MB
iterations	number	no	20.0	Maximum number of iterations
seeds	raster	no	—	Name for input raster map with starting seeds
bounds	raster	no	—	Name of input bounding/constraining raster map
-d *(adv)*	boolean	no	False	Use 8 neighbors (3x3 neighborhood) instead of the default 4 neighbors for each pixel
-w *(adv)*	boolean	no	False	Weighted input, do not perform the default scaling of input raster maps
output	rasterDestination	yes	—	Segmented Raster
goodness	rasterDestination	no	—	Goodness Raster
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster), goodness (outputRaster)

Example usage

```
run grass:i.segment input="a.gpkg;b.gpkg" threshold=0.5 method=0 similarity=0 minsize=1
↪  memory=300 iterations=20 output=output.tif
```

This calls **i.segment**: `input="a.gpkg;b.gpkg"` — Input rasters; `threshold=0.5` — Difference threshold between 0 and 1; `method=0` selects *region_growing*; `similarity=0` selects *euclidean*; `minsize=1` — Minimum number of cells in a segment; `memory=300` — Amount of memory to use in MB; `iterations=20` — Maximum number of iterations; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.smap` — i.smap

group: Imagery (i.*)

Performs contextual image classification using sequential maximum a posteriori (SMAP) estimation.

Parameter	Type	Required	Default	Description
`input`	multilayer	yes	—	Input rasters
`signaturefile`	file	yes	—	Name of input file containing signatures
`blocksize`	number	no	1024.0	Size of submatrix to process at one time
`-m *(adv)*`	boolean	no	False	Use maximum likelihood estimation (instead of smap)
`output`	rasterDestination	yes	—	Classification
`goodness`	rasterDestination	no	—	Goodness_of_fit
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster), `goodness` (outputRaster)

Example usage

```
run grass:i.smap input="a.gpkg;b.gpkg" signaturefile=input.dat blocksize=1024
↪  output=output.tif
```

This calls **i.smap**: `input="a.gpkg;b.gpkg"` — Input rasters; `signaturefile=input.dat` — Name of input file containing signatures; `blocksize=1024` — Size of submatrix to process at one time; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.tasscap` — i.tasscap

group: Imagery (i.*)

Performs Tasseled Cap (Kauth Thomas) transformation.

Parameter	Type	Required	Default	Description
`input`	multilayer	yes	—	Input rasters. Landsat4-7: bands 1,2,3,4,5,7; Landsat8: bands 2,3,4,5,6,7; MODIS: bands 1,2,3,4,5,6,7
`sensor`	enum	no	0	Satellite sensor — options: 0=landsat4_tm, 1=landsat5_tm, 2=landsat7_etm, 3=landsat8_oli, 4=modis
`output`	folderDestination	yes	—	Output Directory
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFolder)

Example usage

```
run grass:i.tasscap input="a.gpkg;b.gpkg" sensor=0 output=output/
```

This calls **i.tasscap**: input="a.gpkg;b.gpkg" — Input rasters. Landsat4-7: bands 1,2,3,4,5,7; Landsat8: bands 2,3,4,5,6,7;...; sensor=0 selects *landsat4_tm*; output=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.topo.coor.ill — i.topo.coor.ill

group: Imagery (i.*)

i.topo.coor.ill - Creates illumination model for topographic correction of reflectance.

Parameter	Type	Required	Default	Description
basemap	raster	yes	—	Name of elevation raster map
zenith	number	no	0.0	Solar zenith in degrees
azimuth	number	no	0.0	Solar azimuth in degrees
output	rasterDestination	yes	—	Illumination Model
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.topo.coor.ill basemap=dem.tif zenith=0 azimuth=0 output=output.tif
```

This calls **i.topo.coor.ill**: basemap=dem.tif — Name of elevation raster map; zenith=0 — Solar zenith in degrees; azimuth=0 — Solar azimuth in degrees; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.topo.corr — i.topo.corr

group: Imagery (i.*)

Computes topographic correction of reflectance.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Name of reflectance raster maps to be corrected topographically
basemap	raster	yes	—	Name of illumination input base raster map
zenith	number	no	0.0	Solar zenith in degrees
method	enum	no	0	Topographic correction method — options: 0=cosine, 1=minnaert, 2=c-factor, 3=percent
-s *(adv)*	boolean	no	False	Scale output to input and copy color rules
output	folderDestination	yes	—	Output Directory

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFolder)

Example usage

```
run grass:i.topo.corr input="a.gpkg;b.gpkg" basemap=dem.tif zenith=0 method=0
 ↪  output=output/
```

This calls **i.topo.corr**: input="a.gpkg;b.gpkg" — Name of reflectance raster maps to be corrected topographically; basemap=dem.tif — Name of illumination input base raster map; zenith=0 — Solar zenith in degrees; method=0 selects *cosine*; output=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:i.vi — i.vi

group: Imagery (i.*)

Calculates different types of vegetation indices.

Parameter	Type	Required	Default	Description
red	raster	no	—	Name of input red channel surface reflectance map [0.0-1.0]
viname	enum	no	10	Type of vegetation index — options: 0=arvi, 1=dvi, 2=evi, 3=evi2, 4=gvi, 5=gari, 6=gemi, 7=ipvi, 8=msavi, 9=msavi2, 10=ndvi, 11=pvi, 12=savi, 13=sr, 14=vari, 15=wdvi
nir	raster	no	—	Name of input nir channel surface reflectance map [0.0-1.0]
green	raster	no	—	Name of input green channel surface reflectance map [0.0-1.0]
blue	raster	no	—	Name of input blue channel surface reflectance map [0.0-1.0]
band5	raster	no	—	Name of input 5th channel surface reflectance map [0.0-1.0]
band7	raster	no	—	Name of input 7th channel surface reflectance map [0.0-1.0]
soil_line_slope	number	no	—	Value of the slope of the soil line (MSAVI2 only)
soil_line_intercept	number	no	—	Value of the factor of reduction of soil noise (MSAVI2 only)
soil_noise_reduction	number	no	—	Value of the slope of the soil line (MSAVI2 only)
storage_bit	enum	no	1	Maximum bits for digital numbers — options: 0=7, 1=8, 2=9, 3=10, 4=16
output	rasterDestination	yes	—	Vegetation Index
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:i.vi red=dem.tif viname=10 nir=dem.tif green=dem.tif blue=dem.tif band5=dem.tif
↪  band7=dem.tif output=output.tif
```

This calls **i.vi**: `red=dem.tif` — Name of input red channel surface reflectance map [0.0-1.0]; `viname=10` selects *ndvi*; `nir=dem.tif` — Name of input nir channel surface reflectance map [0.0-1.0]; `green=dem.tif` — Name of input green channel surface reflectance map [0.0-1.0]; `blue=dem.tif` — Name of input blue channel surface reflectance map [0.0-1.0]; `band5=dem.tif` — Name of input 5th channel surface reflectance map [0.0-1.0]; `band7=dem.tif` — Name of input 7th channel surface reflectance map [0.0-1.0]; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:i.zc` — i.zc

group: Imagery (i.*)

Zero-crossing ”edge detection” raster function for image processing.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Name of input raster map
`width`	number	no	9.0	x-y extent of the Gaussian filter
`threshold`	number	no	10.0	Sensitivity of Gaussian filter
`orientations`	number	no	1.0	Number of azimuth directions categorized
`output`	rasterDestination	yes	—	Zero crossing
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:i.zc input=dem.tif width=9 threshold=10 orientations=1 output=output.tif
```

This calls **i.zc**: `input=dem.tif` — Name of input raster map; `width=9` — x-y extent of the Gaussian filter; `threshold=10` — Sensitivity of Gaussian filter; `orientations=1` — Number of azimuth directions categorized; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:m.cogo` — m.cogo

group: Miscellaneous (m.*)

A simple utility for converting bearing and distance measurements to coordinates and vice versa. It assumes a Cartesian coordinate system

Parameter	Type	Required	Default	Description
`input`	file	yes	—	Name of input file
`output`	fileDestination	yes	—	Output text file
`coordinates`	point	no	0.0,0.0	Starting coordinate pair
`-l *(adv)*`	boolean	no	False	Lines are labelled
`-q *(adv)*`	boolean	no	False	Suppress warnings
`-r *(adv)*`	boolean	no	False	Convert from coordinates to bearing and distance
`-c *(adv)*`	boolean	no	False	Repeat the starting coordinate at the end to close a loop
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Outputs: `output` (outputFile)

Example usage

```
run grass:m.cogo input=input.dat coordinates=100,100 output=output.txt
```

This calls **m.cogo**: `input=input.dat` — Name of input file; `coordinates=100,100` — Starting coordinate pair; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:nviz` — nviz

group: Visualization(NVIZ)

Visualization and animation tool for GRASS data.

Parameter	Type	Required	Default	Description
`elevation`	multilayer	yes	—	Name of elevation raster map
`color`	multilayer	yes	—	Name of raster map(s) for Color
`vector`	multilayer	yes	—	Name of vector lines/areas overlay map(s)
`point`	multilayer	no	—	Name of vector points overlay file(s)
`volume`	multilayer	no	—	Name of existing 3d raster map
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Example usage

```
run grass:nviz elevation="a.gpkg;b.gpkg" color="a.gpkg;b.gpkg" vector="a.gpkg;b.gpkg"
↪  point="a.gpkg;b.gpkg" volume="a.gpkg;b.gpkg"
```

This calls **nviz**: `elevation="a.gpkg;b.gpkg"` — Name of elevation raster map; `color="a.gpkg;b.gpkg"` — Name of raster map(s) for Color; `vector="a.gpkg;b.gpkg"` — Name of vector lines/areas overlay map(s); `point="a.gpkg;b.gpkg"` — Name of vector points overlay file(s); `volume="a.gpkg;b.gpkg"` — Name of existing 3d raster map. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.basins.fill` — r.basins.fill

group: Raster (r.*)

Generates watershed subbasins raster map.

Parameter	Type	Required	Default	Description
`cnetwork`	raster	yes	—	Input coded stream network raster layer
`tnetwork`	raster	yes	—	Input thinned ridge network raster layer
`number`	number	no	1.0	Number of passes through the dataset
`output`	rasterDestination	yes	—	Watersheds
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.basins.fill cnetwork=dem.tif tnetwork=dem.tif number=1 output=output.tif
```

This calls **r.basins.fill**: `cnetwork=dem.tif` — Input coded stream network raster layer; `tnetwork=dem.tif` — Input thinned ridge network raster layer; `number=1` — Number of passes through the dataset; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.blend.combine — r.blend.combine

group: Raster (r.*)

r.blend.combine - Blends color components of two raster maps by a given ratio and export into a unique raster.

Parameter	Type	Required	Default	Description
`first`	raster	yes	—	Name of first raster map for blending
`second`	raster	yes	—	Name of second raster map for blending
`percent`	number	no	50.0	Percentage weight of first map for color blending
`output`	rasterDestination	yes	—	Blended
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.blend.combine first=dem.tif second=dem.tif percent=50 output=output.tif
```

This calls **r.blend.combine**: `first=dem.tif` — Name of first raster map for blending; `second=dem.tif` — Name of second raster map for blending; `percent=50` — Percentage weight of first map for color blending; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.blend.rgb — r.blend.rgb

group: Raster (r.*)

r.blend.rgb - Blends color components of two raster maps by a given ratio and exports into three rasters.

Parameter	Type	Required	Default	Description
`first`	raster	yes	—	Name of first raster map for blending
`second`	raster	yes	—	Name of second raster map for blending
`percent`	number	no	50.0	Percentage weight of first map for color blending
`output_red`	rasterDestination	yes	—	Blended Red
`output_green`	rasterDestination	yes	—	Blended Green
`output_blue`	rasterDestination	yes	—	Blended Blue
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output_red (outputRaster), output_green (outputRaster), output_blue (outputRaster)

Example usage

```
run grass:r.blend.rgb first=dem.tif second=dem.tif percent=50 output_red=output_red.tif
```

This calls **r.blend.rgb**: first=dem.tif — Name of first raster map for blending; second=dem.tif — Name of second raster map for blending; percent=50 — Percentage weight of first map for color blending; output_red=output_red.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.buffer — r.buffer

group: Raster (r.*)

Creates a raster map layer showing buffer zones surrounding cells that contain non-NULL category values.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
distances	string	yes	—	Distance zone(s) (e.g. 100,200,300)
units	enum	no	0	Units of distance — options: 0=meters, 1=kilometers, 2=feet, 3=miles, 4=nautmiles
-z	boolean	no	False	Ignore zero (0) data cells instead of NULL cells
output	rasterDestination	yes	—	Buffer
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.buffer input=dem.tif distances=value units=0 output=output.tif
```

This calls **r.buffer**: input=dem.tif — Input raster layer; distances=value — Distance zone(s) (e.g. 100,200,300); units=0 selects *meters*; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.buffer.lowmem — r.buffer.lowmem

group: Raster (r.*)

Creates a raster map layer showing buffer zones surrounding cells that contain non-NULL category values (low-memory alternative).

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer

Parameter	Type	Required	Default	Description
distances	string	yes	—	Distance zone(s) (e.g. 100,200,300)
units	enum	no	0	Units of distance — options: 0=meters, 1=kilometers, 2=feet, 3=miles, 4=naut-miles
-z	boolean	no	False	Ignore zero (0) data cells instead of NULL cells
output	rasterDestination	yes	—	Buffer
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for de-fault)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.buffer.lowmem input=dem.tif distances=value units=0 output=output.tif
```

This calls **r.buffer.lowmem**: input=dem.tif — Input raster layer; distances=value — Distance zone(s) (e.g. 100,200,300); units=0 selects *meters*; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.carve — r.carve

group: Raster (r.*)

Takes vector stream data, transforms it to raster and subtracts depth from the output DEM.

Parameter	Type	Required	Default	Description
raster	raster	yes	—	Elevation
vector	source	yes	—	Vector layer containing stream(s)
width	number	no	—	Stream width (in meters). Default is raster cell width
depth	number	no	—	Additional stream depth (in meters)
-n	boolean	no	False	No flat areas allowed in flow direction
output	rasterDestination	yes	—	Modified elevation
points	vectorDestination	yes	—	Adjusted stream points
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for de-fault)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with cat-egory are exported

Outputs: output (outputRaster), points (outputVector)

Example usage

```
run grass:r.carve raster=dem.tif vector=input.gpkg width=10 depth=10 output=output.tif
```

This calls **r.carve**: `raster=dem.tif` — Elevation; `vector=input.gpkg` — Vector layer containing stream(s); `width=10` — Stream width (in meters). Default is raster cell width; `depth=10` — Additional stream depth (in meters); `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.category — r.category

group: Raster (r.*)

Manages category values and labels associated with user-specified raster map layers.

Parameter	Type	Required	Default	Description
`map`	raster	yes	—	Name of raster map
`separator`	string	no	tab	Field separator (Special characters: pipe, comma, space, tab, newline)
`rules`	file	no	—	File containing category label rules
`txtrules`	string	no	—	Inline category label rules
`raster`	raster	no	—	Raster map from which to copy category table
`format *(adv)*`	string	no	—	Default label or format string for dynamic labeling. Used when no explicit label exists for the category
`coefficients *(adv)*`	string	no	—	Dynamic label coefficients. Two pairs of category multiplier and offsets, for $1 and $2
`output`	rasterDestination	yes	—	Category
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.category map=dem.tif separator=tab rules=input.dat txtrules=value
 ↪  raster=dem.tif output=output.tif
```

This calls **r.category**: `map=dem.tif` — Name of raster map; `separator=tab` — Field separator (Special characters: pipe, comma, space, tab, newline); `rules=input.dat` — File containing category label rules; `txtrules=value` — Inline category label rules; `raster=dem.tif` — Raster map from which to copy category table; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.category.out — r.category.out

group: Raster (r.*)

r.category.out - Exports category values and labels associated with user-specified raster map layers.

Parameter	Type	Required	Default	Description
`map`	raster	yes	—	Name of raster map
`cats`	string	no	—	Category values (for Integer rasters). Example: 1,3,7-9,13
`values`	string	no	—	Comma separated value list (for float rasters). Example: 1.4,3.8,13

Parameter	Type	Required	Default	Description
separator	string	no	tab	Field separator (Special characters: pipe, comma, space, tab, newline)
html	fileDestination	yes	—	Category
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: html (outputHtml)

Example usage

```
run grass:r.category.out map=dem.tif cats=value values=value separator=tab html=html.txt
```

This calls **r.category.out**: map=dem.tif — Name of raster map; cats=value — Category values (for Integer rasters). Example: 1,3,7-9,13; values=value — Comma separated value list (for float rasters). Example: 1.4,3.8,13; separator=tab — Field separator (Special characters: pipe, comma, space, tab, newline); html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.circle — r.circle

group: Raster (r.*)

Creates a raster map containing concentric rings around a given point.

Parameter	Type	Required	Default	Description
coordinates	point	no	0,0	The coordinate of the center (east,north)
min	number	no	—	Minimum radius for ring/circle map (in meters)
max	number	no	—	Maximum radius for ring/circle map (in meters)
multiplier	number	no	—	Data value multiplier
-b	boolean	no	False	Generate binary raster map
output	rasterDestination	yes	—	Circles
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.circle coordinates=100,100 min=10 max=10 multiplier=10 output=output.tif
```

This calls **r.circle**: coordinates=100,100 — The coordinate of the center (east,north); min=10 — Minimum radius for ring/circle map (in meters); max=10 — Maximum radius for ring/circle map (in meters); multiplier=10 — Data value multiplier; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.clump — r.clump

group: Raster (r.*)

Recategorizes data in a raster map by grouping cells that form physically discrete areas into unique categories.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input layer
title	string	yes	—	Title for output raster map
-d *(adv)*	boolean	no	False	Clump also diagonal cells
output	rasterDestination	yes	—	Clumps
threshold	number	no	0.0	Threshold to identify similar cells
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.clump input=dem.tif title=value threshold=0 output=output.tif
```

This calls **r.clump**: input=dem.tif — Input layer; title=value — Title for output raster map; threshold=0 — Threshold to identify similar cells; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.coin — r.coin

group: Raster (r.*)

Tabulates the mutual occurrence (coincidence) of categories for two raster map layers.

Parameter	Type	Required	Default	Description
first	raster	yes	—	Name of first raster map
second	raster	yes	—	Name of second raster map
units	enum	yes	—	Unit of measure — options: 0=c, 1=p, 2=x, 3=y, 4=a, 5=h, 6=k, 7=m
-w	boolean	no	False	Wide report, 132 columns (default: 80)
html	fileDestination	no	report.html	Coincidence report
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: html (outputHtml)

Example usage

```
run grass:r.coin first=dem.tif second=dem.tif units=0 html=html.txt
```

This calls **r.coin**: first=dem.tif — Name of first raster map; second=dem.tif — Name of second raster map; units=0 selects *c*; html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.colors — r.colors

group: Raster (r.*)

Creates/modifies the color table associated with a raster map.

Parameter	Type	Required	Default	Description
map	multilayer	yes	—	Name of raster maps(s)

Parameter	Type	Required	Default	Description
`color`	enum	no	0	Name of color table — options: 0=not selected, 1=aspect, 2=aspectcolr, 3=bcyr, 4=bgyr, 5=blues, 6=byg, 7=byr, 8=celsius, 9=corine, 10=curvature, 11=differences, 12=elevation, 13=etopo2, 14=evi, 15=fahrenheit, 16=gdd, 17=greens, 18=grey, 19=grey.eq, 20=grey.log, 21=grey1.0, 22=grey255, 23=gyr, 24=haxby, 25=kelvin, 26=ndvi, 27=ndwi, 28=oranges, 29=population, 30=population_dens, 31=precipitation, 32=precipitation_daily, 33=precipitation_monthly, 34=rainbow, 35=ramp, 36=random, 37=reds, 38=rstcurv, 39=ryb, 40=ryg, 41=sepia, 42=slope, 43=srtm, 44=srtm_plus, 45=terrain, 46=wave
`rules_txt`	string	no	—	Color rules
`rules`	file	no	—	Color rules file
`raster`	raster	no	—	Raster map from which to copy color table
`-r`	boolean	no	False	Remove existing color table
`-w`	boolean	no	False	Only write new color table if it does not already exist
`-n`	boolean	no	False	Invert colors
`-g`	boolean	no	False	Logarithmic scaling
`-a`	boolean	no	False	Logarithmic-absolute scaling
`-e`	boolean	no	False	Histogram equalization
`output_dir`	folderDestination	yes	—	Output Directory
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output_dir` (outputFolder)

Example usage

```
run grass:r.colors map="a.gpkg;b.gpkg" color=0 rules_txt=value rules=input.dat
 ↪  raster=dem.tif output_dir=output_dir/
```

This calls **r.colors**: `map="a.gpkg;b.gpkg"` — Name of raster maps(s); `color=0` selects *not selected*; `rules_txt=value` — Color rules; `rules=input.dat` — Color rules file; `raster=dem.tif` — Raster map from which to copy color table; `output_dir=output_dir/` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.colors.out` — r.colors.out

group: Raster (r.*)

Exports the color table associated with a raster map.

Parameter	Type	Required	Default	Description
`map`	raster	yes	—	Name of raster map
`-p *(adv)*`	boolean	no	False	Output values as percentages
`rules`	fileDestination	yes	—	Color Table
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `rules` (outputFile)

Example usage

```
run grass:r.colors.out map=dem.tif rules=rules.txt
```

This calls **r.colors.out**: `map=dem.tif` — Name of raster map; `rules=rules.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.colors.stddev — r.colors.stddev

group: Raster (r.*)

Sets color rules based on stddev from a raster map's mean value.

Parameter	Type	Required	Default	Description
map	raster	yes	—	Name of raster map
-b *(adv)*	boolean	no	False	Color using standard deviation bands
-z *(adv)*	boolean	no	False	Force center at zero
output	rasterDestination	yes	—	Stddev Colors
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.colors.stddev map=dem.tif output=output.tif
```

This calls **r.colors.stddev**: `map=dem.tif` — Name of raster map; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.composite — r.composite

group: Raster (r.*)

Combines red, green and blue raster maps into a single composite raster map.

Parameter	Type	Required	Default	Description
red	raster	yes	—	Red
green	raster	yes	—	Green
blue	raster	yes	—	Blue
levels	number	no	32.0	Number of levels to be used for each component
level_red	number	no	—	Number of levels to be used for
level_green	number	no	—	Number of levels to be used for
level_blue	number	no	—	Number of levels to be used for
-d	boolean	no	False	Dither
-c	boolean	no	False	Use closest color
output	rasterDestination	yes	—	Composite
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.composite red=dem.tif green=dem.tif blue=dem.tif levels=32 level_red=10
↪  level_green=10 level_blue=10 output=output.tif
```

This calls **r.composite**: `red=dem.tif` — Red; `green=dem.tif` — Green; `blue=dem.tif` — Blue; `levels=32` — Number of levels to be used for each component; `level_red=10` — Number of levels to be used for; `level_green=10` — Number of levels to be used for; `level_blue=10` — Number of levels to be used for; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.contour` — r.contour

group: Raster (r.*)

Produces a vector map of specified contours from a raster map.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster
`step`	number	no	—	Increment between contour levels
`levels`	string	no	—	List of contour levels
`minlevel`	number	no	—	Minimum contour level
`maxlevel`	number	no	—	Maximum contour level
`cut`	number	no	0.0	Minimum number of points for a contour line (0 -> no limit)
`output`	vectorDestination	yes	—	Contours
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:r.contour input=dem.tif step=10 levels=value minlevel=10 maxlevel=10 cut=0
↪  output=output.gpkg
```

This calls **r.contour**: `input=dem.tif` — Input raster; `step=10` — Increment between contour levels; `levels=value` — List of contour levels; `minlevel=10` — Minimum contour level; `maxlevel=10` — Maximum contour level; `cut=0` — Minimum number of points for a contour line (0 -> no limit); `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.cost` — r.cost

group: Raster (r.*)

Creates a raster layer of cumulative cost of moving across a raster layer whose cell values represent cost.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Unit cost layer
start_coordinates	point	no	—	Coordinates of starting point(s) (E,N)
stop_coordinates	point	no	—	Coordinates of stopping point(s) (E,N)
-k	boolean	no	False	Use the 'Knight's move'; slower, but more accurate
-n	boolean	no	True	Keep null values in output raster layer
start_points	source	no	—	Start points
stop_points	source	no	—	Stop points
start_raster	raster	no	—	Name of starting raster points map
max_cost	number	no	0.0	Maximum cumulative cost
null_cost	number	no	—	Cost assigned to null cells. By default, null cells are excluded
memory	number	no	300.0	Maximum memory to be used in MB
output	rasterDestination	yes	—	Cumulative cost
nearest	rasterDestination	no	—	Cost allocation map
outdir	rasterDestination	no	—	Movement directions
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: output (outputRaster), nearest (outputRaster), outdir (outputRaster)

Example usage

```
run grass:r.cost input=dem.tif start_coordinates=100,100 stop_coordinates=100,100
 ↪  start_points=input.gpkg stop_points=input.gpkg start_raster=dem.tif max_cost=0
 ↪  output=output.tif
```

This calls **r.cost**: input=dem.tif — Unit cost layer; start_coordinates=100,100 — Coordinates of starting point(s) (E,N); stop_coordinates=100,100 — Coordinates of stopping point(s) (E,N); start_points=input.gpkg — Start points; stop_points=input.gpkg — Stop points; start_raster=dem.tif — Name of starting raster points map; max_cost=0 — Maximum cumulative cost; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.covar — r.covar

group: Raster (r.*)

Outputs a covariance/correlation matrix for user-specified raster layer(s).

Parameter	Type	Required	Default	Description
map	multilayer	yes	—	Input layers
-r	boolean	no	True	Print correlation matrix
html	fileDestination	no	report.html	Covariance report
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: html (outputHtml)

Example usage

```
run grass:r.covar map="a.gpkg;b.gpkg" html=html.txt
```

This calls **r.covar**: `map="a.gpkg;b.gpkg"` — Input layers; `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.cross` — r.cross

group: Raster (r.*)

Creates a cross product of the category values from multiple raster map layers.

Parameter	Type	Required	Default	Description
`input`	multilayer	yes	—	Input raster layers
`-z`	boolean	no	False	Non-zero data only
`output`	rasterDestination	yes	—	Cross product
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.cross input="a.gpkg;b.gpkg" output=output.tif
```

This calls **r.cross**: `input="a.gpkg;b.gpkg"` — Input raster layers; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.describe` — r.describe

group: Raster (r.*)

Prints terse list of category values found in a raster layer.

Parameter	Type	Required	Default	Description
`map`	raster	yes	—	input raster layer
`null_value`	string	no	*	String representing NULL value
`nsteps`	number	no	255.0	Number of quantization steps
`-r`	boolean	no	False	Only print the range of the data
`-n`	boolean	no	False	Suppress reporting of any NULLs
`-d`	boolean	no	False	Use the current region
`-i`	boolean	no	False	Read floating-point map as integer
`html`	fileDestination	no	report.html	Categories
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `html` (outputHtml)

Example usage

```
run grass:r.describe map=dem.tif null_value=* nsteps=255 html=html.txt
```

This calls **r.describe**: `map=dem.tif` — input raster layer; `null_value=*` — String representing NULL value; `nsteps=255` — Number of quantization steps; `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.distance` — r.distance

group: Raster (r.*)

Locates the closest points between objects in two raster maps.

Parameter	Type	Required	Default	Description
`map`	multilayer	yes	—	Name of two input raster for computing inter-class distances
`separator`	string	no	:	Field separator (Special characters: pipe, comma, space, tab, newline)
`sort`	enum	no	0	Sort output by distance — options: 0=asc, 1=desc
`-l *(adv)*`	boolean	no	False	Include category labels in the output
`-o *(adv)*`	boolean	no	False	Report zero distance if rasters are overlapping
`-n *(adv)*`	boolean	no	False	Report null objects as *
`html`	fileDestination	yes	—	Distance
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `html` (outputHtml)

Example usage

```
run grass:r.distance map="a.gpkg;b.gpkg" separator=: sort=0 html=html.txt
```

This calls **r.distance**: `map="a.gpkg;b.gpkg"` — Name of two input raster for computing inter-class distances; `separator=:` — Field separator (Special characters: pipe, comma, space, tab, newline); `sort=0` selects *asc*; `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.drain` — r.drain

group: Raster (r.*)

Traces a flow through an elevation model on a raster map.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Elevation
`direction`	raster	no	—	Name of input movement direction map associated with the cost surface
`start_coordinates`	point	no	—	Map coordinates of starting point(s) (E,N)
`start_points`	source	no	—	Vector layer containing starting point(s)
`-c`	boolean	no	False	Copy input cell values on output
`-a`	boolean	no	False	Accumulate input values along the path
`-n`	boolean	no	False	Count cell numbers along the path
`-d`	boolean	no	False	The input raster map is a cost surface (direction surface must also be specified)
`output`	rasterDestination	yes	—	Least cost path
`drain`	vectorDestination	yes	—	Drain
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputRaster), `drain` (outputVector)

Example usage

```
run grass:r.drain input=dem.tif direction=dem.tif start_coordinates=100,100
↪  start_points=input.gpkg output=output.tif
```

This calls **r.drain**: `input=dem.tif` — Elevation; `direction=dem.tif` — Name of input movement direction map associated with the cost surface; `start_coordinates=100,100` — Map coordinates of starting point(s) (E,N); `start_points=input.gpkg` — Vector layer containing starting point(s); `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.fill.dir` — r.fill.dir

group: Raster (r.*)

Filters and generates a depressionless elevation layer and a flow direction layer from a given elevation raster layer.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Elevation
format	enum	no	0	Output aspect direction format — options: 0=grass, 1=agnps, 2=answers
-f *(adv)*	boolean	no	False	Find unresolved areas only
output	rasterDestination	yes	—	Depressionless DEM
direction	rasterDestination	yes	—	Flow direction
areas	rasterDestination	yes	—	Problem areas
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster), `direction` (outputRaster), `areas` (outputRaster)

Example usage

```
run grass:r.fill.dir input=dem.tif format=0 output=output.tif
```

This calls **r.fill.dir**: `input=dem.tif` — Elevation; `format=0` selects *grass*; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.fill.stats — r.fill.stats

group: Raster (r.*)

Rapidly fills 'no data' cells (NULLs) of a raster map with interpolated values (IDW).

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer with data gaps to fill
-k	boolean	no	False	Preserve original cell values (By default original values are smoothed)
mode	enum	no	0	Statistic for interpolated cell values — options: 0=wmean, 1=mean, 2=median, 3=mode
-m	boolean	no	False	Interpret distance as map units, not number of cells (Do not select with geodetic coordinates)
distance	number	no	3.0	Distance threshold (default: in cells) for interpolation
minimum	number	no	—	Minimum input data value to include in interpolation
maximum	number	no	—	Maximum input data value to include in interpolation
power	number	no	2.0	Power coefficient for IDW interpolation
cells	number	no	8.0	Minimum number of data cells within search radius
output	rasterDestination	yes	—	Output Map
uncertainty	rasterDestination	no	—	Uncertainty Map
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster), uncertainty (outputRaster)

Example usage

```
run grass:r.fill.stats input=dem.tif mode=0 distance=3 minimum=10 maximum=10 power=2
↪  cells=8 output=output.tif
```

This calls **r.fill.stats**: input=dem.tif — Input raster layer with data gaps to fill; mode=0 selects *wmean*; distance=3 — Distance threshold (default: in cells) for interpolation; minimum=10 — Minimum input data value to include in interpolation; maximum=10 — Maximum input data value to include in interpolation; power=2 — Power coefficient for IDW interpolation; cells=8 — Minimum number of data cells within search radius; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.fillnulls — r.fillnulls

group: Raster (r.*)

Fills no-data areas in raster maps using spline interpolation.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer to fill
method	enum	no	2	Interpolation method to use — options: 0=bilinear, 1=bicubic, 2=rst
tension	number	no	40.0	Spline tension parameter
smooth	number	no	0.1	Spline smoothing parameter

Parameter	Type	Required	Default	Description
edge	number	no	3.0	Width of hole edge used for interpolation (in cells)
npmin	number	no	600.0	Minimum number of points for approximation in a segment (>segmax)
segmax	number	no	300.0	Maximum number of points in a segment
lambda	number	no	0.01	Tykhonov regularization parameter (affects smoothing)
output	rasterDestination	yes	—	Filled
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.fillnulls input=dem.tif method=2 tension=40 smooth=0.1 edge=3 npmin=600
↪  segmax=300 output=output.tif
```

This calls **r.fillnulls**: input=dem.tif — Input raster layer to fill; method=2 selects *rst*; tension=40 — Spline tension parameter; smooth=0.1 — Spline smoothing parameter; edge=3 — Width of hole edge used for interpolation (in cells); npmin=600 — Minimum number of points for approximation in a segment (>segmax); segmax=300 — Maximum number of points in a segment; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.flow — r.flow

group: Raster (r.*)

Construction of flowlines, flowpath lengths, and flowaccumulation (contributing areas) from a raster digital elevation model (DEM).

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Elevation
aspect	raster	no	—	Aspect
barrier	raster	no	—	Barrier
skip	number	no	—	Number of cells between flowlines
bound	number	no	—	Maximum number of segments per flowline
-u	boolean	no	False	Compute upslope flowlines instead of default downhill flowlines
-3	boolean	no	False	3-D lengths instead of 2-D
-m *(adv)*	boolean	no	False	Use less memory, at a performance penalty
flowline	vectorDestination	no	—	Flow line
flowlength	rasterDestination	yes	—	Flow path length
flowaccumulation	rasterDestination	yes	—	Flow accumulation
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)

Parameter	Type	Required	Default	Description
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: flowline (outputVector), flowlength (outputRaster), flowaccumulation (outputRaster)

Example usage

```
run grass:r.flow elevation=dem.tif aspect=dem.tif barrier=dem.tif skip=10 bound=10
 ↪  flowline=flowline.gpkg
```

This calls **r.flow**: elevation=dem.tif — Elevation; aspect=dem.tif — Aspect; barrier=dem.tif — Barrier; skip=10 — Number of cells between flowlines; bound=10 — Maximum number of segments per flowline; flowline=flowline.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.geomorphon — r.geomorphon

group: Raster (r.*)

Calculates geomorphons (terrain forms) and associated geometry using machine vision approach.

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Name of input elevation raster map
search	number	no	3.0	Outer search radius
skip	number	no	0.0	Inner search radius
flat	number	no	1.0	Flatness threshold (degrees)
dist	number	no	0.0	Flatness distance, zero for none
forms	rasterDestination	yes	—	Most common geomorphic forms
-m *(adv)*	boolean	no	False	Use meters to define search units (default is cells)
-e *(adv)*	boolean	no	False	Use extended form correction
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: forms (outputRaster)

Example usage

```
run grass:r.geomorphon elevation=dem.tif search=3 skip=0 flat=1 dist=0 forms=forms.tif
```

This calls **r.geomorphon**: elevation=dem.tif — Name of input elevation raster map; search=3 — Outer search radius; skip=0 — Inner search radius; flat=1 — Flatness threshold (degrees); dist=0 — Flatness distance, zero for none; forms=forms.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.grow — r.grow

group: Raster (r.*)

Generates a raster layer with contiguous areas grown by one cell.

Parameter	Type	Required	Default	Description
input	raster	yes	—	input raster layer
radius	number	no	1.01	Radius of buffer in raster cells
metric	enum	no	0	Metric — options: 0=euclidean, 1=maximum, 2=manhattan
old	number	no	—	Value to write for input cells which are non-NULL (-1 => NULL)
new	number	no	—	Value to write for ”grown” cells
-m *(adv)*	boolean	no	False	Radius is in map units rather than cells
output	rasterDestination	yes	—	Expanded
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.grow input=dem.tif radius=1.01 metric=0 old=10 new=10 output=output.tif
```

This calls **r.grow**: input=dem.tif — input raster layer; radius=1.01 — Radius of buffer in raster cells; metric=0 selects *euclidean*; old=10 — Value to write for input cells which are non-NULL (-1 => NULL); new=10 — Value to write for ”grown” cells; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.grow.distance — r.grow.distance

group: Raster (r.*)

Generates a raster layer of distance to features in input layer.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input input raster layer
metric	enum	no	0	Metric — options: 0=euclidean, 1=squared, 2=maximum, 3=manhattan, 4=geodesic
-m *(adv)*	boolean	no	False	Output distances in meters instead of map units
-n *(adv)*	boolean	no	False	Calculate distance to nearest NULL cell
distance	rasterDestination	yes	—	Distance
value	rasterDestination	yes	—	Value of nearest cell
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: distance (outputRaster), value (outputRaster)

Example usage

```
run grass:r.grow.distance input=dem.tif metric=0 distance=distance.tif
```

This calls **r.grow.distance**: input=dem.tif — Input input raster layer; metric=0 selects *euclidean*; distance=distance.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.gwflow — r.gwflow

group: Raster (r.*)

Numerical calculation program for transient, confined and unconfined groundwater flow in two dimensions.

Parameter	Type	Required	Default	Description
phead	raster	yes	—	The initial piezometric head in [m]
status	raster	yes	—	Boundary condition status, 0-inactive, 1-active, 2-dirichlet
hc_x	raster	yes	—	X-part of the hydraulic conductivity tensor in [m/s]
hc_y	raster	yes	—	Y-part of the hydraulic conductivity tensor in [m/s]
q	raster	no	—	Water sources and sinks in [m^3/s]
s	raster	yes	—	Specific yield in [1/m]
recharge	raster	no	—	Recharge map e.g: *610^-9 per cell in [m^3/s*m^2]
top	raster	yes	—	Top surface of the aquifer in [m]
bottom	raster	yes	—	Bottom surface of the aquifer in [m]
type	enum	no	0	The type of groundwater flow — options: 0=confined, 1=unconfined
river_bed	raster	no	—	The height of the river bed in [m]
river_head	raster	no	—	Water level (head) of the river with leakage connection in [m]
river_leak	raster	no	—	The leakage coefficient of the river bed in [1/s]
drain_bed	raster	no	—	The height of the drainage bed in [m]
drain_leak	raster	no	—	The leakage coefficient of the drainage bed in [1/s]
dtime	number	no	86400.0	The calculation time in seconds
maxit	number	no	100000.0	Maximum number of iteration used to solver the linear equation system
error	number	no	1e-06	Error break criteria for iterative solvers (jacobi, sor, cg or bicgstab)
solver	enum	no	0	The type of solver which should solve the symmetric linear equation system — options: 0=cg, 1=pcg, 2=cholesky
relax	string	no	1	The relaxation parameter used by the jacobi and sor solver for speedup or stabilizing
-f	boolean	no	False	Allocate a full quadratic linear equation system, default is a sparse linear equation system
output	rasterDestination	yes	—	Groundwater flow
vx	rasterDestination	yes	—	Groundwater filter velocity vector part in x direction [m/s]
vy	rasterDestination	yes	—	Groundwater filter velocity vector part in y direction [m/s]
budget	rasterDestination	yes	—	Groundwater budget for each cell [m^3/s]
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster), vx (outputRaster), vy (outputRaster), budget (outputRaster)

Example usage

```
run grass:r.gwflow phead=dem.tif status=dem.tif hc_x=dem.tif hc_y=dem.tif s=dem.tif
 ↪  top=dem.tif bottom=dem.tif output=output.tif
```

This calls **r.gwflow**: phead=dem.tif — The initial piezometric head in [m]; status=dem.tif — Boundary condition status, 0-inactive, 1-active, 2-dirichlet; hc_x=dem.tif — X-part of the hydraulic conductivity tensor in [m/s]; hc_y=dem.tif — Y-part of the hydraulic conductivity tensor in [m/s]; s=dem.tif — Specific yield in [1/m]; top=dem.tif — Top surface of the aquifer in [m]; bottom=dem.tif — Bottom surface of the aquifer in [m]; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.his — r.his

group: Raster (r.*)

Generates red, green and blue raster layers combining hue, intensity and saturation (HIS) values from user-specified input raster layers.

Parameter	Type	Required	Default	Description
hue	raster	yes	—	Hue
intensity	raster	yes	—	Intensity
saturation	raster	yes	—	Saturation
bgcolor	string	no	—	Color to use instead of NULL values. Either a standard color name, R:G:B triplet, or "none"
-c	boolean	no	False	Use colors from color tables for NULL values
red	rasterDestination	yes	—	Red
green	rasterDestination	yes	—	Green
blue	rasterDestination	yes	—	Blue
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: red (outputRaster), green (outputRaster), blue (outputRaster)

Example usage

```
run grass:r.his hue=dem.tif intensity=dem.tif saturation=dem.tif bgcolor=value red=red.tif
```

This calls **r.his**: hue=dem.tif — Hue; intensity=dem.tif — Intensity; saturation=dem.tif — Saturation; bgcolor=value — Color to use instead of NULL values. Either a standard color name, R:G:B...; red=red.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.horizon — r.horizon

group: Raster (r.*)

Horizon angle computation from a digital elevation model.

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Name of input elevation raster map
direction	number	no	—	Direction in which you want to know the horizon height
step	number	no	—	Angle step size for multidirectional horizon
start	number	no	0.0	Start angle for multidirectional horizon
end	number	no	360.0	End angle for multidirectional horizon
bufferzone	number	no	—	For horizon rasters, read from the DEM an extra buffer around the present region

Parameter	Type	Required	Default	Description
e_buff	number	no	—	For horizon rasters, read from the DEM an extra buffer eastward the present region
w_buff	number	no	—	For horizon rasters, read from the DEM an extra buffer westward the present region
n_buff	number	no	—	For horizon rasters, read from the DEM an extra buffer northward the present region
s_buff	number	no	—	For horizon rasters, read from the DEM an extra buffer southward the present region
maxdistance	number	no	—	The maximum distance to consider when finding the horizon height
distance	number	no	1.0	Sampling distance step coefficient
-d	boolean	no	False	Write output in degrees (default is radians)
-c	boolean	no	False	Write output in compass orientation (default is CCW, East=0)
output	folderDestination	yes	—	Folder to get horizon rasters
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFolder)

Example usage

```
run grass:r.horizon elevation=dem.tif direction=10 step=10 start=0 end=360 bufferzone=10
↪  e_buff=10 output=output/
```

This calls **r.horizon**: elevation=dem.tif — Name of input elevation raster map; direction=10 — Direction in which you want to know the horizon height; step=10 — Angle step size for multidirectional horizon; start=0 — Start angle for multidirectional horizon; end=360 — End angle for multidirectional horizon; bufferzone=10 — For horizon rasters, read from the DEM an extra buffer around the present region; e_buff=10 — For horizon rasters, read from the DEM an extra buffer eastward the present...; output=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.horizon.height — r.horizon.height

group: Raster (r.*)

r.horizon.height - Horizon angle computation from a digital elevation model.

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Name of input elevation raster map
coordinates	point	no	0,0	Coordinate for which you want to calculate the horizon
direction	number	no	—	Direction in which you want to know the horizon height
step	number	no	—	Angle step size for multidirectional horizon
start	number	no	0.0	Start angle for multidirectional horizon
end	number	no	360.0	End angle for multidirectional horizon
bufferzone	number	no	—	For horizon rasters, read from the DEM an extra buffer around the present region
e_buff	number	no	—	For horizon rasters, read from the DEM an extra buffer eastward the present region
w_buff	number	no	—	For horizon rasters, read from the DEM an extra buffer westward the present region
n_buff	number	no	—	For horizon rasters, read from the DEM an extra buffer northward the present region
s_buff	number	no	—	For horizon rasters, read from the DEM an extra buffer southward the present region

Parameter	Type	Required	Default	Description
maxdistance	number	no	—	The maximum distance to consider when finding the horizon height
distance	number	no	1.0	Sampling distance step coefficient
-d	boolean	no	False	Write output in degrees (default is radians)
-c	boolean	no	False	Write output in compass orientation (default is CCW, East=0)
html	fileDestination	no	report.html	Horizon
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: html (outputHtml)

Example usage

```
run grass:r.horizon.height elevation=dem.tif coordinates=100,100 direction=10 step=10
 ↪  start=0 end=360 bufferzone=10 html=html.txt
```

This calls **r.horizon.height**: elevation=dem.tif — Name of input elevation raster map; coordinates=100,100 — Coordinate for which you want to calculate the horizon; direction=10 — Direction in which you want to know the horizon height; step=10 — Angle step size for multidirectional horizon; start=0 — Start angle for multidirectional horizon; end=360 — End angle for multidirectional horizon; bufferzone=10 — For horizon rasters, read from the DEM an extra buffer around the present region; html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.in.lidar — r.in.lidar

group: Raster (r.*)

Creates a raster map from LAS LiDAR points using univariate statistics.

Parameter	Type	Required	Default	Description
input	file	yes	—	LAS input file
method	enum	no	5	Statistic to use for raster values — options: 0=n, 1=min, 2=max, 3=range, 4=sum, 5=mean, 6=stddev, 7=variance, 8=coeff_var, 9=median, 10=percentile, 11=skewness, 12=trimmean
type	enum	no	1	Storage type for resultant raster map — options: 0=CELL, 1=FCELL, 2=DCELL
base_raster	raster	no	—	Subtract raster values from the Z coordinates
zrange	range	no	—	Filter range for z data (min, max)
zscale	number	no	1.0	Scale to apply to z data
intensity_range	range	no	—	Filter range for intensity values (min, max)
intensity_scale	number	no	1.0	Scale to apply to intensity values
percent	number	no	100.0	Percent of map to keep in memory
pth	number	no	—	pth percentile of the values (between 1 and 100)
trim	number	no	—	Discard percent of the smallest and percent of the largest observations (0-50)
resolution	number	no	—	Output raster resolution
return_filter	string	no	—	Only import points of selected return type Options: first, last, mid
class_filter	string	no	—	Only import points of selected class(es) (comma separated integers)
-e *(adv)*	boolean	no	False	Use the extent of the input for the raster extent

Parameter	Type	Required	Default	Description
`-n *(adv)*`	boolean	no	False	Set computation region to match the new raster map
`-o *(adv)*`	boolean	no	False	Override projection check (use current location's projection)
`-i *(adv)*`	boolean	no	False	Use intensity values rather than Z values
`-j *(adv)*`	boolean	no	False	Use Z values for filtering, but intensity values for statistics
`-d *(adv)*`	boolean	no	False	Use base raster resolution instead of computational region
`-v *(adv)*`	boolean	no	False	Use only valid points
`output`	rasterDestination	yes	—	Lidar Raster
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.in.lidar input=input.dat method=5 type=1 base_raster=dem.tif zscale=1
↪  output=output.tif
```

This calls **r.in.lidar**: `input=input.dat` — LAS input file; `method=5` selects *mean*; `type=1` selects *FCELL*; `base_raster=dem.tif` — Subtract raster values from the Z coordinates; `zscale=1` — Scale to apply to z data; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.in.lidar.info` — r.in.lidar.info

group: Raster (r.*)

r.in.lidar.info - Extract information from LAS file

Parameter	Type	Required	Default	Description
`input`	file	yes	—	LAS input file
`html`	fileDestination	no	report.html	LAS information
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Outputs: `html` (outputHtml)

Example usage

```
run grass:r.in.lidar.info input=input.dat html=html.txt
```

This calls **r.in.lidar.info**: `input=input.dat` — LAS input file; `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.info` — r.info

group: Raster (r.*)

Output basic information about a raster layer.

Parameter	Type	Required	Default	Description
`map`	raster	yes	—	Raster layer

Parameter	Type	Required	Default	Description
`-r`	boolean	no	False	Print range only
`-g`	boolean	no	False	Print raster array information in shell script style
`-h`	boolean	no	False	Print raster history instead of info
`-e`	boolean	no	False	Print extended metadata information in shell script style
`html`	fileDestination	no	report.html	Basic information
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `html` (outputHtml)

Example usage

```
run grass:r.info map=dem.tif html=html.txt
```

This calls **r.info**: `map=dem.tif` — Raster layer; `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.kappa` — r.kappa

group: Raster (r.*)

Calculate error matrix and kappa parameter for accuracy assessment of classification result.

Parameter	Type	Required	Default	Description
`classification`	raster	yes	—	Raster layer containing classification result
`reference`	raster	yes	—	Raster layer containing reference classes
`title`	string	no	ACCURACY ASSESSMENT	Title for error matrix and kappa
`-h`	boolean	no	False	No header in the report
`-w`	boolean	no	False	Wide report (132 columns)
`output`	fileDestination	yes	—	Error matrix and kappa
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output` (outputFile)

Example usage

```
run grass:r.kappa classification=dem.tif reference=dem.tif title="ACCURACY ASSESSMENT"
↪  output=output.txt
```

This calls **r.kappa**: `classification=dem.tif` — Raster layer containing classification result; `reference=dem.tif` — Raster layer containing reference classes; `title="ACCURACY ASSESSMENT"` — Title for error matrix and kappa; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.lake` — r.lake

group: Raster (r.*)

Fills lake at given point to given level.

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Elevation
water_level	number	yes	—	Water level
coordinates	point	no	—	Seed point coordinates
seed	raster	no	—	Raster layer with starting point(s) (at least 1 cell > 0)
-n	boolean	no	False	Use negative depth values for lake raster layer
lake	rasterDestination	yes	—	Lake
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: lake (outputRaster)

Example usage

```
run grass:r.lake elevation=dem.tif water_level=10 coordinates=100,100 seed=dem.tif
↪  lake=lake.tif
```

This calls **r.lake**: elevation=dem.tif — Elevation; water_level=10 — Water level; coordinates =100,100 — Seed point coordinates; seed=dem.tif — Raster layer with starting point(s) (at least 1 cell > 0); lake=lake.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.latlong — r.latlong

group: Raster (r.*)

Creates a latitude/longitude raster map.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
-l	boolean	no	False	Longitude output
output	rasterDestination	yes	—	LatLong
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.latlong input=dem.tif output=output.tif
```

This calls **r.latlong**: input=dem.tif — Name of input raster map; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.cwed — r.li.cwed

group: Raster (r.*)

Calculates contrast weighted edge density index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
path	file	yes	—	Name of file that contains the weight to calculate the index
output	rasterDestination	yes	—	CWED
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.cwed input=dem.tif path=input.dat config_txt=value config=input.dat
↪  output=output.tif
```

This calls **r.li.cwed**: input=dem.tif — Name of input raster map; path=input.dat — Name of file that contains the weight to calculate the index; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.cwed.ascii — r.li.cwed.ascii

group: Raster (r.*)

r.li.cwed.ascii - Calculates contrast weighted edge density index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
path	file	yes	—	Name of file that contains the weight to calculate the index
output_txt	fileDestination	yes	—	CWED
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.cwed.ascii input=dem.tif path=input.dat config_txt=value config=input.dat
↪  output_txt=output_txt.txt
```

This calls **r.li.cwed.ascii**: input=dem.tif — Name of input raster map; path=input.dat — Name of file that contains the weight to calculate the index; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.dominance — r.li.dominance

group: Raster (r.*)

Calculates dominance's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Dominance
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.dominance input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.dominance**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.dominance.ascii — r.li.dominance.ascii

group: Raster (r.*)

r.li.dominance.ascii - Calculates dominance's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Dominance
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.dominance.ascii input=dem.tif config_txt=value config=input.dat
↪ output_txt=output_txt.txt
```

This calls **r.li.dominance.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.edgedensity — r.li.edgedensity

group: Raster (r.*)

Calculates edge density index on a raster map, using a 4 neighbour algorithm

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
patch_type	string	no	—	The value of the patch type
-b	boolean	no	False	Exclude border edges
output	rasterDestination	yes	—	Edge Density
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.edgedensity input=dem.tif config_txt=value config=input.dat
↪  patch_type=value output=output.tif
```

This calls **r.li.edgedensity**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; patch_type=value — The value of the patch type; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.edgedensity.ascii — r.li.edgedensity.ascii

group: Raster (r.*)

r.li.edgedensity.ascii - Calculates edge density index on a raster map, using a 4 neighbour algorithm

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
patch_type	string	no	—	The value of the patch type
-b	boolean	no	False	Exclude border edges
output_txt	fileDestination	yes	—	Edge Density
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.edgedensity.ascii input=dem.tif config_txt=value config=input.dat
↪  patch_type=value output_txt=output_txt.txt
```

This calls **r.li.edgedensity.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; patch_type=value — The value of the patch type; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.mpa — r.li.mpa

group: Raster (r.*)

Calculates mean pixel attribute index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Mean Pixel Attribute
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.mpa input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.mpa**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.mpa.ascii — r.li.mpa.ascii

group: Raster (r.*)

r.li.mpa.ascii - Calculates mean pixel attribute index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Mean Pixel Attribute
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.mpa.ascii input=dem.tif config_txt=value config=input.dat
↪ output_txt=output_txt.txt
```

This calls **r.li.mpa.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.mps — r.li.mps

group: Raster (r.*)

Calculates mean patch size index on a raster map, using a 4 neighbour algorithm

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Mean Patch Size
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.mps input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.mps**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.mps.ascii — r.li.mps.ascii

group: Raster (r.*)

r.li.mps.ascii - Calculates mean patch size index on a raster map, using a 4 neighbour algorithm

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	yes	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Mean Patch Size
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.mps.ascii input=dem.tif config=input.dat config_txt=value
 ↪  output_txt=output_txt.txt
```

This calls **r.li.mps.ascii**: input=dem.tif — Name of input raster map; config=input.dat — Landscape structure configuration file; config_txt=value — Landscape structure configuration; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.padcv — r.li.padcv

group: Raster (r.*)

Calculates coefficient of variation of patch area on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	PADCV
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.padcv input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.padcv**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.padcv.ascii — r.li.padcv.ascii

group: Raster (r.*)

r.li.padcv.ascii - Calculates coefficient of variation of patch area on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	PADCV
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.padcv.ascii input=dem.tif config_txt=value config=input.dat
↪  output_txt=output_txt.txt
```

This calls **r.li.padcv.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.padrange — r.li.padrange

group: Raster (r.*)

Calculates range of patch area size on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration

Parameter	Type	Required	Default	Description
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Pad Range
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.li.padrange input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.padrange**: `input=dem.tif` — Name of input raster map; `config_txt=value` — Landscape structure configuration; `config=input.dat` — Landscape structure configuration file; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.li.padrange.ascii` — r.li.padrange.ascii

group: Raster (r.*)

r.li.padrange.ascii - Calculates range of patch area size on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Pad Range
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output_txt` (outputFile)

Example usage

```
run grass:r.li.padrange.ascii input=dem.tif config_txt=value config=input.dat
↪  output_txt=output_txt.txt
```

This calls **r.li.padrange.ascii**: `input=dem.tif` — Name of input raster map; `config_txt=value` — Landscape structure configuration; `config=input.dat` — Landscape structure configuration file; `output_txt=output_txt.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.li.padsd` — r.li.padsd

group: Raster (r.*)

Calculates standard deviation of patch area a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Patch Area SD

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.padsd input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.padsd**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.padsd.ascii — r.li.padsd.ascii

group: Raster (r.*)

r.li.padsd.ascii - Calculates standard deviation of patch area a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Patch Area SD
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.padsd.ascii input=dem.tif config_txt=value config=input.dat
↪  output_txt=output_txt.txt
```

This calls **r.li.padsd.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.patchdensity — r.li.patchdensity

group: Raster (r.*)

Calculates patch density index on a raster map, using a 4 neighbour algorithm

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Patch Density
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.patchdensity input=dem.tif config_txt=value config=input.dat
 ↪  output=output.tif
```

This calls **r.li.patchdensity**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.patchdensity.ascii — r.li.patchdensity.ascii

group: Raster (r.*)

r.li.patchdensity.ascii - Calculates patch density index on a raster map, using a 4 neighbour algorithm

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Patch Density
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.patchdensity.ascii input=dem.tif config_txt=value config=input.dat
 ↪  output_txt=output_txt.txt
```

This calls **r.li.patchdensity.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.patchnum — r.li.patchnum

group: Raster (r.*)

Calculates patch number index on a raster map, using a 4 neighbour algorithm.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Patch Number
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.patchnum input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.patchnum**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.patchnum.ascii — r.li.patchnum.ascii

group: Raster (r.*)

r.li.patchnum.ascii - Calculates patch number index on a raster map, using a 4 neighbour algorithm.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Patch Number
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.patchnum.ascii input=dem.tif config_txt=value config=input.dat
 ↪  output_txt=output_txt.txt
```

This calls **r.li.patchnum.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.pielou — r.li.pielou

group: Raster (r.*)

Calculates Pielou's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Pielou
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Parameter	Type	Required	Default	Description
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.pielou input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.pielou**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.pielou.ascii — r.li.pielou.ascii

group: Raster (r.*)

r.li.pielou.ascii - Calculates Pielou's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Pielou
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.pielou.ascii input=dem.tif config_txt=value config=input.dat
↪  output_txt=output_txt.txt
```

This calls **r.li.pielou.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.renyi — r.li.renyi

group: Raster (r.*)

Calculates Renyi's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
alpha	string	yes	—	Alpha value is the order of the generalized entropy
output	rasterDestination	yes	—	Renyi
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Parameter	Type	Required	Default	Description
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.renyi input=dem.tif alpha=value config_txt=value config=input.dat
 ↪  output=output.tif
```

This calls **r.li.renyi**: input=dem.tif — Name of input raster map; alpha=value — Alpha value is the order of the generalized entropy; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.renyi.ascii — r.li.renyi.ascii

group: Raster (r.*)

r.li.renyi.ascii - Calculates Renyi's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
alpha	string	yes	—	Alpha value is the order of the generalized entropy
output_txt	fileDestination	yes	—	Renyi
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.renyi.ascii input=dem.tif alpha=value config_txt=value config=input.dat
 ↪  output_txt=output_txt.txt
```

This calls **r.li.renyi.ascii**: input=dem.tif — Name of input raster map; alpha=value — Alpha value is the order of the generalized entropy; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.richness — r.li.richness

group: Raster (r.*)

Calculates richness index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Richness

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.li.richness input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.richness**: `input=dem.tif` — Name of input raster map; `config_txt=value` — Landscape structure configuration; `config=input.dat` — Landscape structure configuration file; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.richness.ascii — r.li.richness.ascii

group: Raster (r.*)

r.li.richness.ascii - Calculates richness index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Richness
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output_txt` (outputFile)

Example usage

```
run grass:r.li.richness.ascii input=dem.tif config_txt=value config=input.dat
↪  output_txt=output_txt.txt
```

This calls **r.li.richness.ascii**: `input=dem.tif` — Name of input raster map; `config_txt=value` — Landscape structure configuration; `config=input.dat` — Landscape structure configuration file; `output_txt=output_txt.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.shannon — r.li.shannon

group: Raster (r.*)

Calculates Shannon's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Shannon
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.shannon input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.shannon**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.shannon.ascii — r.li.shannon.ascii

group: Raster (r.*)

r.li.shannon.ascii - Calculates Shannon's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Shannon
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.shannon.ascii input=dem.tif config_txt=value config=input.dat
↪  output_txt=output_txt.txt
```

This calls **r.li.shannon.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.shape — r.li.shape

group: Raster (r.*)

Calculates shape index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Shape
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Parameter	Type	Required	Default	Description
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.shape input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.shape**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.shape.ascii — r.li.shape.ascii

group: Raster (r.*)

r.li.shape.ascii - Calculates shape index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Shape
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.shape.ascii input=dem.tif config_txt=value config=input.dat
↪  output_txt=output_txt.txt
```

This calls **r.li.shape.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.simpson — r.li.simpson

group: Raster (r.*)

Calculates Simpson's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output	rasterDestination	yes	—	Simpson
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)

Parameter	Type	Required	Default	Description
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.li.simpson input=dem.tif config_txt=value config=input.dat output=output.tif
```

This calls **r.li.simpson**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.li.simpson.ascii — r.li.simpson.ascii

group: Raster (r.*)

r.li.simpson.ascii - Calculates Simpson's diversity index on a raster map

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
config_txt	string	no	—	Landscape structure configuration
config	file	no	—	Landscape structure configuration file
output_txt	fileDestination	yes	—	Simpson
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output_txt (outputFile)

Example usage

```
run grass:r.li.simpson.ascii input=dem.tif config_txt=value config=input.dat
↪  output_txt=output_txt.txt
```

This calls **r.li.simpson.ascii**: input=dem.tif — Name of input raster map; config_txt=value — Landscape structure configuration; config=input.dat — Landscape structure configuration file; output_txt=output_txt.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.mapcalc.simple — r.mapcalc.simple

group: Raster (r.*)

Calculate new raster map from a r.mapcalc expression.

Parameter	Type	Required	Default	Description
a	raster	yes	—	Raster layer A
b	raster	no	—	Raster layer B
c	raster	no	—	Raster layer C
d	raster	no	—	Raster layer D
e	raster	no	—	Raster layer E
f	raster	no	—	Raster layer F
expression	string	no	A*2	Formula
output	rasterDestination	yes	—	Calculated
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.mapcalc.simple a=dem.tif b=dem.tif c=dem.tif d=dem.tif e=dem.tif f=dem.tif
↪  expression=A*2 output=output.tif
```

This calls **r.mapcalc.simple**: a=dem.tif — Raster layer A; b=dem.tif — Raster layer B; c=dem.tif — Raster layer C; d=dem.tif — Raster layer D; e=dem.tif — Raster layer E; f=dem.tif — Raster layer F; expression=A*2 — Formula; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.mask.rast — r.mask.rast

group: Raster (r.*)

r.mask.rast - Creates a MASK for limiting raster operation.

Parameter	Type	Required	Default	Description
raster	raster	yes	—	Name of raster map to use as mask
input	raster	yes	—	Name of raster map to which apply the mask
maskcats	string	no	*	Raster values to use for mask. Format: 1 2 3 thru 7 *
-i *(adv)*	boolean	no	False	Create inverse mask
output	rasterDestination	yes	—	Masked
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.mask.rast raster=dem.tif input=dem.tif maskcats=* output=output.tif
```

This calls **r.mask.rast**: raster=dem.tif — Name of raster map to use as mask; input=dem.tif — Name of raster map to which apply the mask; maskcats=* — Raster values to use for mask. Format: 1 2 3 thru 7 *; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.mask.vect — r.mask.vect

group: Raster (r.*)

r.mask.vect - Creates a MASK for limiting raster operation with a vector layer.

Parameter	Type	Required	Default	Description
vector	source	yes	—	Name of vector map to use as mask
input	raster	yes	—	Name of raster map to which apply the mask
cats *(adv)*	string	no	—	Category values. Example: 1,3,7-9,13

Parameter	Type	Required	Default	Description
`where *(adv)*`	string	no	—	WHERE conditions of SQL statement without 'where' keyword
`-i *(adv)*`	boolean	no	False	Create inverse mask
`output`	rasterDestination	yes	—	Masked
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.mask.vect vector=input.gpkg input=dem.tif output=output.tif
```

This calls **r.mask.vect**: `vector=input.gpkg` — Name of vector map to use as mask; `input=dem.tif` — Name of raster map to which apply the mask; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.mfilter` — r.mfilter

group: Raster (r.*)

Performs raster map matrix filter.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input layer
`filter`	file	yes	—	Filter file
`repeat`	number	no	1.0	Number of times to repeat the filter
`-z`	boolean	no	False	Apply filter only to zero data values
`output`	rasterDestination	yes	—	Filtered
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.mfilter input=dem.tif filter=input.dat repeat=1 output=output.tif
```

This calls **r.mfilter**: `input=dem.tif` — Input layer; `filter=input.dat` — Filter file; `repeat=1` — Number of times to repeat the filter; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.mode` — r.mode

group: Raster (r.*)

Finds the mode of values in a cover layer within areas assigned the same category value in a user-specified base layer.

Parameter	Type	Required	Default	Description
base	raster	yes	—	Base layer to be reclassified
cover	raster	yes	—	Categories layer
output	rasterDestination	yes	—	Mode
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.mode base=dem.tif cover=dem.tif output=output.tif
```

This calls **r.mode**: base=dem.tif — Base layer to be reclassified; cover=dem.tif — Categories layer; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.neighbors — r.neighbors

group: Raster (r.*)

Makes each cell category value a function of the category values assigned to the cells around it

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
selection	raster	no	—	Raster layer to select the cells which should be processed
method	enum	no	0	Neighborhood operation — options: 0=average, 1=median, 2=mode, 3=minimum, 4=maximum, 5=range, 6=stddev, 7=sum, 8=count, 9=variance, 10=diversity, 11=interspersion, 12=quart1, 13=quart3, 14=perc90, 15=quantile
size	number	no	3.0	Neighborhood size (must be odd)
gauss	number	no	—	Sigma (in cells) for Gaussian filter
quantile	string	no	—	Quantile to calculate for method=quantile
-c	boolean	no	False	Use circular neighborhood
-a *(adv)*	boolean	no	False	Do not align output with the input
weight *(adv)*	file	no	—	File containing weights
output	rasterDestination	yes	—	Neighbors
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.neighbors input=dem.tif selection=dem.tif method=0 size=3 gauss=10
↪ quantile=value output=output.tif
```

This calls **r.neighbors**: input=dem.tif — Input raster layer; selection=dem.tif — Raster layer to select the cells which should be processed; method=0 selects *average*; size=3 — Neighborhood size (must be odd); gauss=10 — Sigma (in cells) for Gaussian filter; quantile=value — Quantile

to calculate for method=quantile; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.null` — r.null

group: Raster (r.*)

Manages NULL-values of given raster map.

Parameter	Type	Required	Default	Description
`map`	raster	yes	—	Name of raster map for which to edit null values
`setnull`	string	no	—	List of cell values to be set to NULL
`null`	number	no	—	The value to replace the null value by
`-f *(adv)*`	boolean	no	False	Only do the work if the map is floating-point
`-i *(adv)*`	boolean	no	False	Only do the work if the map is integer
`-n *(adv)*`	boolean	no	False	Only do the work if the map doesn't have a NULL-value bitmap file
`-c *(adv)*`	boolean	no	False	Create NULL-value bitmap file validating all data cells
`-r *(adv)*`	boolean	no	False	Remove NULL-value bitmap file
`output`	rasterDestination	yes	—	NullRaster
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.null map=dem.tif setnull=value null=10 output=output.tif
```

This calls **r.null**: `map=dem.tif` — Name of raster map for which to edit null values; `setnull=value` — List of cell values to be set to NULL; `null=10` — The value to replace the null value by; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.out.ascii` — r.out.ascii

group: Raster (r.*)

Export a raster layer into a GRASS ASCII text file

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster
`precision`	number	no	—	Number of significant digits
`width`	number	no	—	Number of values printed before wrapping a line (only SURFER or MODFLOW format)
`null_value`	string	no	*	String to represent null cell (GRASS grid only)
`-s *(adv)*`	boolean	no	False	Write SURFER (Golden Software) ASCII grid
`-m *(adv)*`	boolean	no	False	Write MODFLOW (USGS) ASCII array
`-i *(adv)*`	boolean	no	False	Force output of integer values
`output`	fileDestination	yes	—	GRASS Ascii
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output` (outputFile)

Example usage

```
run grass:r.out.ascii input=dem.tif precision=10 width=10 null_value=* output=output.txt
```

This calls **r.out.ascii**: `input=dem.tif` — Input raster; `precision=10` — Number of significant digits; `width=10` — Number of values printed before wrapping a line (only SURFER or MODFLOW format); `null_value=*` — String to represent null cell (GRASS grid only); `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.out.gridatb` — r.out.gridatb

group: Raster (r.*)

Exports GRASS raster map to GRIDATB.FOR map file (TOPMODEL)

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Name of input raster map
`output`	fileDestination	yes	—	GRIDATB
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output` (outputFile)

Example usage

```
run grass:r.out.gridatb input=dem.tif output=output.txt
```

This calls **r.out.gridatb**: `input=dem.tif` — Name of input raster map; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.out.mat` — r.out.mat

group: Raster (r.*)

Exports a GRASS raster to a binary MAT-File

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster
`output`	fileDestination	yes	—	MAT File
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output` (outputFile)

Example usage

```
run grass:r.out.mat input=dem.tif output=output.txt
```

This calls **r.out.mat**: `input=dem.tif` — Input raster; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.out.mpeg — r.out.mpeg

group: Raster (r.*)

Converts raster map series to MPEG movie

Parameter	Type	Required	Default	Description
view1	multilayer	yes	—	Name of input raster map(s) for view no.1
view2	multilayer	no	—	Name of input raster map(s) for view no.2
view3	multilayer	no	—	Name of input raster map(s) for view no.3
view4	multilayer	no	—	Name of input raster map(s) for view no.4
quality	number	no	3.0	Quality factor (1 = highest quality, lowest compression)
output	fileDestination	yes	—	MPEG file
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.out.mpeg view1="a.gpkg;b.gpkg" view2="a.gpkg;b.gpkg" view3="a.gpkg;b.gpkg"
 ↪  view4="a.gpkg;b.gpkg" quality=3 output=output.txt
```

This calls **r.out.mpeg**: view1="a.gpkg;b.gpkg" — Name of input raster map(s) for view no.1; view2="a.gpkg;b.gpkg" — Name of input raster map(s) for view no.2; view3="a.gpkg;b.gpkg" — Name of input raster map(s) for view no.3; view4="a.gpkg;b.gpkg" — Name of input raster map(s) for view no.4; quality=3 — Quality factor (1 = highest quality, lowest compression); output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.out.png — r.out.png

group: Raster (r.*)

Export a GRASS raster map as a non-georeferenced PNG image

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster
compression	number	no	6.0	Compression level of PNG file (0 = none, 1 = fastest, 9 = best)
-t *(adv)*	boolean	no	False	Make NULL cells transparent
-w *(adv)*	boolean	no	False	Output world file
output	fileDestination	yes	—	PNG File
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.out.png input=dem.tif compression=6 output=output.txt
```

This calls **r.out.png**: input=dem.tif — Input raster; compression=6 — Compression level of PNG file (0 = none, 1 = fastest, 9 = best); output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.out.pov — r.out.pov

group: Raster (r.*)

Converts a raster map layer into a height-field file for POV-Ray

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster
hftype	number	no	0.0	Height-field type (0=actual heights 1=normalized)
bias	number	no	—	Elevation bias
scale	number	no	—	Vertical scaling factor
output	fileDestination	yes	—	Name of output povray file (TGA height field file)
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.out.pov input=dem.tif hftype=0 bias=10 scale=10 output=output.txt
```

This calls **r.out.pov**: input=dem.tif — Input raster; hftype=0 — Height-field type (0=actual heights 1=normalized); bias=10 — Elevation bias; scale=10 — Vertical scaling factor; output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.out.ppm — r.out.ppm

group: Raster (r.*)

Converts a raster layer to a PPM image file at the pixel resolution of the currently defined region.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
-g	boolean	no	True	Output greyscale instead of color
output	fileDestination	yes	—	PPM
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.out.ppm input=dem.tif output=output.txt
```

This calls **r.out.ppm**: input=dem.tif — Input raster layer; output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.out.ppm3 — r.out.ppm3

group: Raster (r.*)

Converts 3 GRASS raster layers (R,G,B) to a PPM image file

Parameter	Type	Required	Default	Description
red	raster	yes	—	Name of raster map to be used for
green	raster	yes	—	Name of raster map to be used for
blue	raster	yes	—	Name of raster map to be used for
-c	boolean	no	False	Add comments to describe the region
output	fileDestination	yes	—	Name for new PPM file
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.out.ppm3 red=dem.tif green=dem.tif blue=dem.tif output=output.txt
```

This calls **r.out.ppm3**: red=dem.tif — Name of raster map to be used for; green=dem.tif — Name of raster map to be used for; blue=dem.tif — Name of raster map to be used for; output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.out.vrml — r.out.vrml

group: Raster (r.*)

Export a raster layer to the Virtual Reality Modeling Language (VRML)

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Elevation layer
color	raster	yes	—	Color layer
exaggeration	number	no	1.0	Vertical exaggeration
output	fileDestination	yes	—	VRML
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.out.vrml elevation=dem.tif color=dem.tif exaggeration=1 output=output.txt
```

This calls **r.out.vrml**: elevation=dem.tif — Elevation layer; color=dem.tif — Color layer; exaggeration=1 — Vertical exaggeration; output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.out.vtk — r.out.vtk

group: Raster (r.*)

Converts raster maps into the VTK-ASCII format

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Input raster
elevation	raster	no	—	Input elevation raster map
null	number	no	-99999.99	Value to represent no data cell
z	number	no	0.0	Constant elevation (if no elevation map is specified)

Parameter	Type	Required	Default	Description
rgbmaps	multilayer	no	—	Three (r,g,b) raster maps to create RGB values
vectormaps	multilayer	no	—	Three (x,y,z) raster maps to create vector values
zscale	number	no	1.0	Scale factor for elevation
precision	number	no	12.0	Number of significant digits
-p *(adv)*	boolean	no	False	Create VTK point data instead of VTK cell data
-s *(adv)*	boolean	no	False	Use structured grid for elevation (not recommended)
-t *(adv)*	boolean	no	False	Use polydata-trianglestrips for elevation grid creation
-v *(adv)*	boolean	no	False	Use polydata-vertices for elevation grid creation
-o *(adv)*	boolean	no	False	Scale factor affects the origin (if no elevation map is given)
-c *(adv)*	boolean	no	False	Correct the coordinates to match the VTK-OpenGL precision
output	fileDestination	yes	—	VTK File
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.out.vtk input="a.gpkg;b.gpkg" elevation=dem.tif null=-100000 z=0
 ↪  rgbmaps="a.gpkg;b.gpkg" vectormaps="a.gpkg;b.gpkg" zscale=1 output=output.txt
```

This calls **r.out.vtk**: input="a.gpkg;b.gpkg" — Input raster; elevation=dem.tif — Input elevation raster map; null=-100000 — Value to represent no data cell; z=0 — Constant elevation (if no elevation map is specified); rgbmaps="a.gpkg;b.gpkg" — Three (r,g,b) raster maps to create RGB values; vectormaps="a.gpkg;b.gpkg" — Three (x,y,z) raster maps to create vector values; zscale=1 — Scale factor for elevation; output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.out.xyz — r.out.xyz

group: Raster (r.*)

Exports a raster map to a text file as x,y,z values based on cell centers

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Input raster(s)
separator	string	no	pipe	Field separator
-i *(adv)*	boolean	no	False	Include no data values
output	fileDestination	yes	—	XYZ File
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.out.xyz input="a.gpkg;b.gpkg" separator=pipe output=output.txt
```

This calls **r.out.xyz**: input="a.gpkg;b.gpkg" — Input raster(s); separator=pipe — Field separator; output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.param.scale — r.param.scale

group: Raster (r.*)

Extracts terrain parameters from a DEM.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
slope_tolerance	number	no	1.0	Slope tolerance that defines a 'flat' surface (degrees)
curvature_tolerance	number	no	0.0001	Curvature tolerance that defines 'planar' surface
size	number	no	3.0	Size of processing window (odd number only, max: 499)
method	enum	no	0	Morphometric parameter in 'size' window to calculate — options: 0=elev, 1=slope, 2=aspect, 3=profc, 4=planc, 5=longc, 6=crosc, 7=minic, 8=maxic, 9=feature
exponent	number	no	0.0	Exponent for distance weighting (0.0-4.0)
zscale	number	no	1.0	Vertical scaling factor
-c	boolean	no	False	Constrain model through central window cell
output	rasterDestination	yes	—	Morphometric parameter
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.param.scale input=dem.tif slope_tolerance=1 curvature_tolerance=0.0001 size=3
 ↪  method=0 exponent=0 zscale=1 output=output.tif
```

This calls **r.param.scale**: input=dem.tif — Name of input raster map; slope_tolerance=1 — Slope tolerance that defines a 'flat' surface (degrees); curvature_tolerance=0.0001 — Curvature tolerance that defines 'planar' surface; size=3 — Size of processing window (odd number only, max: 499); method=0 selects *elev*; exponent=0 — Exponent for distance weighting (0.0-4.0); zscale=1 — Vertical scaling factor; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.patch — r.patch

group: Raster (r.*)

Creates a composite raster layer by using one (or more) layer(s) to fill in areas of ”no data” in another map layer.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Raster layers to be patched together
-z	boolean	no	False	Use zero (0) for transparency instead of NULL
output	rasterDestination	yes	—	Patched
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.patch input="a.gpkg;b.gpkg" output=output.tif
```

This calls **r.patch**: `input="a.gpkg;b.gpkg"` — Raster layers to be patched together; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.path` — r.path

group: Raster (r.*)

Traces paths from starting points following input directions.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Name of input direction
`format`	enum	no	0	Format of the input direction map — options: 0=auto, 1=degree, 2=45degree, 3=bitmask
`values`	raster	no	—	Name of input raster values to be used for output
`raster_path`	rasterDestination	yes	—	Name for output raster path map
`vector_path`	vectorDestination	yes	—	Name for output vector path map
`start_points`	source	yes	—	Vector layer containing starting point(s)
`-c`	boolean	no	False	Copy input cell values on output
`-a`	boolean	no	False	Accumulate input values along the path
`-n`	boolean	no	False	Count cell numbers along the path
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `raster_path` (outputRaster), `vector_path` (outputVector)

Example usage

```
run grass:r.path input=dem.tif start_points=input.gpkg format=0 values=dem.tif
↪  raster_path=raster_path.tif
```

This calls **r.path**: `input=dem.tif` — Name of input direction; `start_points=input.gpkg` — Vector layer containing starting point(s); `format=0` selects *auto*; `values=dem.tif` — Name of input raster values to be used for output; `raster_path=raster_path.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.path.coordinate.txt — r.path.coordinate.txt

group: Raster (r.*)

r.path.coordinate.txt - Traces paths from starting points following input directions.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input direction
format	enum	no	0	Format of the input direction map — options: 0=auto, 1=degree, 2=45degree, 3=bitmask
values	raster	no	—	Name of input raster values to be used for output
raster_path	rasterDestination	yes	—	Name for output raster path map
vector_path	vectorDestination	yes	—	Name for output vector path map
start_coordinates	point	yes	—	Map coordinate of starting point (E,N)
-c	boolean	no	False	Copy input cell values on output
-a	boolean	no	False	Accumulate input values along the path
-n	boolean	no	False	Count cell numbers along the path
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: raster_path (outputRaster), vector_path (outputVector)

Example usage

```
run grass:r.path.coordinate.txt input=dem.tif start_coordinates=100,100 format=0
↪  values=dem.tif raster_path=raster_path.tif
```

This calls **r.path.coordinate.txt**: input=dem.tif — Name of input direction; start_coordinates=100,100 — Map coordinate of starting point (E,N); format=0 selects *auto*; values=dem.tif — Name of input raster values to be used for output; raster_path=raster_path.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.plane — r.plane

group: Raster (r.*)

Creates raster plane layer given dip (inclination), aspect (azimuth) and one point.

Parameter	Type	Required	Default	Description
dip	number	no	0.0	Dip of plane
azimuth	number	no	0.0	Azimuth of the plane
easting	number	yes	—	Easting coordinate of a point on the plane
northing	number	yes	—	Northing coordinate of a point on the plane
elevation	number	yes	—	Elevation coordinate of a point on the plane
type	enum	no	1	Data type of resulting layer — options: 0=CELL, 1=FCELL, 2=DCELL
output	rasterDestination	yes	—	Plane

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.plane easting=10 northing=10 elevation=10 dip=0 azimuth=0 type=1
↪  output=output.tif
```

This calls **r.plane**: easting=10 — Easting coordinate of a point on the plane; northing=10 — Northing coordinate of a point on the plane; elevation=10 — Elevation coordinate of a point on the plane; dip=0 — Dip of plane; azimuth=0 — Azimuth of the plane; type=1 selects *FCELL*; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.profile — r.profile

group: Raster (r.*)

Outputs the raster layer values lying on user-defined line(s).

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
coordinates	string	no	0,0,1,1	Profile coordinate pairs
resolution	number	no	—	Resolution along profile
null_value	string	no	*	Character to represent no data cell
file	file	no	—	Name of input file containing coordinate pairs
-g	boolean	no	False	Output easting and northing in first two columns of four column output
-c	boolean	no	False	Output RRR:GGG:BBB color values for each profile point
output	fileDestination	yes	—	Profile
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.profile input=dem.tif coordinates=0,0,1,1 resolution=10 null_value=*
↪  file=input.dat output=output.txt
```

This calls **r.profile**: input=dem.tif — Input raster layer; coordinates=0,0,1,1 — Profile coordinate pairs; resolution=10 — Resolution along profile; null_value=* — Character to represent no data cell; file=input.dat — Name of input file containing coordinate pairs; output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.proj — r.proj

group: Raster (r.*)

Re-projects a raster layer to another coordinate reference system

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster to reproject
crs	crs	yes	—	New coordinate reference system
method	enum	no	0	Interpolation method to use — options: 0=nearest, 1=bilinear, 2=bicubic, 3=lanczos, 4=bilinear_f, 5=bicubic_f, 6=lanczos_f
memory	number	no	300.0	Maximum memory to be used (in MB)
resolution	number	no	—	Resolution of output raster map
-n *(adv)*	boolean	no	False	Do not perform region cropping optimization
output	rasterDestination	yes	—	Reprojected raster
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.proj input=dem.tif crs=EPSG:6346 method=0 memory=300 resolution=10
↪  output=output.tif
```

This calls **r.proj**: input=dem.tif — Input raster to reproject; crs=EPSG:6346 — New coordinate reference system; method=0 selects *nearest*; memory=300 — Maximum memory to be used (in MB); resolution=10 — Resolution of output raster map; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.quant — r.quant

group: Raster (r.*)

Produces the quantization file for a floating-point map.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Raster layer(s) to be quantized
basemap	raster	no	—	Base layer to take quant rules from
fprange	range	no	—	Floating point range: dmin,dmax
range	range	no	—	Integer range: min,max
-t	boolean	no	False	Truncate floating point data
-r	boolean	no	False	Round floating point data
output	folderDestination	yes	—	Quantized raster(s)
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: output (outputFolder)

Example usage

```
run grass:r.quant input="a.gpkg;b.gpkg" basemap=dem.tif output=output/
```

This calls **r.quant**: input="a.gpkg;b.gpkg" — Raster layer(s) to be quantized; basemap=dem.tif — Base layer to take quant rules from; output=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.quantile — r.quantile

group: Raster (r.*)

Compute quantiles using two passes.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
quantiles	number	no	4.0	Number of quantiles
percentiles	string	no	—	List of percentiles
bins	number	no	1000000.0	Number of bins to use
-r *(adv)*	boolean	no	False	Generate recode rules based on quantile-defined intervals
file	fileDestination	no	report.html	Quantiles
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: file (outputHtml)

Example usage

```
run grass:r.quantile input=dem.tif quantiles=4 percentiles=value bins=1000000
↪  file=file.txt
```

This calls **r.quantile**: input=dem.tif — Input raster layer; quantiles=4 — Number of quantiles; percentiles=value — List of percentiles; bins=1000000 — Number of bins to use; file=file.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.quantile.plain — r.quantile.plain

group: Raster (r.*)

r.quantile.plain - Compute quantiles using two passes and save them as plain text.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
quantiles	number	no	4.0	Number of quantiles
percentiles	string	no	—	List of percentiles
bins	number	no	1000000.0	Number of bins to use
-r *(adv)*	boolean	no	False	Generate recode rules based on quantile-defined intervals
file	fileDestination	no	report.txt	Quantiles
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: file (outputFile)

Example usage

```
run grass:r.quantile.plain input=dem.tif quantiles=4 percentiles=value bins=1000000
↪  file=file.txt
```

This calls **r.quantile.plain**: input=dem.tif — Input raster layer; quantiles=4 — Number of quantiles; percentiles=value — List of percentiles; bins=1000000 — Number of bins to use; file=file.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.random` — r.random

group: Raster (r.*)

Creates a raster layer and vector point map containing randomly located points.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster layer
`cover`	raster	yes	—	Input cover raster layer
`npoints`	number	yes	—	The number of points to allocate
`-z`	boolean	no	False	Generate points also for NULL category
`-d`	boolean	no	False	Generate vector points as 3D points
`-b`	boolean	no	False	Do not build topology
`raster`	rasterDestination	yes	—	Random raster
`vector`	vectorDestination	yes	—	Random vector
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `raster` (outputRaster), `vector` (outputVector)

Example usage

```
run grass:r.random input=dem.tif cover=dem.tif npoints=10 raster=raster.tif
```

This calls **r.random**: `input=dem.tif` — Input raster layer; `cover=dem.tif` — Input cover raster layer; `npoints=10` — The number of points to allocate; `raster=raster.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.random.cells` — r.random.cells

group: Raster (r.*)

Generates random cell values with spatial dependence.

Parameter	Type	Required	Default	Description
`distance`	number	no	0.0	Maximum distance of spatial correlation (value(s) >= 0.0)
`ncells`	number	no	—	Maximum number of cells to be created
`seed *(adv)*`	number	no	—	Random seed (SEED_MIN >= value >= SEED_MAX) (default [random])
`output`	rasterDestination	yes	—	Random
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.random.cells distance=0 ncells=10 output=output.tif
```

This calls **r.random.cells**: `distance=0` — Maximum distance of spatial correlation (value(s) >= 0.0); `ncells=10` — Maximum number of cells to be created; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.random.surface` — r.random.surface

group: Raster (r.*)

Generates random surface(s) with spatial dependence.

Parameter	Type	Required	Default	Description
`distance`	number	no	0.0	Maximum distance of spatial correlation
`exponent`	number	no	1.0	Distance decay exponent
`flat`	number	no	0.0	Distance filter remains flat before beginning exponent
`seed`	number	no	—	Random seed (SEED_MIN >= value >= SEED_MAX)
`high`	number	no	255.0	Maximum cell value of distribution
`-u`	boolean	no	False	Uniformly distributed cell values
`output`	rasterDestination	yes	—	Random_Surface
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.random.surface distance=0 exponent=1 flat=0 seed=10 high=255 output=output.tif
```

This calls **r.random.surface**: `distance=0` — Maximum distance of spatial correlation; `exponent=1` — Distance decay exponent; `flat=0` — Distance filter remains flat before beginning exponent; `seed=10` — Random seed (SEED_MIN >= value >= SEED_MAX); `high=255` — Maximum cell value of distribution; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.reclass` — r.reclass

group: Raster (r.*)

Creates a new map layer whose category values are based upon a reclassification of the categories in an existing raster map layer.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster layer
`rules`	file	no	—	File containing reclass rules
`txtrules`	string	no	—	Reclass rules text (if rule file not used)

Parameter	Type	Required	Default	Description
output	rasterDestination	yes	—	Reclassified
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.reclass input=dem.tif rules=input.dat txtrules=value output=output.tif
```

This calls **r.reclass**: input=dem.tif — Input raster layer; rules=input.dat — File containing reclass rules; txtrules=value — Reclass rules text (if rule file not used); output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.reclass.area — r.reclass.area

group: Raster (r.*)

Reclassifies a raster layer, greater or less than user specified area size (in hectares)

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
value	number	no	1.0	Value option that sets the area size limit [hectares]
mode	enum	no	0	Lesser or greater than specified value — options: 0=lesser, 1=greater
method	enum	no	0	Method used for reclassification — options: 0=reclass, 1=rmarea
-c *(adv)*	boolean	no	False	Input map is clumped
-d *(adv)*	boolean	no	False	Clumps including diagonal neighbors
output	rasterDestination	yes	—	Reclassified
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.reclass.area input=dem.tif value=1 mode=0 method=0 output=output.tif
```

This calls **r.reclass.area**: input=dem.tif — Input raster layer; value=1 — Value option that sets the area size limit [hectares]; mode=0 selects *lesser*; method=0 selects *reclass*; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.recode — r.recode

group: Raster (r.*)

Recodes categorical raster maps.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input layer
rules	file	no	NoneFalse	File containing recode rules
-d *(adv)*	boolean	no	False	Force output to 'double' raster map type (DCELL)
-a *(adv)*	boolean	no	False	Align the current region to the input raster map
output	rasterDestination	yes	—	Recoded
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.recode input=dem.tif rules=input.dat output=output.tif
```

This calls **r.recode**: input=dem.tif — Input layer; rules=input.dat — File containing recode rules; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.regression.line — r.regression.line

group: Raster (r.*)

Calculates linear regression from two raster layers : y = a + b*x.

Parameter	Type	Required	Default	Description
mapx	raster	yes	—	Layer for x coefficient
mapy	raster	yes	—	Layer for y coefficient
html	fileDestination	no	report.html	Regression coefficients
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: html (outputHtml)

Example usage

```
run grass:r.regression.line mapx=dem.tif mapy=dem.tif html=html.txt
```

This calls **r.regression.line**: mapx=dem.tif — Layer for x coefficient; mapy=dem.tif — Layer for y coefficient; html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.regression.multi — r.regression.multi

group: Raster (r.*)

Calculates multiple linear regression from raster maps.

Parameter	Type	Required	Default	Description
mapx	multilayer	yes	—	Map(s) for x coefficient
mapy	raster	yes	—	Map for y coefficient
residuals	rasterDestination	yes	—	Residual Map
estimates	rasterDestination	yes	—	Estimates Map

Parameter	Type	Required	Default	Description
html	fileDestination	no	report.html	Regression coefficients
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: residuals (outputRaster), estimates (outputRaster), html (outputHtml)

Example usage

```
run grass:r.regression.multi mapx="a.gpkg;b.gpkg" mapy=dem.tif residuals=residuals.tif
```

This calls **r.regression.multi**: mapx="a.gpkg;b.gpkg" — Map(s) for x coefficient; mapy=dem.tif — Map for y coefficient; residuals=residuals.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.relief — r.relief

group: Raster (r.*)

Creates shaded relief from an elevation layer (DEM).

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input elevation layer
altitude	number	no	30.0	Altitude of the sun in degrees above the horizon
azimuth	number	no	270.0	Azimuth of the sun in degrees to the east of north
zscale	number	no	1.0	Factor for exaggerating relief
scale	number	no	1.0	Scale factor for converting horizontal units to elevation units
units	enum	no	0	Elevation units (overrides scale factor) — options: 0=intl, 1=survey
output	rasterDestination	yes	—	Output shaded relief layer
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.relief input=dem.tif altitude=30 azimuth=270 zscale=1 scale=1 units=0
 ↪  output=output.tif
```

This calls **r.relief**: input=dem.tif — Input elevation layer; altitude=30 — Altitude of the sun in degrees above the horizon; azimuth=270 — Azimuth of the sun in degrees to the east of north; zscale=1 — Factor for exaggerating relief; scale=1 — Scale factor for converting horizontal units to elevation units; units=0 selects *intl*; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.relief.scaling — r.relief.scaling

group: Raster (r.*)

r.relief.scaling - Creates shaded relief from an elevation layer (DEM).

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input elevation layer
`altitude`	number	no	30.0	Altitude of the sun in degrees above the horizon
`azimuth`	number	no	270.0	Azimuth of the sun in degrees to the east of north
`zscale`	number	no	1.0	Factor for exaggerating relief
`scale`	number	no	1.0	Scale factor for converting horizontal units to elevation units
`units`	enum	yes	—	Elevation units (overrides scale factor) — options: 0=intl, 1=survey
`output`	rasterDestination	yes	—	Output shaded relief layer
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.relief.scaling input=dem.tif units=0 altitude=30 azimuth=270 zscale=1 scale=1
↪  output=output.tif
```

This calls **r.relief.scaling**: `input=dem.tif` — Input elevation layer; `units=0` selects *intl*; `altitude=30` — Altitude of the sun in degrees above the horizon; `azimuth=270` — Azimuth of the sun in degrees to the east of north; `zscale=1` — Factor for exaggerating relief; `scale=1` — Scale factor for converting horizontal units to elevation units; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.report` — r.report

group: Raster (r.*)

Reports statistics for raster layers.

Parameter	Type	Required	Default	Description
`map`	multilayer	yes	—	Raster layer(s) to report on
`units`	enum	no	1	Units — options: 0=mi, 1=me, 2=k, 3=a, 4=h, 5=c, 6=p
`null_value`	string	no	*	Character representing no data cell value
`page_length`	number	no	0.0	Page length
`page_width`	number	no	79.0	Page width
`nsteps`	number	no	255.0	Number of fp subranges to collect stats from
`sort`	enum	no	0	Sort output statistics by cell counts — options: 0=asc, 1=desc
`-h`	boolean	no	False	Suppress page headers
`-f`	boolean	no	False	Use formfeeds between pages
`-e`	boolean	no	False	Scientific format
`-n`	boolean	no	False	Do not report no data cells
`-a`	boolean	no	False	Do not report cells where all maps have no data
`-c`	boolean	no	False	Report for cats floating-point ranges (floating-point maps only)
`-i`	boolean	no	False	Read floating-point map as integer (use map's quant rules)

Parameter	Type	Required	Default	Description
output	fileDestination	no	—	Name for output file to hold the report
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.report map="a.gpkg;b.gpkg" units=1 null_value=* page_length=0 page_width=79
↪  nsteps=255 sort=0 output=output.txt
```

This calls **r.report**: map="a.gpkg;b.gpkg" — Raster layer(s) to report on; units=1 selects *me*; null_value=* — Character representing no data cell value; page_length=0 — Page length; page_width=79 — Page width; nsteps=255 — Number of fp subranges to collect stats from; sort=0 selects *asc*; output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.resamp.bspline — r.resamp.bspline

group: Raster (r.*)

Performs bilinear or bicubic spline interpolation with Tykhonov regularization.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
mask	raster	no	—	Name of raster map to use for masking. Only cells that are not NULL and not zero are interpolated
method	enum	no	1	Sampling interpolation method — options: 0=bilinear, 1=bicubic
ew_step	number	no	—	Length (float) of each spline step in the east-west direction
ns_step	number	no	—	Length (float) of each spline step in the north-south direction
lambda	number	no	0.01	Tykhonov regularization parameter (affects smoothing)
memory	number	no	300.0	Maximum memory to be used (in MB). Cache size for raster rows
-n *(adv)*	boolean	no	False	Only interpolate null cells in input raster map
-c *(adv)*	boolean	no	False	Find the best Tykhonov regularizing parameter using a ”leave-one-out” cross validation method
output	rasterDestination	yes	—	Resampled BSpline
grid	vectorDestination	yes	—	Interpolation Grid
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputRaster), grid (outputVector)

Example usage

```
run grass:r.resamp.bspline input=dem.tif mask=dem.tif method=1 ew_step=10 ns_step=10
↪  lambda=0.01 memory=300 output=output.tif
```

This calls **r.resamp.bspline**: `input=dem.tif` — Input raster layer; `mask=dem.tif` — Name of raster map to use for masking. Only cells that are not NULL and not...; `method=1` selects *bicubic*; `ew_step=10` — Length (float) of each spline step in the east-west direction; `ns_step=10` — Length (float) of each spline step in the north-south direction; `lambda=0.01` — Tykhonov regularization parameter (affects smoothing); `memory=300` — Maximum memory to be used (in MB). Cache size for raster rows; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.resamp.filter` — r.resamp.filter

group: Raster (r.*)

Resamples raster map layers using an analytic kernel.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster layer
`filter`	enum	no	[0]	Filter kernel(s) — options: 0=box, 1=bartlett, 2=gauss, 3=normal, 4=hermite, 5=sinc, 6=lanczos1, 7=lanczos2, 8=lanczos3, 9=hann, 10=hamming, 11=blackman
`radius`	string	no	—	Filter radius for each filter (comma separated list of float if multiple)
`x_radius`	string	no	—	Filter radius (horizontal) for each filter (comma separated list of float if multiple)
`y_radius`	string	no	—	Filter radius (vertical) for each filter (comma separated list of float if multiple)
`-n *(adv)*`	boolean	no	False	Propagate NULLs
`output`	rasterDestination	yes	—	Resampled Filter
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.resamp.filter input=dem.tif filter=0 radius=value x_radius=value
↪  y_radius=value output=output.tif
```

This calls **r.resamp.filter**: `input=dem.tif` — Input raster layer; `filter=0` selects *box*; `radius=value` — Filter radius for each filter (comma separated list of float if multiple); `x_radius=value` — Filter radius (horizontal) for each filter (comma separated list of float if...; `y_radius=value` — Filter radius (vertical) for each filter (comma separated list of float if...; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.resamp.interp` — r.resamp.interp

group: Raster (r.*)

Resamples raster map to a finer grid using interpolation.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
method	enum	no	1	Sampling interpolation method — options: 0=nearest, 1=bilinear, 2=bicubic, 3=lanczos
output	rasterDestination	yes	—	Resampled interpolated
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.resamp.interp input=dem.tif method=1 output=output.tif
```

This calls **r.resamp.interp**: input=dem.tif — Input raster layer; method=1 selects *bilinear*; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.resamp.rst — r.resamp.rst

group: Raster (r.*)

Reinterpolates using regularized spline with tension and smoothing.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Raster layer
smooth	raster	no	—	Input raster map containing smoothing
maskmap	raster	no	—	Input raster map to be used as mask
ew_res	number	yes	—	Desired east-west resolution
ns_res	number	yes	—	Desired north-south resolution
overlap	number	no	3.0	Rows/columns overlap for segmentation
zscale	number	no	1.0	Multiplier for z-values
tension	number	no	40.0	Spline tension value
theta	number	no	—	Anisotropy angle (in degrees counterclockwise from East)
scalex	number	no	—	Anisotropy scaling factor
-t	boolean	no	False	Use dnorm independent tension
-d	boolean	no	False	Output partial derivatives instead of topographic parameters
elevation	rasterDestination	yes	—	Resampled RST
slope	rasterDestination	yes	—	Slope raster
aspect	rasterDestination	yes	—	Aspect raster
pcurvature	rasterDestination	yes	—	Profile curvature raster
tcurvature	rasterDestination	yes	—	Tangential curvature raster
mcurvature	rasterDestination	yes	—	Mean curvature raster
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: elevation (outputRaster), slope (outputRaster), aspect (outputRaster), pcurvature (outputRaster), tcurvature (outputRaster), mcurvature (outputRaster)

Example usage

```
run grass:r.resamp.rst input=dem.tif ew_res=10 ns_res=10 smooth=dem.tif maskmap=dem.tif
↪  overlap=3 zscale=1 elevation=elevation.tif
```

This calls **r.resamp.rst**: `input=dem.tif` — Raster layer; `ew_res=10` — Desired east-west resolution; `ns_res=10` — Desired north-south resolution; `smooth=dem.tif` — Input raster map containing smoothing; `maskmap=dem.tif` — Input raster map to be used as mask; `overlap=3` — Rows/columns overlap for segmentation; `zscale=1` — Multiplier for z-values; `elevation=elevation.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.resamp.stats` — r.resamp.stats

group: Raster (r.*)

Resamples raster layers to a coarser grid using aggregation.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster layer
`method`	enum	no	0	Aggregation method — options: 0=average, 1=median, 2=mode, 3=minimum, 4=maximum, 5=quart1, 6=quart3, 7=perc90, 8=sum, 9=variance, 10=stddev, 11=quantile
`quantile`	number	no	0.5	Quantile to calculate for method=quantile
`-n`	boolean	no	False	Propagate NULLs
`-w`	boolean	no	False	Weight according to area (slower)
`output`	rasterDestination	yes	—	Resampled aggregated
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.resamp.stats input=dem.tif method=0 quantile=0.5 output=output.tif
```

This calls **r.resamp.stats**: `input=dem.tif` — Input raster layer; `method=0` selects *average*; `quantile=0.5` — Quantile to calculate for method=quantile; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.resample` — r.resample

group: Raster (r.*)

GRASS raster map layer data resampling capability using nearest neighbors.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster layer
`output`	rasterDestination	yes	—	Resampled NN
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.resample input=dem.tif output=output.tif
```

This calls **r.resample**: `input=dem.tif` — Input raster layer; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.rescale` — r.rescale

group: Raster (r.*)

Rescales the range of category values in a raster layer.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster layer
`from`	range	no	—	The input data range to be rescaled
`to`	range	yes	—	The output data range
`output`	rasterDestination	yes	—	Rescaled
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.rescale input=dem.tif to=… output=output.tif
```

This calls **r.rescale**: `input=dem.tif` — Input raster layer; `to=…` — The output data range; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.rescale.eq` — r.rescale.eq

group: Raster (r.*)

Rescales histogram equalized the range of category values in a raster layer.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Input raster layer
`from`	range	no	—	The input data range to be rescaled
`to`	range	yes	—	The output data range
`output`	rasterDestination	yes	—	Rescaled equalized
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.rescale.eq input=dem.tif to=… output=output.tif
```

This calls **r.rescale.eq**: `input=dem.tif` — Input raster layer; `to=…` — The output data range; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.rgb — r.rgb

group: Raster (r.*)

Splits a raster map into red, green and blue maps.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
red	rasterDestination	yes	—	Red
green	rasterDestination	yes	—	Green
blue	rasterDestination	yes	—	Blue
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: red (outputRaster), green (outputRaster), blue (outputRaster)

Example usage

```
run grass:r.rgb input=dem.tif red=red.tif
```

This calls **r.rgb**: `input=dem.tif` — Name of input raster map; `red=red.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.ros — r.ros

group: Raster (r.*)

Generates rate of spread raster maps.

Parameter	Type	Required	Default	Description
model	raster	yes	—	Raster map containing fuel models
moisture_1h	raster	no	—	Raster map containing the 1-hour fuel moisture (%)
moisture_10h	raster	no	—	Raster map containing the 10-hour fuel moisture (%)
moisture_100h	raster	no	—	Raster map containing the 100-hour fuel moisture (%)
moisture_live	raster	yes	—	Raster map containing live fuel moisture (%)
velocity	raster	no	—	Raster map containing midflame wind velocities (ft/min)
direction	raster	no	—	Name of raster map containing wind directions (degree)
slope	raster	no	—	Name of raster map containing slope (degree)
aspect	raster	no	—	Raster map containing aspect (degree, CCW from E)
elevation	raster	no	—	Raster map containing elevation (m, required for spotting)
base_ros	rasterDestination	yes	—	Base ROS

Parameter	Type	Required	Default	Description
max_ros	rasterDestination	yes	—	Max ROS
direction_ros	rasterDestination	yes	—	Direction ROS
spotting_distance	rasterDestination	yes	—	Spotting Distance
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: base_ros (outputRaster), max_ros (outputRaster), direction_ros (outputRaster), spotting_distance (outputRaster)

Example usage

```
run grass:r.ros model=dem.tif moisture_live=dem.tif moisture_1h=dem.tif
 ↪ moisture_10h=dem.tif moisture_100h=dem.tif velocity=dem.tif direction=dem.tif
 ↪ base_ros=base_ros.tif
```

This calls **r.ros**: model=dem.tif — Raster map containing fuel models; moisture_live=dem.tif — Raster map containing live fuel moisture (%); moisture_1h=dem.tif — Raster map containing the 1-hour fuel moisture (%); moisture_10h=dem.tif — Raster map containing the 10-hour fuel moisture (%); moisture_100h=dem.tif — Raster map containing the 100-hour fuel moisture (%); velocity=dem.tif — Raster map containing midflame wind velocities (ft/min); direction=dem.tif — Name of raster map containing wind directions (degree); base_ros=base_ros.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.series — r.series

group: Raster (r.*)

Makes each output cell value a function of the values assigned to the corresponding cells in the input raster layers. Input rasters layers/bands must be separated in different data sources.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Input raster layer(s)
-n	boolean	no	False	Propagate NULLs
method	enum	no	[0]	Aggregate operation — options: 0=average, 1=count, 2=median, 3=mode, 4=minimum, 5=min_raster, 6=maximum, 7=max_raster, 8=stddev, 9=range, 10=sum, 11=variance, 12=diversity, 13=slope, 14=offset, 15=detcoeff, 16=quart1, 17=quart3, 18=perc90, 19=skewness, 20=kurtosis, 21=quantile, 22=tvalue
quantile	string	no	—	Quantile to calculate for method=quantile
weights	string	no	—	Weighting factor for each input map, default value is 1.0
range *(adv)*	range	no	—	Ignore values outside this range (lo,hi)
output	rasterDestination	yes	—	Aggregated
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.series input="a.gpkg;b.gpkg" method=0 quantile=value weights=value
 ↪  output=output.tif
```

This calls **r.series**: `input="a.gpkg;b.gpkg"` — Input raster layer(s); `method=0` selects *average*; `quantile=value` — Quantile to calculate for method=quantile; `weights=value` — Weighting factor for each input map, default value is 1.0; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.series.accumulate` — r.series.accumulate

group: Raster (r.*)

Makes each output cell value an accumulation function of the values assigned to the corresponding cells in the input raster map layers.

Parameter	Type	Required	Default	Description
`input`	multilayer	yes	—	Input raster layer(s)
`lower`	raster	no	—	Raster map specifying the lower accumulation limit
`upper`	raster	no	—	Raster map specifying the upper accumulation limit
`method`	enum	no	0	This method will be applied to compute the accumulative values from the input maps — options: 0=gdd, 1=bedd, 2=huglin, 3=mean
`scale`	number	no	1.0	Scale factor for input
`shift`	number	no	0.0	Shift factor for input
`range`	range	no	—	Ignore values outside this range (min,max)
`limits`	range	no	10,30	Lower and upper accumulation limits (lower,upper)
`-n`	boolean	no	False	Propagate NULLs
`-f *(adv)*`	boolean	no	False	Create a FCELL map (floating point single precision) as output
`output`	rasterDestination	yes	—	Accumulated
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.series.accumulate input="a.gpkg;b.gpkg" lower=dem.tif upper=dem.tif method=0
 ↪  scale=1 shift=0 output=output.tif
```

This calls **r.series.accumulate**: `input="a.gpkg;b.gpkg"` — Input raster layer(s); `lower=dem.tif` — Raster map specifying the lower accumulation limit; `upper=dem.tif` — Raster map specifying the upper accumulation limit; `method=0` selects *gdd*; `scale=1` — Scale factor for input; `shift=0` — Shift factor for input; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.series.interp` — r.series.interp

group: Raster (r.*)

Interpolates raster maps located (temporal or spatial) in between input raster maps at specific sampling positions.

Parameter	Type	Required	Default	Description
`input`	multilayer	yes	—	Input raster layer(s)
`datapos`	string	no	—	Data point position for each input map
`infile`	file	no	—	Input file with one input raster map name and data point position per line, field separator between name and sample point is 'pipe'
`output`	string	no	—	Name for output raster map (comma separated list if multiple)
`samplingpos`	string	no	—	Sampling point position for each output map (comma separated list)
`outfile`	file	no	—	Input file with one output raster map name and sample point position per line, field separator between name and sample point is 'pipe'
`method`	enum	no	0	Interpolation method, currently only linear interpolation is supported — options: 0=linear
`output_dir`	folderDestination	yes	—	Interpolated rasters
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output_dir` (outputFolder)

Example usage

```
run grass:r.series.interp input="a.gpkg;b.gpkg" datapos=value infile=input.dat
 ↪  output=value samplingpos=value outfile=input.dat method=0 output_dir=output_dir/
```

This calls **r.series.interp**: `input="a.gpkg;b.gpkg"` — Input raster layer(s); `datapos=value` — Data point position for each input map; `infile=input.dat` — Input file with one input raster map name and data point position per line,...; `output=value` — Name for output raster map (comma separated list if multiple); `samplingpos=value` — Sampling point position for each output map (comma separated list); `outfile=input.dat` — Input file with one output raster map name and sample point position per line,...; `method=0` selects *linear*; `output_dir=output_dir/` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.shade` — r.shade

group: Raster (r.*)

Drapes a color raster over an shaded relief or aspect map.

Parameter	Type	Required	Default	Description
`shade`	raster	yes	—	Name of shaded relief or aspect raster map
`color`	raster	yes	—	Name of raster to drape over relief raster map
`brighten`	number	no	0.0	Percent to brighten
`bgcolor`	string	no	—	Color to use instead of NULL values. Either a standard color name, R:G:B triplet, or ”none”
`-c *(adv)*`	boolean	no	False	Use colors from color tables for NULL values
`output`	rasterDestination	yes	—	Shaded
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)

Parameter	Type	Required	Default	Description
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.shade shade=dem.tif color=dem.tif brighten=0 bgcolor=value output=output.tif
```

This calls **r.shade**: shade=dem.tif — Name of shaded relief or aspect raster map; color=dem.tif — Name of raster to drape over relief raster map; brighten=0 — Percent to brighten; bgcolor=value — Color to use instead of NULL values. Either a standard color name, R:G:B...; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.sim.sediment — r.sim.sediment

group: Raster (r.*)

Sediment transport and erosion/deposition simulation using path sampling method (SIMWE).

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Name of the elevation raster map [m]
water_depth	raster	yes	—	Name of the water depth raster map [m]
dx	raster	yes	—	Name of the x-derivatives raster map [m/m]
dy	raster	yes	—	Name of the y-derivatives raster map [m/m]
detachment_coeff	raster	yes	—	Name of the detachment capacity coefficient raster map [s/m]
transport_coeff	raster	yes	—	Name of the transport capacity coefficient raster map [s]
shear_stress	raster	yes	—	Name of the critical shear stress raster map [Pa]
man	raster	no	—	Name of the Mannings n raster map
man_value	number	no	0.1	Name of the Mannings n value
observation	source	no	—	Sampling locations vector points
nwalkers	number	no	—	Number of walkers
niterations	number	no	10.0	Time used for iterations [minutes]
output_step	number	no	2.0	Time interval for creating output maps [minutes]
diffusion_coeff	number	no	0.8	Water diffusion constant
transport_capacity	rasterDestination	yes	—	Transport capacity [kg/ms]
tlimit_erosion_deposition	rasterDestination	yes	—	Transport limited erosion-deposition [kg/m2s]
sediment_concentration	rasterDestination	yes	—	Sediment concentration [particle/m3]
sediment_flux	rasterDestination	yes	—	Sediment flux [kg/ms]
erosion_deposition	rasterDestination	yes	—	Erosion-deposition [kg/m2s]
walkers_output	vectorDestination	no	—	Name of the output walkers vector points layer
logfile	fileDestination	no	—	Name for sampling points output text file.
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Parameter	Type	Required	Default	Description
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: transport_capacity (outputRaster), tlimit_erosion_deposition (outputRaster), sediment_concentration (outputRaster), sediment_flux (outputRaster), erosion_deposition (outputRaster), walkers_output (outputVector), logfile (outputFile)

Example usage

```
run grass:r.sim.sediment elevation=dem.tif water_depth=dem.tif dx=dem.tif dy=dem.tif
 ↪ detachment_coeff=dem.tif transport_coeff=dem.tif shear_stress=dem.tif
 ↪ transport_capacity=transport_capacity.tif
```

This calls **r.sim.sediment**: elevation=dem.tif — Name of the elevation raster map [m]; water_depth=dem.tif — Name of the water depth raster map [m]; dx=dem.tif — Name of the x-derivatives raster map [m/m]; dy=dem.tif — Name of the y-derivatives raster map [m/m]; detachment_coeff=dem.tif — Name of the detachment capacity coefficient raster map [s/m]; transport_coeff=dem.tif — Name of the transport capacity coefficient raster map [s]; shear_stress=dem.tif — Name of the critical shear stress raster map [Pa]; transport_capacity=transport_capacity.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.sim.water — r.sim.water

group: Raster (r.*)

Overland flow hydrologic simulation using path sampling method (SIMWE).

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Name of the elevation raster map [m]
dx	raster	yes	—	Name of the x-derivatives raster map [m/m]
dy	raster	yes	—	Name of the y-derivatives raster map [m/m]
rain	raster	no	—	Name of the rainfall excess rate (rain-infilt) raster map [mm/hr]
rain_value	number	no	50.0	Rainfall excess rate unique value [mm/hr]
infil	raster	no	—	Name of the runoff infiltration rate raster map [mm/hr]
infil_value	number	no	0.0	Runoff infiltration rate unique value [mm/hr]
man	raster	no	—	Name of the Mannings n raster map
man_value	number	no	0.1	Manning's n unique value
flow_control	raster	no	—	Name of the flow controls raster map (permeability ratio 0-1)
observation	source	no	—	Sampling locations vector points
nwalkers	number	no	—	Number of walkers, default is twice the number of cells
niterations	number	no	10.0	Time used for iterations [minutes]
output_step	number	no	2.0	Time interval for creating output maps [minutes]
diffusion_coeff	number	no	0.8	Water diffusion constant
hmax	number	no	4.0	Threshold water depth [m]
halpha	number	no	4.0	Diffusion increase constant
hbeta	number	no	0.5	Weighting factor for water flow velocity vector
-t	boolean	no	False	Time-series output

Parameter	Type	Required	Default	Description
depth	rasterDestination	yes	—	Water depth [m]
discharge	rasterDestination	yes	—	Water discharge [m3/s]
error	rasterDestination	yes	—	Simulation error [m]
walkers_output	vectorDestination	no	—	Name of the output walkers vector points layer
logfile	fileDestination	no	—	Name for sampling points output text file.
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: depth (outputRaster), discharge (outputRaster), error (outputRaster), walkers_output (outputVector), logfile (outputFile)

Example usage

```
run grass:r.sim.water elevation=dem.tif dx=dem.tif dy=dem.tif rain=dem.tif rain_value=50
 ↪  infil=dem.tif infil_value=0 depth=depth.tif
```

This calls **r.sim.water**: elevation=dem.tif — Name of the elevation raster map [m]; dx=dem.tif — Name of the x-derivatives raster map [m/m]; dy=dem.tif — Name of the y-derivatives raster map [m/m]; rain=dem.tif — Name of the rainfall excess rate (rain-infilt) raster map [mm/hr]; rain_value=50 — Rainfall excess rate unique value [mm/hr]; infil=dem.tif — Name of the runoff infiltration rate raster map [mm/hr]; infil_value=0 — Runoff infiltration rate unique value [mm/hr]; depth=depth.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.slope.aspect — r.slope.aspect

group: Raster (r.*)

Generates raster layers of slope, aspect, curvatures and partial derivatives from a elevation raster layer.

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Elevation
format	enum	no	0	Format for reporting the slope — options: 0=degrees, 1=percent
precision	enum	no	0	Type of output aspect and slope layer — options: 0=FCELL, 1=CELL, 2=DCELL
-a	boolean	no	True	Do not align the current region to the elevation layer
-e	boolean	no	False	Compute output at edges and near NULL values
-n	boolean	no	False	Create aspect as degrees clockwise from North (azimuth), with flat = -9999

Parameter	Type	Required	Default	Description
zscale	number	no	1.0	Multiplicative factor to convert elevation units to meters
min_slope	number	no	0.0	Minimum slope val. (in percent) for which aspect is computed
slope	rasterDestination	no	—	Slope
aspect	rasterDestination	no	—	Aspect
pcurvature	rasterDestination	no	—	Profile curvature
tcurvature	rasterDestination	no	—	Tangential curvature
dx	rasterDestination	no	—	First order partial derivative dx (E-W slope)
dy	rasterDestination	no	—	First order partial derivative dy (N-S slope)
dxx	rasterDestination	no	—	Second order partial derivative dxx
dyy	rasterDestination	no	—	Second order partial derivative dyy
dxy	rasterDestination	no	—	Second order partial derivative dxy
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: slope (outputRaster), aspect (outputRaster), pcurvature (outputRaster), tcurvature (outputRaster), dx (outputRaster), dy (outputRaster), dxx (outputRaster), dyy (outputRaster), dxy (outputRaster)

Example usage

```
run grass:r.slope.aspect elevation=dem.tif format=0 precision=0 zscale=1 min_slope=0
↪  slope=slope.tif
```

This calls **r.slope.aspect**: elevation=dem.tif — Elevation; format=0 selects *degrees*; precision=0 selects *FCELL*; zscale=1 — Multiplicative factor to convert elevation units to meters; min_slope=0 — Minimum slope val. (in percent) for which aspect is computed; slope=slope.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.solute.transport — r.solute.transport

group: Raster (r.*)

Numerical calculation program for transient, confined and unconfined solute transport in two dimensions

Parameter	Type	Required	Default	Description
c	raster	yes	—	The initial concentration in [kg/m^3]
phead	raster	yes	—	The piezometric head in [m]
hc_x	raster	yes	—	The x-part of the hydraulic conductivity tensor in [m/s]
hc_y	raster	yes	—	The y-part of the hydraulic conductivity tensor in [m/s]
status	raster	yes	—	The status for each cell, = 0 - inactive cell, 1 - active cell, 2 - dirichlet- and 3 - transfer boundary condition
diff_x	raster	yes	—	The x-part of the diffusion tensor in [m^2/s]
diff_y	raster	yes	—	The y-part of the diffusion tensor in [m^2/s]
q	raster	no	—	Groundwater sources and sinks in [m^3/s]
cin	raster	no	—	Concentration sources and sinks bounded to a water source or sink in [kg/s]
cs	raster	yes	—	Concentration of inner sources and inner sinks in [kg/s] (i.e. a chemical reaction)

Parameter	Type	Required	Default	Description
rd	raster	yes	—	Retardation factor [-]
nf	raster	yes	—	Effective porosity [-]
top	raster	yes	—	Top surface of the aquifer in [m]
bottom	raster	yes	—	Bottom surface of the aquifer in [m]
dtime	number	no	86400.0	Calculation time (in seconds)
maxit	number	no	10000.0	Maximum number of iteration used to solve the linear equation system
error	number	no	1e-06	Error break criteria for iterative solver
solver	enum	no	4	The type of solver which should solve the linear equation system — options: 0=gauss, 1=lu, 2=jacobi, 3=sor, 4=bicgstab
relax	number	no	1.0	The relaxation parameter used by the jacobi and sor solver for speedup or stabilizing
al	number	no	0.0	The longitudinal dispersivity length. [m]
at	number	no	0.0	The transversal dispersivity length. [m]
loops	number	no	1.0	Use this number of time loops if the CFL flag is off. The timestep will become dt/loops.
stab	enum	no	0	Set the flow stabilizing scheme (full or exponential upwinding). — options: 0=full, 1=exp
-c *(adv)*	boolean	no	False	Use the Courant-Friedrichs-Lewy criteria for time step calculation
-f *(adv)*	boolean	no	False	Use a full filled quadratic linear equation system, default is a sparse linear equation system.
output	rasterDestination	yes	—	Solute Transport
vx	rasterDestination	no	—	Calculate and store the groundwater filter velocity vector part in x direction [m/s]
vy	rasterDestination	no	—	Calculate and store the groundwater filter velocity vector part in y direction [m/s]
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster), vx (outputRaster), vy (outputRaster)

Example usage

```
run grass:r.solute.transport c=dem.tif phead=dem.tif hc_x=dem.tif hc_y=dem.tif
 ↪  status=dem.tif diff_x=dem.tif diff_y=dem.tif output=output.tif
```

This calls **r.solute.transport**: c=dem.tif — The initial concentration in [kg/m^3]; phead=dem.tif — The piezometric head in [m]; hc_x=dem.tif — The x-part of the hydraulic conductivity tensor in [m/s]; hc_y=dem.tif — The y-part of the hydraulic conductivity tensor in [m/s]; status=dem.tif — The status for each cell, = 0 - inactive cell, 1 - active cell, 2 - dirichlet-...; diff_x=dem.tif — The x-part of the diffusion tensor in [m^2/s]; diff_y=dem.tif — The y-part of the diffusion tensor in [m^2/s]; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.spread — r.spread

group: Raster (r.*)

Simulates elliptically anisotropic spread.

Parameter	Type	Required	Default	Description
base_ros	raster	yes	—	Raster map containing base ROS (cm/min)
max_ros	raster	yes	—	Raster map containing maximal ROS (cm/min)
direction_ros	raster	yes	—	Raster map containing directions of maximal ROS (degree)
start	raster	yes	—	Raster map containing starting sources
spotting_distance	raster	no	—	Raster map containing maximal spotting distance (m, required with -s)
wind_speed	raster	no	—	Raster map containing midflame wind speed (ft/min, required with -s)
fuel_moisture	raster	no	—	Raster map containing fine fuel moisture of the cell receiving a spotting firebrand (%, required with -s)
backdrop	raster	no	—	Name of raster map as a display backdrop
least_size	enum	no	0	Basic sampling window size needed to meet certain accuracy (3) — options: 0=3, 1=5, 2=7, 3=9, 4=11, 5=13, 6=15
comp_dens	number	no	0.5	Sampling density for additional computing (range: 0.0 - 1.0 (0.5))
init_time	number	no	0.0	Initial time for current simulation (0) (min)
lag	number	no	—	Simulating time duration LAG (fill the region) (min)
-s *(adv)*	boolean	no	False	Consider spotting effect (for wildfires)
-i *(adv)*	boolean	no	False	Use start raster map values in output spread time raster map
output	rasterDestination	yes	—	Spread Time
x_output	rasterDestination	yes	—	X Back Coordinates
y_output	rasterDestination	yes	—	Y Back Coordinates
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster), x_output (outputRaster), y_output (outputRaster)

Example usage

```
run grass:r.spread base_ros=dem.tif max_ros=dem.tif direction_ros=dem.tif start=dem.tif
 ↪  spotting_distance=dem.tif wind_speed=dem.tif fuel_moisture=dem.tif output=output.tif
```

This calls **r.spread**: base_ros=dem.tif — Raster map containing base ROS (cm/min); max_ros=dem.tif — Raster map containing maximal ROS (cm/min); direction_ros=dem.tif — Raster map containing directions of maximal ROS (degree); start=dem.tif — Raster map containing starting sources; spotting_distance=dem.tif — Raster map containing maximal spotting distance (m, required with -s); wind_speed=dem.tif — Raster map containing midflame wind speed (ft/min, required with -s); fuel_moisture=dem.tif — Raster map containing fine fuel moisture of the cell receiving a spotting…; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.spreadpath — r.spreadpath

group: Raster (r.*)

Recursively traces the least cost path backwards to cells from which the cumulative cost was determined.

Parameter	Type	Required	Default	Description
x_input	raster	yes	—	x_input
y_input	raster	yes	—	y_input
coordinates	point	no	0,0	coordinate
output	rasterDestination	yes	—	Backward least cost
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.spreadpath x_input=dem.tif y_input=dem.tif coordinates=100,100
↪  output=output.tif
```

This calls **r.spreadpath**: x_input=dem.tif — x_input; y_input=dem.tif — y_input; coordinates=100,100 — coordinate; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.statistics — r.statistics

group: Raster (r.*)

Calculates category or object oriented statistics.

Parameter	Type	Required	Default	Description
base	raster	yes	—	Base raster layer
cover	raster	yes	—	Cover raster layer
method	enum	no	0	method — options: 0=diversity, 1=average, 2=mode, 3=median, 4=avedev, 5=stddev, 6=variance, 7=skewness, 8=kurtosis, 9=min, 10=max, 11=sum
-c	boolean	no	False	Cover values extracted from the category labels of the cover map
routput	rasterDestination	yes	—	Statistics
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: routput (outputRaster)

Example usage

```
run grass:r.statistics base=dem.tif cover=dem.tif method=0 routput=routput.tif
```

This calls **r.statistics**: base=dem.tif — Base raster layer; cover=dem.tif — Cover raster layer; method=0 selects *diversity*; routput=routput.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.stats — r.stats

group: Raster (r.*)

Generates area statistics for raster layers.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Name of input raster map
separator	string	no	space	Output field separator
null_value	string	no	*	String representing no data cell value
nsteps	number	no	255.0	Number of floating-point subranges to collect stats from
sort	enum	no	0	Sort output statistics by cell counts — options: 0=asc, 1=desc
-1	boolean	no	True	One cell (range) per line
-A	boolean	no	False	Print averaged values instead of intervals
-a	boolean	no	False	Print area totals
-c	boolean	no	False	Print cell counts
-p	boolean	no	False	Print APPROXIMATE percents (total percent may not be 100%)
-l	boolean	no	False	Print category labels
-g	boolean	no	False	Print grid coordinates (east and north)
-x	boolean	no	False	Print x and y (column and row)
-r	boolean	no	False	Print raw indexes of fp ranges (fp maps only)
-n	boolean	no	False	Suppress reporting of any NULLs
-N	boolean	no	False	Suppress reporting of NULLs when all values are NULL
-C	boolean	no	False	Report for cats fp ranges (fp maps only)
-i	boolean	no	False	Read fp map as integer (use map's quant rules)
html	fileDestination	no	report.html	Statistics
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `html` (outputHtml)

Example usage

```
run grass:r.stats input="a.gpkg;b.gpkg" separator=space null_value=* nsteps=255 sort=0
↪  html=html.txt
```

This calls **r.stats**: `input="a.gpkg;b.gpkg"` — Name of input raster map; `separator=space` — Output field separator; `null_value=*` — String representing no data cell value; `nsteps=255` — Number of floating-point subranges to collect stats from; `sort=0` selects *asc*; `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.stats.quantile.out` — r.stats.quantile.out

group: Raster (r.*)

r.stats.quantile.out - Compute category quantiles using two passes and output statistics

Parameter	Type	Required	Default	Description
base	raster	yes	—	Name of base raster map
cover	raster	yes	—	Name of cover raster map
quantiles	number	no	—	Number of quantiles
percentiles	string	no	—	List of percentiles
bins	number	no	1000.0	Number of bins to use
-r *(adv)*	boolean	no	False	Create reclass map with statistics as category labels
file	fileDestination	yes	—	Statistics File
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `file` (outputFile)

Example usage

```
run grass:r.stats.quantile.out base=dem.tif cover=dem.tif quantiles=10 percentiles=value
↪  bins=1000 file=file.txt
```

This calls **r.stats.quantile.out**: `base=dem.tif` — Name of base raster map; `cover=dem.tif` — Name of cover raster map; `quantiles=10` — Number of quantiles; `percentiles=value` — List of percentiles; `bins=1000` — Number of bins to use; `file=file.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.stats.quantile.rast` — r.stats.quantile.rast

group: Raster (r.*)

r.stats.quantile.rast - Compute category quantiles using two passes and output rasters.

Parameter	Type	Required	Default	Description
`base`	raster	yes	—	Name of base raster map
`cover`	raster	yes	—	Name of cover raster map
`quantiles`	number	no	—	Number of quantiles
`percentiles`	string	no	—	List of percentiles
`bins`	number	no	1000.0	Number of bins to use
`-r *(adv)*`	boolean	no	False	Create reclass map with statistics as category labels
`output`	folderDestination	yes	—	Output Directory
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output` (outputFolder)

Example usage

```
run grass:r.stats.quantile.rast base=dem.tif cover=dem.tif quantiles=10 percentiles=value
↪  bins=1000 output=output/
```

This calls **r.stats.quantile.rast**: `base=dem.tif` — Name of base raster map; `cover=dem.tif` — Name of cover raster map; `quantiles=10` — Number of quantiles; `percentiles=value` — List of percentiles; `bins=1000` — Number of bins to use; `output=output/` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.stats.zonal` — r.stats.zonal

group: Raster (r.*)

Calculates category or object oriented statistics (accumulator-based statistics)

Parameter	Type	Required	Default	Description
`base`	raster	yes	—	Base raster
`cover`	raster	yes	—	Cover raster
`method`	enum	no	0	Method of object-based statistic — options: 0=count, 1=sum, 2=min, 3=max, 4=range, 5=average, 6=avedev, 7=variance, 8=stddev, 9=skewness, 10=kurtosis, 11=variance2, 12=stddev2, 13=skewness2, 14=kurtosis2

Parameter	Type	Required	Default	Description
`-c *(adv)*`	boolean	no	False	Cover values extracted from the category labels of the cover map
`-r *(adv)*`	boolean	no	False	Create reclass map with statistics as category labels
`output`	rasterDestination	yes	—	Resultant raster
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.stats.zonal base=dem.tif cover=dem.tif method=0 output=output.tif
```

This calls **r.stats.zonal**: `base=dem.tif` — Base raster; `cover=dem.tif` — Cover raster; `method=0` selects *count*; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.stream.extract` — r.stream.extract

group: Raster (r.*)

Stream network extraction

Parameter	Type	Required	Default	Description
`elevation`	raster	yes	—	Input map: elevation map
`accumulation`	raster	no	—	Input map: accumulation map
`depression`	raster	no	—	Input map: map with real depressions
`threshold`	number	no	1.0	Minimum flow accumulation for streams
`mexp`	number	no	0.0	Montgomery exponent for slope
`stream_length`	number	no	0.0	Delete stream segments shorter than cells
`d8cut`	number	no	—	Use SFD above this threshold
`memory *(adv)*`	number	no	300.0	Maximum memory to be used (in MB)
`stream_raster`	rasterDestination	no	—	Unique stream ids (rast)
`stream_vector`	vectorDestination	no	—	Unique stream ids (vect)
`direction`	rasterDestination	no	—	Flow direction
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `stream_raster` (outputRaster), `stream_vector` (outputVector), `direction` (outputRaster)

Example usage

```
run grass:r.stream.extract elevation=dem.tif accumulation=dem.tif depression=dem.tif
 ↪  threshold=1 mexp=0 stream_length=0 d8cut=10 stream_raster=stream_raster.tif
```

This calls **r.stream.extract**: `elevation=dem.tif` — Input map: elevation map; `accumulation=dem.tif` — Input map: accumulation map; `depression=dem.tif` — Input map: map with real depressions; `threshold=1` — Minimum flow accumulation for streams; `mexp=0` — Montgomery exponent for slope; `stream_length=0` — Delete stream segments shorter than cells; `d8cut=10` — Use SFD above this threshold; `stream_raster=stream_raster.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.sun.incidout` — r.sun.incidout

group: Raster (r.*)

r.sun.incidout - Solar irradiance and irradiation model (for the set local time).

Parameter	Type	Required	Default	Description
`elevation`	raster	yes	—	Elevation layer [meters]
`aspect`	raster	yes	—	Aspect layer [decimal degrees]
`aspect_value`	number	no	270.0	A single value of the orientation (aspect), 270 is south
`slope`	raster	yes	—	Name of the input slope raster map (terrain slope or solar panel inclination) [decimal degrees]
`slope_value`	number	no	0.0	A single value of inclination (slope)
`linke`	raster	no	—	Name of the Linke atmospheric turbidity coefficient input raster map
`albedo`	raster	no	—	Name of the ground albedo coefficient input raster map
`albedo_value`	number	no	0.2	A single value of the ground albedo coefficient
`lat`	raster	no	—	Name of input raster map containing latitudes [decimal degrees]
`long`	raster	no	—	Name of input raster map containing longitudes [decimal degrees]
`coeff_bh`	raster	no	—	Name of real-sky beam radiation coefficient input raster map
`coeff_dh`	raster	no	—	Name of real-sky diffuse radiation coefficient input raster map
`horizon_basemap`	raster	no	—	The horizon information input map basename
`horizon_step`	number	no	—	Angle step size for multidirectional horizon [degrees]
`day`	number	no	1.0	No. of day of the year (1-365)
`step *(adv)*`	number	no	0.5	Time step when computing all-day radiation sums [decimal hours]
`declination *(adv)*`	number	no	—	Declination value (overriding the internally computed value) [radians]
`distance_step *(adv)*`	number	no	1.0	Sampling distance step coefficient (0.5-1.5)
`npartitions *(adv)*`	number	no	1.0	Read the input files in this number of chunks
`civil_time *(adv)*`	number	no	—	Civil time zone value, if none, the time will be local solar time
`time`	number	yes	—	Local (solar) time (decimal hours)
`-p`	boolean	no	False	Do not incorporate the shadowing effect of terrain
`-m *(adv)*`	boolean	no	False	Use the low-memory version of the program
`incidout`	rasterDestination	no	—	incidence angle raster map
`beam_rad`	rasterDestination	no	—	Beam irradiance [W.m-2]
`diff_rad`	rasterDestination	no	—	Diffuse irradiance [W.m-2]
`refl_rad`	rasterDestination	no	—	Ground reflected irradiance [W.m-2]
`glob_rad`	rasterDestination	no	—	Global (total) irradiance/irradiation [W.m-2]

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: incidout (outputRaster), beam_rad (outputRaster), diff_rad (outputRaster), refl_rad (outputRaster), glob_rad (outputRaster)

Example usage

```
run grass:r.sun.incidout elevation=dem.tif aspect=dem.tif slope=dem.tif time=10
 ↪  aspect_value=270 slope_value=0 linke=dem.tif incidout=incidout.tif
```

This calls **r.sun.incidout**: elevation=dem.tif — Elevation layer [meters]; aspect=dem.tif — Aspect layer [decimal degrees]; slope=dem.tif — Name of the input slope raster map (terrain slope or solar panel inclination)...; time=10 — Local (solar) time (decimal hours); aspect_value=270 — A single value of the orientation (aspect), 270 is south; slope_value=0 — A single value of inclination (slope); linke=dem.tif — Name of the Linke atmospheric turbidity coefficient input raster map; incidout=incidout.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.sun.insoltime — r.sun.insoltime

group: Raster (r.*)

r.sun.insoltime - Solar irradiance and irradiation model (daily sums).

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Elevation layer [meters]
aspect	raster	yes	—	Aspect layer [decimal degrees]
aspect_value	number	no	270.0	A single value of the orientation (aspect), 270 is south
slope	raster	yes	—	Name of the input slope raster map (terrain slope or solar panel inclination) [decimal degrees]
slope_value	number	no	0.0	A single value of inclination (slope)
linke	raster	no	—	Name of the Linke atmospheric turbidity coefficient input raster map
albedo	raster	no	—	Name of the ground albedo coefficient input raster map
albedo_value	number	no	0.2	A single value of the ground albedo coefficient
lat	raster	no	—	Name of input raster map containing latitudes [decimal degrees]
long	raster	no	—	Name of input raster map containing longitudes [decimal degrees]
coeff_bh	raster	no	—	Name of real-sky beam radiation coefficient input raster map
coeff_dh	raster	no	—	Name of real-sky diffuse radiation coefficient input raster map
horizon_basemap	raster	no	—	The horizon information input map basename
horizon_step	number	no	—	Angle step size for multidirectional horizon [degrees]
day	number	no	1.0	No. of day of the year (1-365)
step *(adv)*	number	no	0.5	Time step when computing all-day radiation sums [decimal hours]

Parameter	Type	Required	Default	Description
declination *(adv)*	number	no	—	Declination value (overriding the internally computed value) [radians]
distance_step *(adv)*	number	no	1.0	Sampling distance step coefficient (0.5-1.5)
npartitions *(adv)*	number	no	1.0	Read the input files in this number of chunks
civil_time *(adv)*	number	no	—	Civil time zone value, if none, the time will be local solar time
-p	boolean	no	False	Do not incorporate the shadowing effect of terrain
-m *(adv)*	boolean	no	False	Use the low-memory version of the program
insol_time	rasterDestination	no	—	Insolation time [h]
beam_rad	rasterDestination	no	—	Irradiation raster map [Wh.m-2.day-1]
diff_rad	rasterDestination	no	—	Irradiation raster map [Wh.m-2.day-1]
refl_rad	rasterDestination	no	—	Irradiation raster map [Wh.m-2.day-1]
glob_rad	rasterDestination	no	—	Irradiance/irradiation raster map [Wh.m-2.day-1]
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: insol_time (outputRaster), beam_rad (outputRaster), diff_rad (outputRaster), refl_rad (outputRaster), glob_rad (outputRaster)

Example usage

```
run grass:r.sun.insoltime elevation=dem.tif aspect=dem.tif slope=dem.tif aspect_value=270
 ↪  slope_value=0 linke=dem.tif albedo=dem.tif insol_time=insol_time.tif
```

This calls **r.sun.insoltime**: elevation=dem.tif — Elevation layer [meters]; aspect=dem.tif — Aspect layer [decimal degrees]; slope=dem.tif — Name of the input slope raster map (terrain slope or solar panel inclination)...; aspect_value=270 — A single value of the orientation (aspect), 270 is south; slope_value=0 — A single value of inclination (slope); linke=dem.tif — Name of the Linke atmospheric turbidity coefficient input raster map; albedo=dem.tif — Name of the ground albedo coefficient input raster map; insol_time=insol_time.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.sunhours — r.sunhours

group: Raster (r.*)

Calculates solar elevation, solar azimuth, and sun hours.

Parameter	Type	Required	Default	Description
year	number	no	2017.0	Year
month	number	no	1.0	Month
day	number	no	1.0	Day
hour	number	no	12.0	Hour
minute	number	no	0.0	Minutes
second	number	no	0.0	Seconds
-t *(adv)*	boolean	no	False	Time is local sidereal time, not Greenwich standard time
-s *(adv)*	boolean	no	False	Do not use SOLPOS algorithm of NREL
elevation	rasterDestination	yes	—	Solar Elevation Angle
azimuth	rasterDestination	yes	—	Solar Azimuth Angle
sunhour	rasterDestination	yes	—	Sunshine Hours
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: elevation (outputRaster), azimuth (outputRaster), sunhour (outputRaster)

Example usage

```
run grass:r.sunhours year=2017 month=1 day=1 hour=12 minute=0 second=0
↪  elevation=elevation.tif
```

This calls **r.sunhours**: year=2017 — Year; month=1 — Month; day=1 — Day; hour=12 — Hour; minute=0 — Minutes; second=0 — Seconds; elevation=elevation.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.sunmask.datetime — r.sunmask.datetime

group: Raster (r.*)

r.sunmask.datetime - Calculates cast shadow areas from sun position and elevation raster map.

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Elevation raster layer [meters]
year	number	no	2000.0	year
month	number	no	1.0	month
day	number	no	1.0	day
hour	number	no	1.0	hour
minute	number	no	0.0	minute
second	number	no	0.0	second
timezone	number	no	0.0	East positive, offset from GMT
east	string	yes	—	Easting coordinate (point of interest)
north	string	yes	—	Northing coordinate (point of interest)
-z	boolean	no	True	Do not ignore zero elevation
-s	boolean	no	False	Calculate sun position only and exit
output	rasterDestination	yes	—	Shadows
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.sunmask.datetime elevation=dem.tif east=value north=value year=2000 month=1
↪  day=1 hour=1 output=output.tif
```

This calls **r.sunmask.datetime**: elevation=dem.tif — Elevation raster layer [meters]; east=value — Easting coordinate (point of interest); north=value — Northing coordinate (point of interest); year=2000 — year; month=1 — month; day=1 — day; hour=1 — hour; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.sunmask.position — r.sunmask.position

group: Raster (r.*)

r.sunmask.position - Calculates cast shadow areas from sun position and elevation raster map.

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Elevation raster layer [meters]
altitude	number	no	—	Altitude of the sun in degrees above the horizon
azimuth	number	no	—	Azimuth of the sun in degrees from north
east	string	no	False	Easting coordinate (point of interest)
north	string	no	False	Northing coordinate (point of interest)
-z	boolean	no	True	Do not ignore zero elevation
-s	boolean	no	False	Calculate sun position only and exit
output	rasterDestination	yes	—	Shadows
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.sunmask.position elevation=dem.tif altitude=10 azimuth=10 east=False
↪  north=False output=output.tif
```

This calls **r.sunmask.position**: elevation=dem.tif — Elevation raster layer [meters]; altitude=10 — Altitude of the sun in degrees above the horizon; azimuth=10 — Azimuth of the sun in degrees from north; east=False — Easting coordinate (point of interest); north=False — Northing coordinate (point of interest); output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.surf.area — r.surf.area

group: Raster (r.*)

Surface area estimation for rasters.

Parameter	Type	Required	Default	Description
map	raster	yes	—	Input layer
vscale	number	no	1.0	Vertical scale
units	enum	no	1	Units — options: 0=miles, 1=feet, 2=meters, 3=kilometers, 4=acres, 5=hectares
html	fileDestination	no	report.html	Area
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: html (outputHtml)

Example usage

```
run grass:r.surf.area map=dem.tif vscale=1 units=1 html=html.txt
```

This calls **r.surf.area**: map=dem.tif — Input layer; vscale=1 — Vertical scale; units=1 selects *feet*; html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.surf.contour — r.surf.contour

group: Raster (r.*)

Surface generation program from rasterized contours.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Raster layer with rasterized contours
output	rasterDestination	yes	—	DTM from contours
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.surf.contour input=dem.tif output=output.tif
```

This calls **r.surf.contour**: input=dem.tif — Raster layer with rasterized contours; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.surf.fractal — r.surf.fractal

group: Raster (r.*)

Creates a fractal surface of a given fractal dimension.

Parameter	Type	Required	Default	Description
dimension	number	no	2.05	Fractal dimension of surface (2 < D < 3)
number	number	no	0.0	Number of intermediate images to produce
output	rasterDestination	yes	—	Fractal Surface
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.surf.fractal dimension=2.05 number=0 output=output.tif
```

This calls **r.surf.fractal**: dimension=2.05 — Fractal dimension of surface (2 < D < 3); number=0 — Number of intermediate images to produce; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.surf.gauss — r.surf.gauss

group: Raster (r.*)

Creates a raster layer of Gaussian deviates.

Parameter	Type	Required	Default	Description
mean	number	no	0.0	Distribution mean
sigma	number	no	1.0	Standard deviation
output	rasterDestination	yes	—	Gaussian deviates
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.surf.gauss mean=0 sigma=1 output=output.tif
```

This calls **r.surf.gauss**: mean=0 — Distribution mean; sigma=1 — Standard deviation; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.surf.idw — r.surf.idw

group: Raster (r.*)

Surface interpolation utility for raster layers.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster layer
npoints	number	no	12.0	Number of interpolation points
-e	boolean	no	False	Output is the interpolation error
output	rasterDestination	yes	—	Interpolated IDW
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.surf.idw input=dem.tif npoints=12 output=output.tif
```

This calls **r.surf.idw**: input=dem.tif — Name of input raster layer; npoints=12 — Number of interpolation points; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.surf.random — r.surf.random

group: Raster (r.*)

Produces a raster layer of uniform random deviates whose range can be expressed by the user.

Parameter	Type	Required	Default	Description
min	number	no	0.0	Minimum random value
max	number	no	100.0	Maximum random value
-i	boolean	no	False	Create an integer raster layer

Parameter	Type	Required	Default	Description
output	rasterDestination	yes	—	Random
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.surf.random min=0 max=100 output=output.tif
```

This calls **r.surf.random**: min=0 — Minimum random value; max=100 — Maximum random value; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.terraflow — r.terraflow

group: Raster (r.*)

Flow computation for massive grids.

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Name of elevation raster map
-s	boolean	no	False	SFD (D8) flow (default is MFD)
d8cut	number	no	—	Routing using SFD (D8) direction
memory	number	no	300.0	Maximum memory to be used (in MB)
filled	rasterDestination	yes	—	Filled (flooded) elevation
direction	rasterDestination	yes	—	Flow direction
swatershed	rasterDestination	yes	—	Sink-watershed
accumulation	rasterDestination	yes	—	Flow accumulation
tci	rasterDestination	yes	—	Topographic convergence index (tci)
stats	fileDestination	yes	—	Runtime statistics
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: filled (outputRaster), direction (outputRaster), swatershed (outputRaster), accumulation (outputRaster), tci (outputRaster), stats (outputFile)

Example usage

```
run grass:r.terraflow elevation=dem.tif d8cut=10 memory=300 filled=filled.tif
```

This calls **r.terraflow**: elevation=dem.tif — Name of elevation raster map; d8cut=10 — Routing using SFD (D8) direction; memory=300 — Maximum memory to be used (in MB); filled=filled.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.texture — r.texture

group: Raster (r.*)

Generate images with textural features from a raster map.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
method	enum	no	[0]	Textural measurement method(s) — options: 0=asm, 1=contrast, 2=corr, 3=var, 4=idm, 5=sa, 6=se, 7=sv, 8=entr, 9=dv, 10=de, 11=moc1, 12=moc2
size	number	no	3.0	The size of moving window (odd and >= 3)
distance	number	no	1.0	The distance between two samples (>= 1)
-s *(adv)*	boolean	no	False	Separate output for each angle (0, 45, 90, 135)
-a *(adv)*	boolean	no	False	Calculate all textural measurements
output	folderDestination	yes	—	Texture files directory
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFolder)

Example usage

```
run grass:r.texture input=dem.tif method=0 size=3 distance=1 output=output/
```

This calls **r.texture**: input=dem.tif — Name of input raster map; method=0 selects *asm*; size=3 — The size of moving window (odd and >= 3); distance=1 — The distance between two samples (>= 1); output=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.thin — r.thin

group: Raster (r.*)

Thins non-zero cells that denote linear features in a raster layer.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer to thin
iterations	number	no	200.0	Maximum number of iterations
output	rasterDestination	yes	—	Thinned
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.thin input=dem.tif iterations=200 output=output.tif
```

This calls **r.thin**: input=dem.tif — Input raster layer to thin; iterations=200 — Maximum number of iterations; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.tile — r.tile

group: Raster (r.*)

Splits a raster map into tiles

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map
width	number	no	1024.0	Width of tiles (columns)
height	number	no	1024.0	Height of tiles (rows)
overlap	number	no	—	Overlap of tiles
output	folderDestination	yes	—	Tiles Directory
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFolder)

Example usage

```
run grass:r.tile input=dem.tif width=1024 height=1024 overlap=10 output=output/
```

This calls **r.tile**: input=dem.tif — Name of input raster map; width=1024 — Width of tiles (columns); height=1024 — Height of tiles (rows); overlap=10 — Overlap of tiles; output=output/ is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.tileset — r.tileset

group: Raster (r.*)

Produces tilings of the source projection for use in the destination region and projection.

Parameter	Type	Required	Default	Description
sourceproj	crs	yes	—	Source projection
sourcescale	number	no	1.0	Conversion factor from units to meters in source projection
destproj	crs	no	—	Destination projection
destscale	number	no	1.0	Conversion factor from units to meters in destination projection
maxcols	number	no	1024.0	Maximum number of columns for a tile in the source projection
maxrows	number	no	1024.0	Maximum number of rows for a tile in the source projection
overlap	number	no	0.0	Number of cells tiles should overlap in each direction
separator	string	no	pipe	Output field separator
-g *(adv)*	boolean	no	False	Produces shell script output
-w *(adv)*	boolean	no	False	Produces web map server query string output
html	fileDestination	yes	—	Tileset
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Outputs: html (outputHtml)

Example usage

```
run grass:r.tileset sourceproj=EPSG:6346 sourcescale=1 destproj=EPSG:6346 destscale=1
 ↪  maxcols=1024 maxrows=1024 overlap=0 html=html.txt
```

This calls **r.tileset**: sourceproj=EPSG:6346 — Source projection; sourcescale=1 — Conversion factor from units to meters in source projection; destproj=EPSG:6346 — Destination projection; destscale=1 — Conversion factor from units to meters in destination projection; maxcols=1024 — Maximum number of columns for a tile in the source projection; maxrows=1024 — Maximum number of rows for a tile in the source projection; overlap=0 — Number of cells tiles should overlap in each direction; html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.to.vect — r.to.vect

group: Raster (r.*)

Converts a raster into a vector layer.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input raster layer
type	enum	no	2	Feature type — options: 0=line, 1=point, 2=area
column	string	no	value	Name of attribute column to store value
-s	boolean	no	False	Smooth corners of area features
-v	boolean	no	False	Use raster values as categories instead of unique sequence
-z	boolean	no	False	Write raster values as z coordinate
-b	boolean	no	False	Do not build vector topology
-t	boolean	no	False	Do not create attribute table
output	vectorDestination	yes	—	Vectorized
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:r.to.vect input=dem.tif type=2 column=value output=output.gpkg
```

This calls **r.to.vect**: input=dem.tif — Input raster layer; type=2 selects *area*; column=value — Name of attribute column to store value; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.topidx — r.topidx

group: Raster (r.*)

Creates topographic index layer from elevation raster layer

Parameter	Type	Required	Default	Description
input	raster	yes	—	Input elevation layer
output	rasterDestination	yes	—	Topographic index
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.topidx input=dem.tif output=output.tif
```

This calls **r.topidx**: `input=dem.tif` — Input elevation layer; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.topmodel` — r.topmodel

group: Raster (r.*)

Simulates TOPMODEL which is a physically based hydrologic model.

Parameter	Type	Required	Default	Description
`parameters`	file	yes	—	Name of TOPMODEL parameters file
`topidxstats`	file	yes	—	Name of topographic index statistics file
`input`	file	yes	—	Name of rainfall and potential evapotranspiration data file
`timestep`	number	no	—	Time step. Generate output for this time step
`topidxclass`	number	no	—	Topographic index class. Generate output for this topographic index class
`output`	fileDestination	yes	—	TOPMODEL output
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Outputs: `output` (outputFile)

Example usage

```
run grass:r.topmodel parameters=input.dat topidxstats=input.dat input=input.dat
↪  timestep=10 topidxclass=10 output=output.txt
```

This calls **r.topmodel**: `parameters=input.dat` — Name of TOPMODEL parameters file; `topidxstats=input.dat` — Name of topographic index statistics file; `input=input.dat` — Name of rainfall and potential evapotranspiration data file; `timestep=10` — Time step. Generate output for this time step; `topidxclass=10` — Topographic index class. Generate output for this topographic index class; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.topmodel.topidxstats` — r.topmodel.topidxstats

group: Raster (r.*)

r.topmodel.topidxstats - Builds a TOPMODEL topographic index statistics file.

Parameter	Type	Required	Default	Description
`topidx`	raster	yes	—	Name of input topographic index raster map
`ntopidxclasses`	number	no	30.0	Number of topographic index classes
`outtopidxstats`	fileDestination	yes	—	TOPMODEL topographic index statistics file
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `outtopidxstats` (outputFile)

Example usage

```
run grass:r.topmodel.topidxstats topidx=dem.tif ntopidxclasses=30
↪  outtopidxstats=outtopidxstats.txt
```

This calls **r.topmodel.topidxstats**: `topidx=dem.tif` — Name of input topographic index raster map; `ntopidxclasses=30` — Number of topographic index classes; `outtopidxstats=outtopidxstats.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.transect — r.transect

group: Raster (r.*)

Outputs raster map layer values lying along user defined transect line(s).

Parameter	Type	Required	Default	Description
map	raster	yes	—	Raster map to be queried
line	string	yes	—	Transect definition: east,north,azimuth,distance[,east,north,azimuth,distance,...]
null_value	string	no	*	String representing NULL value
-g *(adv)*	boolean	no	False	Output easting and northing in first two columns of four column output
html	fileDestination	yes	—	Transect file
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: html (outputHtml)

Example usage

```
run grass:r.transect map=dem.tif line=value null_value=* html=html.txt
```

This calls **r.transect**: map=dem.tif — Raster map to be queried; line=value — Transect definition:...; null_value=* — String representing NULL value; html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.univar — r.univar

group: Raster (r.*)

Calculates univariate statistics from the non-null cells of a raster map.

Parameter	Type	Required	Default	Description
map	multilayer	yes	—	Name of raster map(s)
zones	raster	no	—	Raster map used for zoning, must be of type CELL
percentile	string	no	—	Percentile to calculate (comma separated list if multiple) (requires extended statistics flag)
separator	string	no	pipe	Field separator. Special characters: pipe, comma, space, tab, newline
-e *(adv)*	boolean	no	False	Calculate extended statistics
output	fileDestination	yes	—	Univariate results
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: output (outputFile)

Example usage

```
run grass:r.univar map="a.gpkg;b.gpkg" zones=dem.tif percentile=value separator=pipe
↪  output=output.txt
```

This calls **r.univar**: map="a.gpkg;b.gpkg" — Name of raster map(s); zones=dem.tif — Raster map used for zoning, must be of type CELL; percentile=value — Percentile to calculate (comma separated list if multiple) (requires extended...; separator=pipe — Field separator. Special characters: pipe, comma, space, tab, newline; output=output.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.uslek — r.uslek

group: Raster (r.*)

Computes USLE Soil Erodibility Factor (K).

Parameter	Type	Required	Default	Description
psand	raster	yes	—	Name of soil sand fraction raster map [0.0-1.0]
pclay	raster	yes	—	Name of soil clay fraction raster map [0.0-1.0]
psilt	raster	yes	—	Name of soil silt fraction raster map [0.0-1.0]
pomat	raster	yes	—	Name of soil organic matter raster map [0.0-1.0]
output	rasterDestination	yes	—	USLE R Raster
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.uslek psand=dem.tif pclay=dem.tif psilt=dem.tif pomat=dem.tif
↪  output=output.tif
```

This calls **r.uslek**: psand=dem.tif — Name of soil sand fraction raster map [0.0-1.0]; pclay=dem.tif — Name of soil clay fraction raster map [0.0-1.0]; psilt=dem.tif — Name of soil silt fraction raster map [0.0-1.0]; pomat=dem.tif — Name of soil organic matter raster map [0.0-1.0]; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.usler — r.usler

group: Raster (r.*)

Computes USLE R factor, Rainfall erosivity index.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of annual precipitation raster map [mm/year]
method	enum	no	0	Name of USLE R equation — options: 0=roose, 1=morgan, 2=foster, 3=elswaify
output	rasterDestination	yes	—	USLE R Raster
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.usler input=dem.tif method=0 output=output.tif
```

This calls **r.usler**: input=dem.tif — Name of annual precipitation raster map [mm/year]; method=0 selects *roose*; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.viewshed — r.viewshed

group: Raster (r.*)

Computes the viewshed of a point on an elevation raster map.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Elevation
coordinates	point	no	0.0,0.0	Coordinate identifying the viewing position
observer_elevation	number	no	1.75	Viewing elevation above the ground
target_elevation	number	no	0.0	Offset for target elevation above the ground
max_distance	number	no	-1.0	Maximum visibility radius. By default infinity (-1)
refraction_coeff	number	no	0.14286	Refraction coefficient
memory	number	no	500.0	Amount of memory to use in MB
-c *(adv)*	boolean	no	False	Consider earth curvature (current ellipsoid)
-r *(adv)*	boolean	no	False	Consider the effect of atmospheric refraction
-b *(adv)*	boolean	no	False	Output format is invisible = 0, visible = 1
-e *(adv)*	boolean	no	False	Output format is invisible = NULL, else current elev - viewpoint_elev
output	rasterDestination	yes	—	Intervisibility
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster)

Example usage

```
run grass:r.viewshed input=dem.tif coordinates=100,100 observer_elevation=1.75
 ↪  target_elevation=0 max_distance=-1 refraction_coeff=0.14286 memory=500
 ↪  output=output.tif
```

This calls **r.viewshed**: input=dem.tif — Elevation; coordinates=100,100 — Coordinate identifying the viewing position; observer_elevation=1.75 — Viewing elevation above the ground; target_elevation=0 — Offset for target elevation above the ground; max_distance=-1 — Maximum visibility radius. By default infinity (-1); refraction_coeff=0.14286 — Refraction coefficient; memory=500 — Amount of memory to use in MB; output=output.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.volume — r.volume

group: Raster (r.*)

Calculates the volume of data ”clumps”.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Name of input raster map representing data that will be summed within clumps
clump	raster	yes	—	Clumps layer (preferably the output of r.clump)
-f *(adv)*	boolean	no	False	Generate unformatted report
centroids	vectorDestination	yes	—	Centroids
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Parameter	Type	Required	Default	Description
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: centroids (outputVector)

Example usage

```
run grass:r.volume input=dem.tif clump=dem.tif centroids=centroids.gpkg
```

This calls **r.volume**: input=dem.tif — Name of input raster map representing data that will be summed within clumps; clump=dem.tif — Clumps layer (preferably the output of r.clump); centroids=centroids.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.walk.coords — r.walk.coords

group: Raster (r.*)

r.walk.coords - Creates a raster map showing the anisotropic cumulative cost of moving between different geographic locations on an input raster map whose cell category values represent cost from a list of coordinates.

Parameter	Type	Required	Default	Description
elevation	raster	yes	—	Name of input elevation raster map
friction	raster	yes	—	Name of input raster map containing friction costs
start_coordinates	string	yes	—	Coordinates of starting point(s) (a list of E,N)
stop_coordinates	string	no	—	Coordinates of stopping point(s) (a list of E,N)
walk_coeff	string	no	0.72,6.0,1.9998,-1.9998	Coefficients for walking energy formula parameters a,b,c,d
lambda	number	no	1.0	Lambda coefficients for combining walking energy and friction cost
slope_factor	number	no	-0.2125	Slope factor determines travel energy cost per height step
max_cost	number	no	0.0	Maximum cumulative cost
null_cost	number	no	—	Cost assigned to null cells. By default, null cells are excluded
memory *(adv)*	number	no	300.0	Maximum memory to be used in MB
-k *(adv)*	boolean	no	False	Use the 'Knight's move'; slower, but more accurate
-n *(adv)*	boolean	no	False	Keep null values in output raster layer
output	rasterDestination	yes	—	Cumulative cost
outdir	rasterDestination	yes	—	Movement Directions
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: output (outputRaster), outdir (outputRaster)

Example usage

```
run grass:r.walk.coords elevation=dem.tif friction=dem.tif start_coordinates=value
 ↪  stop_coordinates=value walk_coeff=0.72,6.0,1.9998,-1.9998 lambda=1
 ↪  slope_factor=-0.2125 output=output.tif
```

This calls **r.walk.coords**: `elevation=dem.tif` — Name of input elevation raster map; `friction=dem.tif` — Name of input raster map containing friction costs; `start_coordinates=value` — Coordinates of starting point(s) (a list of E,N); `stop_coordinates=value` — Coordinates of stopping point(s) (a list of E,N); `walk_coeff=0.72,6.0,1.9998,-1.9998` — Coefficients for walking energy formula parameters a,b,c,d; `lambda=1` — Lambda coefficients for combining walking energy and friction cost; `slope_factor=-0.2125` — Slope factor determines travel energy cost per height step; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.walk.points` — r.walk.points

group: Raster (r.*)

r.walk.points - Creates a raster map showing the anisotropic cumulative cost of moving between different geographic locations on an input raster map whose cell category values represent cost from point vector layers.

Parameter	Type	Required	Default	Description
`elevation`	raster	yes	—	Name of input elevation raster map
`friction`	raster	yes	—	Name of input raster map containing friction costs
`start_points`	source	yes	—	Start points
`stop_points`	source	no	—	Stop points
`walk_coeff`	string	no	0.72,6.0,1.9998,-1.9998	Coefficients for walking energy formula parameters a,b,c,d
`lambda`	number	no	1.0	Lambda coefficients for combining walking energy and friction cost
`slope_factor`	number	no	-0.2125	Slope factor determines travel energy cost per height step
`max_cost`	number	no	0.0	Maximum cumulative cost
`null_cost`	number	no	—	Cost assigned to null cells. By default, null cells are excluded
`memory *(adv)*`	number	no	300.0	Maximum memory to be used in MB
`-k *(adv)*`	boolean	no	False	Use the 'Knight's move'; slower, but more accurate
`-n *(adv)*`	boolean	no	False	Keep null values in output raster layer
`output`	rasterDestination	yes	—	Cumulative cost
`outdir`	rasterDestination	yes	—	Movement Directions
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputRaster), `outdir` (outputRaster)

Example usage

```
run grass:r.walk.points elevation=dem.tif friction=dem.tif start_points=input.gpkg
 ↪  stop_points=input.gpkg walk_coeff=0.72,6.0,1.9998,-1.9998 lambda=1
 ↪  slope_factor=-0.2125 output=output.tif
```

This calls **r.walk.points**: `elevation=dem.tif` — Name of input elevation raster map; `friction=dem.tif` — Name of input raster map containing friction costs; `start_points=input.gpkg` — Start points; `stop_points=input.gpkg` — Stop points; `walk_coeff=0.72,6.0,1.9998,-1.9998` — Coefficients for walking energy formula parameters a,b,c,d; `lambda=1` — Lambda coefficients for combining walking energy and friction cost; `slope_factor=-0.2125` — Slope factor determines travel energy cost per height step; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.walk.rast` — r.walk.rast

group: Raster (r.*)

r.walk.rast - Creates a raster map showing the anisotropic cumulative cost of moving between different geographic locations on an input raster map whose cell category values represent cost from a raster.

Parameter	Type	Required	Default	Description
`elevation`	raster	yes	—	Name of input elevation raster map
`friction`	raster	yes	—	Name of input raster map containing friction costs
`start_raster`	raster	yes	—	Name of starting raster points map (all non-NULL cells are starting points)
`walk_coeff`	string	no	0.72,6.0,1.9998,-1.9998	Coefficients for walking energy formula parameters a,b,c,d
`lambda`	number	no	1.0	Lambda coefficients for combining walking energy and friction cost
`slope_factor`	number	no	-0.2125	Slope factor determines travel energy cost per height step
`max_cost`	number	no	0.0	Maximum cumulative cost
`null_cost`	number	no	—	Cost assigned to null cells. By default, null cells are excluded
`memory *(adv)*`	number	no	300.0	Maximum memory to be used in MB
`-k *(adv)*`	boolean	no	False	Use the 'Knight's move'; slower, but more accurate
`-n *(adv)*`	boolean	no	False	Keep null values in output raster layer
`output`	rasterDestination	yes	—	Cumulative cost
`outdir`	rasterDestination	yes	—	Movement Directions
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster), `outdir` (outputRaster)

Example usage

```
run grass:r.walk.rast elevation=dem.tif friction=dem.tif start_raster=dem.tif
 ↪  walk_coeff=0.72,6.0,1.9998,-1.9998 lambda=1 slope_factor=-0.2125 max_cost=0
 ↪  output=output.tif
```

This calls **r.walk.rast**: `elevation=dem.tif` — Name of input elevation raster map; `friction=dem.tif` — Name of input raster map containing friction costs; `start_raster=dem.tif` —

Name of starting raster points map (all non-NULL cells are starting points); `walk_coeff=0.72,6.0,1.9998,-1.9998` — Coefficients for walking energy formula parameters a,b,c,d; `lambda=1` — Lambda coefficients for combining walking energy and friction cost; `slope_factor=-0.2125` — Slope factor determines travel energy cost per height step; `max_cost=0` — Maximum cumulative cost; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.water.outlet` — r.water.outlet

group: Raster (r.*)

Watershed basin creation program.

Parameter	Type	Required	Default	Description
`input`	raster	yes	—	Drainage direction raster
`coordinates`	point	no	0.0,0.0	Coordinates of outlet point
`output`	rasterDestination	yes	—	Basin
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: `output` (outputRaster)

Example usage

```
run grass:r.water.outlet input=dem.tif coordinates=100,100 output=output.tif
```

This calls **r.water.outlet**: `input=dem.tif` — Drainage direction raster; `coordinates=100,100` — Coordinates of outlet point; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.watershed` — r.watershed

group: Raster (r.*)

Watershed basin analysis program.

Parameter	Type	Required	Default	Description
`elevation`	raster	yes	—	Elevation
`depression`	raster	no	—	Locations of real depressions
`flow`	raster	no	—	Amount of overland flow per cell
`disturbed_land`	raster	no	—	Percent of disturbed land, for USLE
`blocking`	raster	no	—	Terrain blocking overland surface flow, for USLE
`threshold`	number	no	—	Minimum size of exterior watershed basin
`max_slope_length`	number	no	—	Maximum length of surface flow, for USLE
`convergence`	number	no	5.0	Convergence factor for MFD (1-10)
`memory`	number	no	300.0	Maximum memory to be used with -m flag (in MB)
`-s`	boolean	no	False	Enable Single Flow Direction (D8) flow (default is Multiple Flow Direction)
`-m`	boolean	no	False	Enable disk swap memory option (-m): Operation is slow
`-4`	boolean	no	False	Allow only horizontal and vertical flow of water

Parameter	Type	Required	Default	Description
-a	boolean	no	False	Use positive flow accumulation even for likely underestimates
-b	boolean	no	False	Beautify flat areas
accumulation	rasterDestination	no	—	Number of cells that drain through each cell
drainage	rasterDestination	no	—	Drainage direction
basin	rasterDestination	no	—	Unique label for each watershed basin
stream	rasterDestination	no	—	Stream segments
half_basin	rasterDestination	no	—	Half-basins
length_slope	rasterDestination	no	—	Slope length and steepness (LS) factor for USLE
slope_steepness	rasterDestination	no	—	Slope steepness (S) factor for USLE
tci	rasterDestination	no	—	Topographic index ln(a / tan(b))
spi	rasterDestination	no	—	Stream power index a * tan(b)
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)

Outputs: accumulation (outputRaster), drainage (outputRaster), basin (outputRaster), stream (outputRaster), half_basin (outputRaster), length_slope (outputRaster), slope_steepness (outputRaster), tci (outputRaster), spi (outputRaster)

Example usage

```
run grass:r.watershed elevation=dem.tif depression=dem.tif flow=dem.tif
↪ disturbed_land=dem.tif blocking=dem.tif threshold=10 max_slope_length=10
↪ accumulation=accumulation.tif
```

This calls **r.watershed**: elevation=dem.tif — Elevation; depression=dem.tif — Locations of real depressions; flow=dem.tif — Amount of overland flow per cell; disturbed_land=dem.tif — Percent of disturbed land, for USLE; blocking=dem.tif — Terrain blocking overland surface flow, for USLE; threshold=10 — Minimum size of exterior watershed basin; max_slope_length=10 — Maximum length of surface flow, for USLE; accumulation=accumulation.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:r.what.color — r.what.color

group: Raster (r.*)

Queries colors for a raster map layer.

Parameter	Type	Required	Default	Description
input	raster	yes	—	Raster map to query colors
value	string	no	—	Values to query colors for (comma separated list)
format	string	no	%d:%d:%d	Output format (printf-style)
html	fileDestination	yes	—	Colors file
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: html (outputHtml)

Example usage

```
run grass:r.what.color input=dem.tif value=value format=%d:%d:%d html=html.txt
```

This calls **r.what.color**: `input=dem.tif` — Raster map to query colors; `value=value` — Values to query colors for (comma separated list); `format=%d:%d:%d` — Output format (printf-style); `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.what.coords` — r.what.coords

group: Raster (r.*)

r.what.coords - Queries raster maps on their category values and category labels on a point.

Parameter	Type	Required	Default	Description
`map`	raster	yes	—	Name of raster map
`coordinates`	point	no	0.0, 0.0	Coordinates for query (east, north)
`null_value`	string	no	*	String representing NULL value
`separator`	string	no	pipe	Field separator. Special characters: pipe, comma, space, tab, newlineString representing NULL value
`cache`	number	no	500.0	Size of point cache
`-n *(adv)*`	boolean	no	False	Output header row
`-f *(adv)*`	boolean	no	False	Show the category labels of the grid cell(s)
`-r *(adv)*`	boolean	no	False	Output color values as RRR:GGG:BBB
`-i *(adv)*`	boolean	no	False	Output integer category values, not cell values
`-c *(adv)*`	boolean	no	False	Turn on cache reporting
`output`	fileDestination	yes	—	Raster Value File
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)

Outputs: `output` (outputFile)

Example usage

```
run grass:r.what.coords map=dem.tif coordinates=100,100 null_value=* separator=pipe
↪  cache=500 output=output.txt
```

This calls **r.what.coords**: `map=dem.tif` — Name of raster map; `coordinates=100,100` — Coordinates for query (east, north); `null_value=*` — String representing NULL value; `separator=pipe` — Field separator. Special characters: pipe, comma, space, tab, newlineString...; `cache=500` — Size of point cache; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:r.what.points` — r.what.points

group: Raster (r.*)

r.what.points - Queries raster maps on their category values and category labels on a layer of points.

Parameter	Type	Required	Default	Description
`map`	raster	yes	—	Name of raster map
`points`	source	yes	—	Name of vector points layer for query
`null_value`	string	no	*	String representing NULL value
`separator`	string	no	pipe	Field separator. Special characters: pipe, comma, space, tab, newline
`cache`	number	no	500.0	Size of point cache
`-n *(adv)*`	boolean	no	False	Output header row
`-f *(adv)*`	boolean	no	False	Show the category labels of the grid cell(s)

Parameter	Type	Required	Default	Description
`-r *(adv)*`	boolean	no	False	Output color values as RRR:GGG:BBB
`-i *(adv)*`	boolean	no	False	Output integer category values, not cell values
`-c *(adv)*`	boolean	no	False	Turn on cache reporting
`output`	fileDestination	yes	—	Raster Values File
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputFile)

Example usage

```
run grass:r.what.points map=dem.tif points=input.gpkg null_value=* separator=pipe
 ↪  cache=500 output=output.txt
```

This calls **r.what.points**: `map=dem.tif` — Name of raster map; `points=input.gpkg` — Name of vector points layer for query; `null_value=*` — String representing NULL value; `separator=pipe` — Field separator. Special characters: pipe, comma, space, tab, newline; `cache=500` — Size of point cache; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.buffer` — v.buffer

group: Vector (v.*)

Creates a buffer around vector features of given type.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector layer
`cats`	string	no	—	Category values
`where`	string	no	—	WHERE conditions of SQL statement without 'where' keyword
`type`	enum	no	[0, 1, 4]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area
`distance`	number	no	—	Buffer distance in map units
`minordistance`	number	no	—	Buffer distance along minor axis in map units
`angle`	number	no	0.0	Angle of major axis in degrees
`layer`	string	no	-1	Layer number or name ('-1' for all layers)
`column`	field	no	—	Name of column to use for buffer distances
`scale`	number	no	1.0	Scaling factor for attribute column values
`tolerance`	number	no	0.01	Maximum distance between theoretical arc and polygon segments as multiple of buffer
`-s *(adv)*`	boolean	no	False	Make outside corners straight
`-c *(adv)*`	boolean	no	False	Do not make caps at the ends of polylines
`-t *(adv)*`	boolean	no	False	Transfer categories and attributes
`output`	vectorDestination	yes	—	Buffer
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)

Parameter	Type	Required	Default	Description
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.buffer input=input.gpkg cats=value where=value type=0 distance=10
↪  minordistance=10 angle=0 output=output.gpkg
```

This calls **v.buffer**: input=input.gpkg — Input vector layer; cats=value — Category values; where=value — WHERE conditions of SQL statement without 'where' keyword; type=0 selects *point*; distance=10 — Buffer distance in map units; minordistance=10 — Buffer distance along minor axis in map units; angle=0 — Angle of major axis in degrees; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.build.check — v.build.check

group: Vector (v.*)

v.build.check - Checks for topological errors.

Parameter	Type	Required	Default	Description
map	source	yes	—	Name of vector map
error	vectorDestination	yes	—	Topological errors
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: error (outputVector)

Example usage

```
run grass:v.build.check map=input.gpkg error=error.gpkg
```

This calls **v.build.check**: map=input.gpkg — Name of vector map; error=error.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.build.polylines — v.build.polylines

group: Vector (v.*)

Builds polylines from lines or boundaries.

Parameter	Type	Required	Default	Description
input	source	yes	—	Name of input vector map

Parameter	Type	Required	Default	Description
`cats`	enum	no	0	Category number mode — options: 0=no, 1=first, 2=multi, 3=same
`type`	enum	no	[0, 1]	Input feature type — options: 0=line, 1=boundary
`output`	vectorDestination	yes	—	Polylines
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.build.polylines input=input.gpkg cats=0 type=0 output=output.gpkg
```

This calls **v.build.polylines**: `input=input.gpkg` — Name of input vector map; `cats=0` selects *no*; `type=0` selects *line*; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.class` — v.class

group: Vector (v.*)

Classifies attribute data, e.g. for thematic mapping.

Parameter	Type	Required	Default	Description
`map`	source	yes	—	Input vector layer
`column`	field	yes	—	Column name or expression
`where`	string	no	—	WHERE conditions of SQL statement without 'where' keyword
`algorithm`	enum	no	0	Algorithm to use for classification — options: 0=int, 1=std, 2=qua, 3=equ
`nbclasses`	number	no	3.0	Number of classes to define
`-g`	boolean	no	True	Print only class breaks (without min and max)
`html`	fileDestination	no	report.html	Classification
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `html` (outputHtml)

Example usage

```
run grass:v.class map=input.gpkg column=field1 where=value algorithm=0 nbclasses=3
 ↪  html=html.txt
```

This calls **v.class**: `map=input.gpkg` — Input vector layer; `column=field1` — Column name or expression; `where=value` — WHERE conditions of SQL statement without 'where' keyword; `algorithm=0` selects *int*; `nbclasses=3` — Number of classes to define; `html=html.txt` is where the

result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.clean` — v.clean

group: Vector (v.*)

Toolset for cleaning topology of vector map.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Layer to clean
`type`	enum	no	[0, 1, 2, 3, 4, 5, 6]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area, 5=face, 6=kernel
`tool`	enum	no	[0]	Cleaning tool — options: 0=break, 1=snap, 2=rmdangle, 3=chdangle, 4=rmbridge, 5=chbridge, 6=rmdupl, 7=rmdac, 8=bpol, 9=prune, 10=rmarea, 11=rmline, 12=rmsa
`threshold`	string	no	—	Threshold (comma separated for each tool)
`-b *(adv)*`	boolean	no	False	Do not build topology for the output vector
`-c *(adv)*`	boolean	no	False	Combine tools with recommended follow-up tools
`output`	vectorDestination	yes	—	Cleaned
`error`	vectorDestination	yes	—	Errors
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector), `error` (outputVector)

Example usage

```
run grass:v.clean input=input.gpkg type=0 tool=0 threshold=value output=output.gpkg
```

This calls **v.clean**: `input=input.gpkg` — Layer to clean; `type=0` selects *point*; `tool=0` selects *break*; `threshold=value` — Threshold (comma separated for each tool); `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.cluster` — v.cluster

group: Vector (v.*)

Performs cluster identification

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input layer

Parameter	Type	Required	Default	Description
distance	number	no	—	Maximum distance to neighbors
min	number	no	—	Minimum number of points to create a cluster
method	enum	no	[0]	Clustering method — options: 0=dbscan, 1=dbscan2, 2=density, 3=optics, 4=optics2
-2 *(adv)*	boolean	no	False	Force 2D clustering
-b *(adv)*	boolean	no	False	Do not build topology
-t *(adv)*	boolean	no	False	Do not create attribute table
output	vectorDestination	yes	—	Clustered
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.cluster input=input.gpkg distance=10 min=10 method=0 output=output.gpkg
```

This calls **v.cluster**: input=input.gpkg — Input layer; distance=10 — Maximum distance to neighbors; min=10 — Minimum number of points to create a cluster; method=0 selects *dbscan*; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.db.select — v.db.select

group: Vector (v.*)

Prints vector map attributes

Parameter	Type	Required	Default	Description
map	source	yes	—	Input vector map
layer	number	no	1.0	Layer Number
columns	string	no	—	Name of attribute column(s), comma separated
-c *(adv)*	boolean	no	False	Do not include column names in output
separator	string	no	,	Output field separator
where *(adv)*	string	no	—	WHERE conditions of SQL statement without 'where' keyword
group *(adv)*	string	no	—	GROUP BY conditions of SQL statement without 'group by' keyword
vertical_separator *(adv)*	string	no	—	Output vertical record separator
null_value *(adv)*	string	no	—	Null value indicator
-v *(adv)*	boolean	no	False	Vertical output (instead of horizontal)
-r *(adv)*	boolean	no	False	Print minimal region extent of selected vector features instead of attributes
file	fileDestination	yes	—	Attributes
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: `file` (outputFile)

Example usage

```
run grass:v.db.select map=input.gpkg layer=1 columns=value separator=, file=file.txt
```

This calls **v.db.select**: `map=input.gpkg` — Input vector map; `layer=1` — Layer Number; `columns=value` — Name of attribute column(s), comma separated; `separator=,` — Output field separator; `file=file.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.decimate` — v.decimate

group: Vector (v.*)

Decimates a point cloud

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector
`zrange`	range	no	—	Filter range for z data (min,max)
`cats`	string	no	—	Category values
`skip`	number	no	—	Throw away every n-th point
`preserve`	number	no	—	Preserve only every n-th point
`offset`	number	no	—	Skip first n points
`limit`	number	no	—	Copy only n points
`zdiff`	number	no	—	Minimal difference of z values
`cell_limit`	number	no	—	Preserve only n points per grid cell
`-g *(adv)*`	boolean	no	False	Apply grid-based decimation
`-f *(adv)*`	boolean	no	False	Use only first point in grid cell during grid-based decimation
`-c *(adv)*`	boolean	no	False	Only one point per cat in grid cell
`-z *(adv)*`	boolean	no	False	Use z in grid decimation
`-x *(adv)*`	boolean	no	False	Store only the coordinates, throw away categories
`-b *(adv)*`	boolean	no	False	Do not build topology
`output`	vectorDestination	yes	—	Output vector map
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.decimate input=input.gpkg cats=value skip=10 preserve=10 offset=10 limit=10
↪ output=output.gpkg
```

This calls **v.decimate**: `input=input.gpkg` — Input vector; `cats=value` — Category values; `skip=10` — Throw away every n-th point; `preserve=10` — Preserve only every n-th point; `offset=10` — Skip first n points; `limit=10` — Copy only n points; `output=output.gpkg` is where the

result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.delaunay` — v.delaunay

group: Vector (v.*)

Creates a Delaunay triangulation from an input vector map containing points or centroids.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector layer
`-r`	boolean	no	False	Use only points in current region
`-l`	boolean	no	False	Output triangulation as a graph (lines), not areas
`output`	vectorDestination	yes	—	Delaunay triangulation
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.delaunay input=input.gpkg output=output.gpkg
```

This calls **v.delaunay**: `input=input.gpkg` — Input vector layer; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.dissolve` — v.dissolve

group: Vector (v.*)

Dissolves boundaries between adjacent areas sharing a common category number or attribute.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector layer
`column`	field	no	—	Name of column used to dissolve common boundaries
`output`	vectorDestination	yes	—	Dissolved
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)

Parameter	Type	Required	Default	Description
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.dissolve input=input.gpkg column=field1 output=output.gpkg
```

This calls **v.dissolve**: input=input.gpkg — Input vector layer; column=field1 — Name of column used to dissolve common boundaries; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.distance — v.distance

group: Vector (v.*)

Finds the nearest element in vector map 'to' for elements in vector map 'from'.

Parameter	Type	Required	Default	Description
from	source	yes	—	'from' vector map
from_type *(adv)*	enum	no	[0, 1, 3]	'from' feature type — options: 0=point, 1=line, 2=boundary, 3=area, 4=centroid
to	source	yes	—	'to' vector map
to_type *(adv)*	enum	no	[0, 1, 3]	'to' feature type — options: 0=point, 1=line, 2=boundary, 3=area, 4=centroid
dmax	number	no	-1.0	Maximum distance or -1.0 for no limit
dmin	number	no	-1.0	Minimum distance or -1.0 for no limit
upload	enum	no	[0]	'upload': Values describing the relation between two nearest features — options: 0=cat, 1=dist, 2=to_x, 3=to_y, 4=to_along, 5=to_angle, 6=to_attr
column	field	yes	—	Column name(s) where values specified by 'upload' option will be uploaded
to_column	field	no	—	Column name of nearest feature (used with upload=to_attr)
from_output	vectorDestination	yes	—	Nearest
output	vectorDestination	yes	—	Distance
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: from_output (outputVector), output (outputVector)

Example usage

```
run grass:v.distance from=input.gpkg to=input.gpkg column=field1 dmax=-1 dmin=-1 upload=0
 ↪  to_column=field1 from_output=from_output.gpkg
```

This calls **v.distance**: from=input.gpkg — 'from' vector map; to=input.gpkg — 'to' vector map; column=field1 — Column name(s) where values specified by 'upload' option will be uploaded;

dmax=-1 — Maximum distance or -1.0 for no limit; dmin=-1 — Minimum distance or -1.0 for no limit; upload=0 selects *cat*; to_column=field1 — Column name of nearest feature (used with upload=to_attr); from_output=from_output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.drape — v.drape

group: Vector (v.*)

Converts 2D vector features to 3D by sampling of elevation raster map.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector layer
where	string	no	—	WHERE conditions of SQL statement without 'where' keyword
type	enum	no	[0, 1, 3, 4]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid
elevation	raster	yes	—	Elevation raster map for height extraction
method	enum	no	0	Sampling method — options: 0=nearest, 1=bilinear, 2=bicubic
scale	number	no	1.0	Scale factor sampled raster values
null_value	number	no	—	Height for sampled raster NULL values
output	vectorDestination	yes	—	3D vector
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.drape input=input.gpkg elevation=dem.tif where=value type=0 method=0 scale=1
↪  null_value=10 output=output.gpkg
```

This calls **v.drape**: input=input.gpkg — Input vector layer; elevation=dem.tif — Elevation raster map for height extraction; where=value — WHERE conditions of SQL statement without 'where' keyword; type=0 selects *point*; method=0 selects *nearest*; scale=1 — Scale factor sampled raster values; null_value=10 — Height for sampled raster NULL values; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.edit — v.edit

group: Vector (v.*)

Edits a vector map, allows adding, deleting and modifying selected vector features.

Parameter	Type	Required	Default	Description
map	source	yes	—	Name of vector layer
type	enum	no	[0, 1, 2, 3]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid
tool	enum	no	0	Tool — options: 0=create, 1=add, 2=delete, 3=copy, 4=move, 5=flip, 6=catadd, 7=catdel, 8=merge, 9=break, 10=snap, 11=connect, 12=chtype, 13=vertexadd, 14=vertexdel, 15=vertexmove, 16=areadel, 17=zbulk, 18=select
input	file	no	—	ASCII file for add tool
move	string	no	—	Difference in x,y,z direction for moving feature or vertex
threshold	string	no	—	Threshold distance (coords,snap,query)
ids	string	no	—	Feature ids
cats	string	no	—	Category values
coords	string	no	—	List of point coordinates
bbox	extent	no	—	Bounding box for selecting features
polygon	string	no	—	Polygon for selecting features
where	string	no	—	WHERE conditions of SQL statement without 'where' keyword
query	enum	no	—	Query tool — options: 0=length, 1=dangle
bgmap	source	no	—	Name of background vector map
snap	enum	no	0	Snap added or modified features in the given threshold to the nearest existing feature — options: 0=no, 1=node, 2=vertex
zbulk	string	no	—	Starting value and step for z bulk-labeling. Pair: value,step (e.g. 1100,10)
-r	boolean	no	False	Reverse selection
-c	boolean	no	False	Close added boundaries (using threshold distance)
-n	boolean	no	False	Do not expect header of input data
-b	boolean	no	False	Do not build topology
-1	boolean	no	False	Modify only first found feature in bounding box
output	vectorDestination	yes	—	Edited
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.edit map=input.gpkg type=0 tool=0 input=input.dat move=value threshold=value
↪  ids=value output=output.gpkg
```

This calls **v.edit**: `map=input.gpkg` — Name of vector layer; `type=0` selects *point*; `tool=0` selects *create*; `input=input.dat` — ASCII file for add tool; `move=value` — Difference in x,y,z direction for moving feature or vertex; `threshold=value` — Threshold distance (coords,snap,query); `ids=value` — Feature ids; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.extract — v.extract

group: Vector (v.*)

Selects vector objects from a vector layer and creates a new layer containing only the selected objects.

Parameter	Type	Required	Default	Description
input	source	yes	—	Vector layer
where	string	no	—	WHERE conditions of SQL statement without 'where' keyword
type	enum	no	[0, 1, 3, 4, 5, 6]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area, 5=face
file	file	no	—	Input text file with category numbers/number ranges to be extracted
random	number	no	—	Number of random categories matching vector objects to extract
new	number	no	-1.0	Desired new category value (enter -1 to keep original categories)
-d *(adv)*	boolean	no	True	Dissolve common boundaries
-t *(adv)*	boolean	no	False	Do not copy attributes
-r *(adv)*	boolean	no	False	Reverse selection
output	vectorDestination	yes	—	Selected
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.extract input=input.gpkg where=value type=0 file=input.dat random=10 new=-1
 ↪  output=output.gpkg
```

This calls **v.extract**: input=input.gpkg — Vector layer; where=value — WHERE conditions of SQL statement without 'where' keyword; type=0 selects *point*; file=input.dat — Input text file with category numbers/number ranges to be extracted; random=10 — Number of random categories matching vector objects to extract; new=-1 — Desired new category value (enter -1 to keep original categories); output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.extrude — v.extrude

group: Vector (v.*)

Extrudes flat vector object to 3D with defined height.

Parameter	Type	Required	Default	Description
input	source	yes	—	Name of input 2D vector map

Parameter	Type	Required	Default	Description
`where`	string	no	—	WHERE conditions of SQL statement without 'where' keyword
`type`	enum	no	[0, 1, 2]	Input feature type — options: 0=point, 1=line, 2=area
`zshift`	number	no	0.0	Shifting value for z coordinates
`height`	number	no	—	Fixed height for 3D vector objects
`height_column`	field	no	—	Name of attribute column with object heights
`elevation`	raster	no	—	Elevation raster for height extraction
`method`	enum	no	0	Sampling interpolation method — options: 0=nearest, 1=bilinear, 2=bicubic
`scale`	number	no	1.0	Scale factor sampled raster values
`null_value`	number	no	—	Height for sampled raster NULL values
`-t *(adv)*`	boolean	no	False	Trace elevation
`output`	vectorDestination	yes	—	3D Vector
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.extrude input=input.gpkg where=value type=0 zshift=0 height=10
 ↪  height_column=field1 elevation=dem.tif output=output.gpkg
```

This calls **v.extrude**: `input=input.gpkg` — Name of input 2D vector map; `where=value` — WHERE conditions of SQL statement without 'where' keyword; `type=0` selects *point*; `zshift=0` — Shifting value for z coordinates; `height=10` — Fixed height for 3D vector objects; `height_column=field1` — Name of attribute column with object heights; `elevation=dem.tif` — Elevation raster for height extraction; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.generalize` — v.generalize

group: Vector (v.*)

Vector based generalization.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input layer
`type`	enum	no	[0, 1, 2]	Input feature type — options: 0=line, 1=boundary, 2=area
`cats`	string	no	—	Category values
`where`	string	no	—	WHERE conditions of SQL statement without 'where' keyword

Parameter	Type	Required	Default	Description
`method`	enum	no	0	Generalization algorithm — options: 0=douglas, 1=douglas_reduction, 2=lang, 3=reduction, 4=reumann, 5=boyle, 6=sliding_averaging, 7=distance_weighting, 8=chaiken, 9=hermite, 10=snakes, 11=network, 12=displacement
`threshold`	number	no	1.0	Maximal tolerance value
`look_ahead`	number	no	7.0	Look-ahead parameter
`reduction`	number	no	50.0	Percentage of the points in the output of 'douglas_reduction' algorithm
`slide`	number	no	0.5	Slide of computed point toward the original point
`angle_thresh`	number	no	3.0	Minimum angle between two consecutive segments in Hermite method
`degree_thresh`	number	no	0.0	Degree threshold in network generalization
`closeness_thresh`	number	no	0.0	Closeness threshold in network generalization
`betweeness_thresh`	number	no	0.0	Betweenness threshold in network generalization
`alpha`	number	no	1.0	Snakes alpha parameter
`beta`	number	no	1.0	Snakes beta parameter
`iterations`	number	no	1.0	Number of iterations
`-t *(adv)*`	boolean	no	False	Do not copy attributes
`-l *(adv)*`	boolean	no	True	Disable loop support
`output`	vectorDestination	yes	—	Generalized
`error`	vectorDestination	yes	—	Errors
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector), `error` (outputVector)

Example usage

```
run grass:v.generalize input=input.gpkg type=0 cats=value where=value method=0 threshold=1
↪  look_ahead=7 output=output.gpkg
```

This calls **v.generalize**: `input=input.gpkg` — Input layer; `type=0` selects *line*; `cats=value` — Category values; `where=value` — WHERE conditions of SQL statement without 'where' keyword; `method=0` selects *douglas*; `threshold=1` — Maximal tolerance value; `look_ahead=7` — Look-ahead parameter; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.hull` — v.hull

group: Vector (v.*)

Produces a convex hull for a given vector map.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input layer

Parameter	Type	Required	Default	Description
where	string	no	—	WHERE conditions of SQL statement without 'where' keyword
-f	boolean	no	False	Create a 'flat' 2D hull even if the input is 3D points
output	vectorDestination	yes	—	Convex hull
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.hull input=input.gpkg where=value output=output.gpkg
```

This calls **v.hull**: input=input.gpkg — Input layer; where=value — WHERE conditions of SQL statement without 'where' keyword; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.in.ascii — v.in.ascii

group: Vector (v.*)

Creates a vector map from an ASCII points file or ASCII vector file.

Parameter	Type	Required	Default	Description
input	file	yes	—	ASCII file to be imported
format	enum	no	0	Input file format — options: 0=point, 1=standard
separator	string	no	pipe	Field separator
text	string	no	—	Text delimiter
skip	number	no	0.0	Number of header lines to skip at top of input file
columns	string	no	—	Column definition in SQL style (example: 'x double precision, y double precision, cat int, name varchar(10)')
x	number	no	1.0	Number of column used as x coordinate
y	number	no	2.0	Number of column used as y coordinate
z	number	no	0.0	Number of column used as z coordinate
cat	number	no	0.0	Number of column used as category
-z *(adv)*	boolean	no	False	Create 3D vector map
-n *(adv)*	boolean	no	False	Do not expect a header when reading in standard format
-t *(adv)*	boolean	no	False	Do not create table in points mode
-b *(adv)*	boolean	no	False	Do not build topology in points mode
-r *(adv)*	boolean	no	False	Only import points falling within current region (points mode)
-i *(adv)*	boolean	no	False	Ignore broken line(s) in points mode
output	vectorDestination	yes	—	ASCII
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.in.ascii input=input.dat format=0 separator=pipe text=value skip=0
↪  columns=value x=1 output=output.gpkg
```

This calls **v.in.ascii**: input=input.dat — ASCII file to be imported; format=0 selects *point*; separator=pipe — Field separator; text=value — Text delimiter; skip=0 — Number of header lines to skip at top of input file; columns=value — Column definition in SQL style (example: 'x double precision, y double...; x=1 — Number of column used as x coordinate; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.in.dxf — v.in.dxf

group: Vector (v.*)

Converts files in DXF format to GRASS vector map format.

Parameter	Type	Required	Default	Description
input	file	yes	—	Name of input DXF file
layers	string	no	—	List of layers to import
-e	boolean	no	True	Ignore the map extent of DXF file
-t	boolean	no	False	Do not create attribute tables
-f	boolean	no	True	Import polyface meshes as 3D wire frame
-l	boolean	no	False	List available layers and exit
-i	boolean	no	False	Invert selection by layers (don't import layers in list)
-1	boolean	no	True	Import all objects into one layer
output	vectorDestination	yes	—	Converted
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.in.dxf input=input.dat layers=value output=output.gpkg
```

This calls **v.in.dxf**: input=input.dat — Name of input DXF file; layers=value — List of layers to import; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.in.e00` — v.in.e00

group: Vector (v.*)

Imports E00 file into a vector map

Parameter	Type	Required	Default	Description
`input`	file	yes	—	Name of input E00 file
`type`	enum	no	[0]	Input feature type — options: 0=point, 1=line, 2=area
`output`	vectorDestination	yes	—	Name of output vector
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.in.e00 input=input.dat type=0 output=output.gpkg
```

This calls **v.in.e00**: `input=input.dat` — Name of input E00 file; `type=0` selects *point*; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.in.geonames` — v.in.geonames

group: Vector (v.*)

Imports geonames.org country files into a GRASS vector points map.

Parameter	Type	Required	Default	Description
`input`	file	yes	—	Uncompressed geonames file from (with .txt extension)
`output`	vectorDestination	yes	—	Geonames
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.in.geonames input=input.dat output=output.gpkg
```

This calls **v.in.geonames**: `input=input.dat` — Uncompressed geonames file from (with .txt extension); `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.in.lidar — v.in.lidar

group: Vector (v.*)

Converts LAS LiDAR point clouds to a GRASS vector map with libLAS.

Parameter	Type	Required	Default	Description
input	file	yes	—	LiDAR input files in LAS format (*.las or *.laz)
spatial	extent	no	—	Import subregion only
zrange	range	no	—	Filter range for z data
return_filter	enum	no	—	Only import points of selected return type — options: 0=first, 1=last, 2=mid
class_filter	string	no	—	Only import points of selected class(es) (comma separated integers)
skip	number	no	—	Do not import every n-th point
preserve	number	no	—	Import only every n-th point
offset	number	no	—	Skip first n points
limit	number	no	—	Import only n points
-t *(adv)*	boolean	no	False	Do not create attribute table
-c *(adv)*	boolean	no	False	Do not automatically add unique ID as category to each point
-b *(adv)*	boolean	no	False	Do not build topology
output	vectorDestination	yes	—	Lidar
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.in.lidar input=input.dat spatial=0,1000,0,1000 return_filter=0
 ↪  class_filter=value skip=10 preserve=10 output=output.gpkg
```

This calls **v.in.lidar**: input=input.dat — LiDAR input files in LAS format (*.las or *.laz); spatial=0,1000,0,1000 — Import subregion only; return_filter=0 selects *first*; class_filter=value — Only import points of selected class(es) (comma separated integers); skip=10 — Do not import every n-th point; preserve=10 — Import only every n-th point; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.in.lines — v.in.lines

group: Vector (v.*)

Import ASCII x,y[,z] coordinates as a series of lines.

Parameter	Type	Required	Default	Description
input	file	yes	—	ASCII file to be imported
separator	string	no	pipe	Field separator
-z *(adv)*	boolean	no	False	Create 3D vector map
output	vectorDestination	yes	—	Lines
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.in.lines input=input.dat separator=pipe output=output.gpkg
```

This calls **v.in.lines**: input=input.dat — ASCII file to be imported; separator=pipe — Field separator; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.in.mapgen — v.in.mapgen

group: Vector (v.*)

Imports Mapgen or Matlab-ASCII vector maps into GRASS.

Parameter	Type	Required	Default	Description
input	file	yes	—	Name of input file in Mapgen/Matlab format
-z *(adv)*	boolean	no	False	Create 3D vector map
-f *(adv)*	boolean	no	False	Input map is in Matlab format
output	vectorDestination	yes	—	Mapgen
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.in.mapgen input=input.dat output=output.gpkg
```

This calls **v.in.mapgen**: input=input.dat — Name of input file in Mapgen/Matlab format; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.in.wfs — v.in.wfs

group: Vector (v.*)

Import GetFeature from WFS

Parameter	Type	Required	Default	Description
url	string	no	http://	GetFeature URL starting with 'http'
srs	crs	no	—	Alternate spatial reference system

Parameter	Type	Required	Default	Description
`name`	string	no	—	Comma separated names of data layers to download
`maximum_features`	number	no	—	Maximum number of features to download
`start_index`	number	no	—	Skip earlier feature IDs and start downloading at this one
`output`	vectorDestination	yes	—	Converted
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.in.wfs url=http:// srs=EPSG:6346 name=value maximum_features=10 start_index=10
↪  output=output.gpkg
```

This calls **v.in.wfs**: `url=http://` — GetFeature URL starting with 'http'; `srs=EPSG:6346` — Alternate spatial reference system; `name=value` — Comma separated names of data layers to download; `maximum_features=10` — Maximum number of features to download; `start_index=10` — Skip earlier feature IDs and start downloading at this one; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.info` — v.info

group: Vector (v.*)

Outputs basic information about a user-specified vector map.

Parameter	Type	Required	Default	Description
`map`	source	yes	—	Name of input vector map
`-c`	boolean	no	False	Print types/names of table columns for specified layer instead of info
`-g`	boolean	no	False	Print map region only
`-e`	boolean	no	False	Print extended metadata info in shell script style
`-t`	boolean	no	False	Print topology information only
`html`	fileDestination	no	report.html	Information report
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `html` (outputHtml)

Example usage

```
run grass:v.info map=input.gpkg html=html.txt
```

This calls **v.info**: `map=input.gpkg` — Name of input vector map; `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.kcv — v.kcv

group: Vector (v.*)

Randomly partition points into test/train sets.

Parameter	Type	Required	Default	Description
map	source	yes	—	Input layer
npartitions	number	no	10.0	Number of partitions
column	string	no	part	Name for new column to which partition number is written
output	vectorDestination	yes	—	Partition
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.kcv map=input.gpkg npartitions=10 column=part output=output.gpkg
```

This calls **v.kcv**: map=input.gpkg — Input layer; npartitions=10 — Number of partitions; column=part — Name for new column to which partition number is written; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.kernel.rast — v.kernel.rast

group: Vector (v.*)

v.kernel.rast - Generates a raster density map from vector points map.

Parameter	Type	Required	Default	Description
input	source	yes	—	Name of input vector map with training points
radius	number	no	10.0	Kernel radius in map units
dsize	number	no	0.0	Discretization error in map units
segmax	number	no	100.0	Maximum length of segment on network
distmax	number	no	100.0	Maximum distance from point to network
multiplier	number	no	1.0	Multiply the density result by this number
node	enum	no	0	Node method — options: 0=none, 1=split
kernel	enum	no	5	Kernel function — options: 0=uniform, 1=triangular, 2=epanechnikov, 3=quartic, 4=triweight, 5=gaussian, 6=cosine
-o *(adv)*	boolean	no	False	Try to calculate an optimal radius with given 'radius' taken as maximum (experimental)
output	rasterDestination	yes	—	Kernel
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)

Parameter	Type	Required	Default	Description
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputRaster)

Example usage

```
run grass:v.kernel.rast input=input.gpkg radius=10 dsize=0 segmax=100 distmax=100
 ↪  multiplier=1 node=0 output=output.tif
```

This calls **v.kernel.rast**: `input=input.gpkg` — Name of input vector map with training points; `radius=10` — Kernel radius in map units; `dsize=0` — Discretization error in map units; `segmax=100` — Maximum length of segment on network; `distmax=100` — Maximum distance from point to network; `multiplier=1` — Multiply the density result by this number; `node=0` selects *none*; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.kernel.vector` — v.kernel.vector

group: Vector (v.*)

v.kernel.vector - Generates a vector density map from vector points on a vector network.

Parameter	Type	Required	Default	Description
input	source	yes	—	Name of input vector map with training points
net	source	yes	—	Name of input network vector map
radius	number	no	10.0	Kernel radius in map units
dsize	number	no	0.0	Discretization error in map units
segmax	number	no	100.0	Maximum length of segment on network
distmax	number	no	100.0	Maximum distance from point to network
multiplier	number	no	1.0	Multiply the density result by this number
node	enum	no	0	Node method — options: 0=none, 1=split
kernel	enum	no	5	Kernel function — options: 0=uniform, 1=triangular, 2=epanechnikov, 3=quartic, 4=triweight, 5=gaussian, 6=cosine
-o *(adv)*	boolean	no	False	Try to calculate an optimal radius with given 'radius' taken as maximum (experimental)
-n *(adv)*	boolean	no	False	Normalize values by sum of density multiplied by length of each segment.
-m *(adv)*	boolean	no	False	Multiply the result by number of input points
output	vectorDestination	yes	—	Kernel
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.kernel.vector input=input.gpkg net=input.gpkg radius=10 dsize=0 segmax=100
↪  distmax=100 multiplier=1 output=output.gpkg
```

This calls **v.kernel.vector**: `input=input.gpkg` — Name of input vector map with training points; `net=input.gpkg` — Name of input network vector map; `radius=10` — Kernel radius in map units; `dsize=0` — Discretization error in map units; `segmax=100` — Maximum length of segment on network; `distmax=100` — Maximum distance from point to network; `multiplier=1` — Multiply the density result by this number; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.lidar.correction` — v.lidar.correction

group: Vector (v.*)

Correction of the v.lidar.growing output. It is the last of the three algorithms for LIDAR filtering.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector layer (v.lidar.growing output)
`ew_step`	number	no	25.0	Length of each spline step in the east-west direction
`ns_step`	number	no	25.0	Length of each spline step in the north-south direction
`lambda_c`	number	no	1.0	Regularization weight in reclassification evaluation
`tch`	number	no	2.0	High threshold for object to terrain reclassification
`tcl`	number	no	1.0	Low threshold for terrain to object reclassification
`-e`	boolean	no	False	Estimate point density and distance
`output`	vectorDestination	yes	—	Classified
`terrain`	vectorDestination	yes	—	Only 'terrain' points
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector), `terrain` (outputVector)

Example usage

```
run grass:v.lidar.correction input=input.gpkg ew_step=25 ns_step=25 lambda_c=1 tch=2 tcl=1
↪  output=output.gpkg
```

This calls **v.lidar.correction**: `input=input.gpkg` — Input vector layer (v.lidar.growing output); `ew_step=25` — Length of each spline step in the east-west direction; `ns_step=25` — Length of each spline step in the north-south direction; `lambda_c=1` — Regularization weight in reclassification evaluation; `tch=2` — High threshold for object to terrain reclassification; `tcl=1` — Low threshold for terrain to object reclassification; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.lidar.edgedetection — v.lidar.edgedetection

group: Vector (v.*)

Detects the object's edges from a LIDAR data set.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector layer
ew_step	number	no	4.0	Length of each spline step in the east-west direction
ns_step	number	no	4.0	Length of each spline step in the north-south direction
lambda_g	number	no	0.01	Regularization weight in gradient evaluation
tgh	number	no	6.0	High gradient threshold for edge classification
tgl	number	no	3.0	Low gradient threshold for edge classification
theta_g	number	no	0.26	Angle range for same direction detection
lambda_r	number	no	2.0	Regularization weight in residual evaluation
-e	boolean	no	False	Estimate point density and distance
output	vectorDestination	yes	—	Edges
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.lidar.edgedetection input=input.gpkg ew_step=4 ns_step=4 lambda_g=0.01 tgh=6
 ↪  tgl=3 theta_g=0.26 output=output.gpkg
```

This calls **v.lidar.edgedetection**: input=input.gpkg — Input vector layer; ew_step=4 — Length of each spline step in the east-west direction; ns_step=4 — Length of each spline step in the north-south direction; lambda_g=0.01 — Regularization weight in gradient evaluation; tgh=6 — High gradient threshold for edge classification; tgl=3 — Low gradient threshold for edge classification; theta_g=0.26 — Angle range for same direction detection; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.lidar.growing — v.lidar.growing

group: Vector (v.*)

Building contour determination and Region Growing algorithm for determining the building inside

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector (v.lidar.edgedetection output)
first	source	yes	—	First pulse vector layer
tj	number	no	0.2	Threshold for cell object frequency in region growing
td	number	no	0.6	Threshold for double pulse in region growing

Parameter	Type	Required	Default	Description
output	vectorDestination	yes	—	Buildings
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.lidar.growing input=input.gpkg first=input.gpkg tj=0.2 td=0.6
 ↪  output=output.gpkg
```

This calls **v.lidar.growing**: input=input.gpkg — Input vector (v.lidar.edgedetection output); first=input.gpkg — First pulse vector layer; tj=0.2 — Threshold for cell object frequency in region growing; td=0.6 — Threshold for double pulse in region growing; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.mkgrid — v.mkgrid

group: Vector (v.*)

Creates a GRASS vector layer of a user-defined grid.

Parameter	Type	Required	Default	Description
grid	string	no	10,10	Number of rows and columns in grid
position	enum	no	0	Where to place the grid — options: 0=coor
coordinates	point	no	—	Lower left easting and northing coordinates of map
box	string	yes	—	Width and height of boxes in grid
angle	number	no	0.0	Angle of rotation (in degrees counterclockwise)
breaks	number	no	0.0	Number of vertex points per grid cell
type	enum	no	2	Output feature type — options: 0=point, 1=line, 2=area
-h	boolean	no	False	Create hexagons (default: rectangles)
-a	boolean	no	False	Allow asymmetric hexagons
map	vectorDestination	yes	—	Grid
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: map (outputVector)

Example usage

```
run grass:v.mkgrid box=value grid=10,10 position=0 coordinates=100,100 angle=0 breaks=0
↪  type=2 map=map.gpkg
```

This calls **v.mkgrid**: `box=value` — Width and height of boxes in grid; `grid=10,10` — Number of rows and columns in grid; `position=0` selects *coor*; `coordinates=100,100` — Lower left easting and northing coordinates of map; `angle=0` — Angle of rotation (in degrees counter-clockwise); `breaks=0` — Number of vertex points per grid cell; `type=2` selects *area*; `map=map.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.neighbors` — v.neighbors

group: Vector (v.*)

Makes each cell value a function of attribute values and stores in an output raster map.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector layer
`method`	enum	no	0	Method for aggregate statistics (count if non given) — options: 0=count, 1=sum, 2=average, 3=median, 4=mode, 5=minimum, 6=maximum, 7=range, 8=stddev, 9=variance, 10=diversity
`points_column`	field	no	—	Column name of points map to use for statistics
`size`	number	no	0.1	Neighborhood diameter in map units
`output`	rasterDestination	yes	—	Neighborhood
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputRaster)

Example usage

```
run grass:v.neighbors input=input.gpkg method=0 points_column=field1 size=0.1
↪  output=output.tif
```

This calls **v.neighbors**: `input=input.gpkg` — Input vector layer; `method=0` selects *count*; `points_column=field1` — Column name of points map to use for statistics; `size=0.1` — Neighborhood diameter in map units; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.net` — v.net

group: Vector (v.*)

Performs network maintenance

Parameter	Type	Required	Default	Description
`input`	source	no	—	Input vector line layer (arcs)
`points`	source	no	—	Input vector point layer (nodes)
`file`	file	no	—	Name of input arcs file

Parameter	Type	Required	Default	Description
operation	enum	no	0	Operation to be performed — options: 0=nodes, 1=connect, 2=arcs
threshold	number	no	50.0	Threshold for connecting centers to the network (in map unit)
arc_type	enum	no	[0, 1]	Arc type — options: 0=line, 1=boundary
-s *(adv)*	boolean	no	False	Snap points to network
-c *(adv)*	boolean	no	False	Assign unique categories to new points
output	vectorDestination	yes	—	Network
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net input=input.gpkg points=input.gpkg file=input.dat operation=0 threshold=50
↪  arc_type=0 output=output.gpkg
```

This calls **v.net**: input=input.gpkg — Input vector line layer (arcs); points=input.gpkg — Input vector point layer (nodes); file=input.dat — Name of input arcs file; operation=0 selects *nodes*; threshold=50 — Threshold for connecting centers to the network (in map unit); arc_type=0 selects *line*; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.alloc — v.net.alloc

group: Vector (v.*)

Allocates subnets for nearest centers

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (arcs)
points	source	yes	—	Centers point layer (nodes)
threshold	number	no	50.0	Threshold for connecting centers to the network (in map unit)
center_cats *(adv)*	string	no	1-100000	Category values
arc_type *(adv)*	enum	no	[0, 1]	Arc type — options: 0=line, 1=boundary
method *(adv)*	enum	no	0	Use costs from centers or costs to centers — options: 0=from, 1=to
arc_column *(adv)*	field	no	—	Arc forward/both direction(s) cost column (number)
arc_backward_column *(adv)*	field	no	—	Arc backward direction cost column (number)
node_column *(adv)*	field	no	—	Node cost column (number)
-g *(adv)*	boolean	no	False	Use geodesic calculation for longitude-latitude locations
output	vectorDestination	yes	—	Network Alloction
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.alloc input=input.gpkg points=input.gpkg threshold=50 output=output.gpkg
```

This calls **v.net.alloc**: input=input.gpkg — Input vector line layer (arcs); points=input.gpkg — Centers point layer (nodes); threshold=50 — Threshold for connecting centers to the network (in map unit); output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.allpairs — v.net.allpairs

group: Vector (v.*)

Computes the shortest path between all pairs of nodes in the network

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (arcs)
points	source	yes	—	Centers point layer (nodes)
threshold	number	no	50.0	Threshold for connecting centers to the network (in map unit)
cats *(adv)*	string	no	1-10000	Category values
where *(adv)*	string	no	—	WHERE condition of SQL statement without 'where' keyword'
arc_column *(adv)*	field	no	—	Arc forward/both direction(s) cost column (number)
arc_backward_column *(adv)*	field	no	—	Arc backward direction cost column (number)
node_column *(adv)*	field	no	—	Node cost column (number)
-g *(adv)*	boolean	no	False	Use geodesic calculation for longitude-latitude locations
output	vectorDestination	yes	—	Network_Allpairs
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.allpairs input=input.gpkg points=input.gpkg threshold=50
 ↪  output=output.gpkg
```

This calls **v.net.allpairs**: `input=input.gpkg` — Input vector line layer (arcs); `points=input.gpkg` — Centers point layer (nodes); `threshold=50` — Threshold for connecting centers to the network (in map unit); `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.net.bridge` — v.net.bridge

group: Vector (v.*)

Computes bridges and articulation points in the network.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector line layer (network)
`points`	source	no	—	Centers point layer (nodes)
`method`	enum	no	0	Feature type — options: 0=bridge, 1=articulation
`threshold`	number	no	50.0	Threshold for connecting centers to the network (in map unit)
`arc_column *(adv)*`	field	no	—	Arc forward/both direction(s) cost column (name)
`arc_backward_column *(adv)*`	field	no	—	Arc backward direction cost column (name)
`node_column *(adv)*`	field	no	—	Node cost column (number)
`output`	vectorDestination	yes	—	Bridge
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.net.bridge input=input.gpkg points=input.gpkg method=0 threshold=50
 ↪  output=output.gpkg
```

This calls **v.net.bridge**: `input=input.gpkg` — Input vector line layer (network); `points=input.gpkg` — Centers point layer (nodes); `method=0` selects *bridge*; `threshold=50` — Threshold for connecting centers to the network (in map unit); `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.net.centrality` — v.net.centrality

group: Vector (v.*)

Computes degree, centrality, betweenness, closeness and eigenvector centrality measures in the network.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (network)
degree	string	no	degree	Name of output degree centrality column
closeness	string	no	closeness	Name of output closeness centrality column
betweenness	string	no	betweenness	Name of output betweenness centrality column
eigenvector	string	no	eigenvector	Name of output eigenvector centrality column
iterations *(adv)*	number	no	1000.0	Maximum number of iterations to compute eigenvector centrality
error *(adv)*	number	no	0.1	Cumulative error tolerance for eigenvector centrality
cats *(adv)*	string	no	—	Category values
where *(adv)*	string	no	—	WHERE conditions of SQL statement without 'where' keyword
arc_column *(adv)*	field	no	—	Arc forward/both direction(s) cost column (number)
arc_backward_column *(adv)*	field	no	—	Arc backward direction cost column (number)
node_column *(adv)*	field	no	—	Node cost column (number)
-a *(adv)*	boolean	no	True	Add points on nodes
-g *(adv)*	boolean	no	False	Use geodesic calculation for longitude-latitude locations
output	vectorDestination	yes	—	Network Centrality
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.net.centrality input=input.gpkg degree=degree closeness=closeness
 ↪  betweenness=betweenness eigenvector=eigenvector output=output.gpkg
```

This calls **v.net.centrality**: `input=input.gpkg` — Input vector line layer (network); `degree=degree` — Name of output degree centrality column; `closeness=closeness` — Name of output closeness centrality column; `betweenness=betweenness` — Name of output betweenness centrality column; `eigenvector=eigenvector` — Name of output eigenvector centrality column; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.net.components` — v.net.components

group: Vector (v.*)

Computes strongly and weakly connected components in the network.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (network)
points	source	no	—	Centers point layer (nodes)

Parameter	Type	Required	Default	Description
`threshold`	number	no	50.0	Threshold for connecting centers to the network (in map unit)
`method`	enum	no	0	Type of components — options: 0=weak, 1=strong
`arc_column *(adv)*`	field	no	—	Arc forward/both direction(s) cost column (number)
`arc_backward_column *(adv)*`	field	no	—	Arc backward direction cost column (number)
`node_column *(adv)*`	field	no	—	Node cost column (number)
`-a *(adv)*`	boolean	no	True	Add points on nodes
`output`	vectorDestination	yes	—	Network_Components_Line
`output_point`	vectorDestination	yes	—	Network_Components_Point
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector), `output_point` (outputVector)

Example usage

```
run grass:v.net.components input=input.gpkg points=input.gpkg threshold=50 method=0
 ↪  output=output.gpkg
```

This calls **v.net.components**: `input=input.gpkg` — Input vector line layer (network); `points=input.gpkg` — Centers point layer (nodes); `threshold=50` — Threshold for connecting centers to the network (in map unit); `method=0` selects *weak*; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.net.connectivity` — v.net.connectivity

group: Vector (v.*)

Computes vertex connectivity between two sets of nodes in the network.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector line layer (network)
`points`	source	yes	—	Input vector point layer (first set of nodes)
`threshold`	number	no	50.0	Threshold for connecting centers to the network (in map unit)
`set1_cats`	string	no	—	Set1 Category values
`set1_where`	string	no	—	Set1 WHERE conditions of SQL statement without 'where' keyword
`set2_cats`	string	no	—	Set2 Category values
`set2_where`	string	no	—	Set2 WHERE conditions of SQL statement without 'where' keyword
`arc_column *(adv)*`	field	no	—	Arc forward/both direction(s) cost column (number)
`arc_backward_column *(adv)*`	field	no	—	Arc backward direction cost column (number)
`node_column *(adv)*`	field	no	—	Node cost column (number)

Parameter	Type	Required	Default	Description
output	vectorDestination	yes	—	Network_Connectivity
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.connectivity input=input.gpkg points=input.gpkg threshold=50
↪  set1_cats=value set1_where=value set2_cats=value set2_where=value output=output.gpkg
```

This calls **v.net.connectivity**: input=input.gpkg — Input vector line layer (network); points=input.gpkg — Input vector point layer (first set of nodes); threshold=50 — Threshold for connecting centers to the network (in map unit); set1_cats=value — Set1 Category values; set1_where=value — Set1 WHERE conditions of SQL statement without 'where' keyword; set2_cats=value — Set2 Category values; set2_where=value — Set2 WHERE conditions of SQL statement without 'where' keyword; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.distance — v.net.distance

group: Vector (v.*)

Computes shortest distance via the network between the given sets of features.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (network)
flayer	source	yes	—	Input vector from points layer (from)
tlayer	source	yes	—	Input vector to layer (to)
threshold	number	no	50.0	Threshold for connecting nodes to the network (in map unit)
arc_type *(adv)*	enum	no	[0, 1]	Arc type — options: 0=line, 1=boundary
from_cats *(adv)*	string	no	—	From Category values
from_where *(adv)*	string	no	—	From WHERE conditions of SQL statement without 'where' keyword
to_type *(adv)*	enum	no	[0]	To feature type — options: 0=point, 1=line, 2=boundary
to_cats *(adv)*	string	no	—	To Category values
to_where *(adv)*	string	no	—	To WHERE conditions of SQL statement without 'where' keyword
arc_column *(adv)*	field	no	—	Arc forward/both direction(s) cost column (number)
arc_backward_column *(adv)*	field	no	—	Arc backward direction cost column (number)
node_column *(adv)*	field	no	—	Node cost column (number)
-g *(adv)*	boolean	no	False	Use geodesic calculation for longitude-latitude locations
-l *(adv)*	boolean	no	False	Write each output path as one line, not as original input segments
output	vectorDestination	yes	—	Network_Distance

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.distance input=input.gpkg flayer=input.gpkg tlayer=input.gpkg threshold=50
↪  output=output.gpkg
```

This calls **v.net.distance**: input=input.gpkg — Input vector line layer (network); flayer=input.gpkg — Input vector from points layer (from); tlayer=input.gpkg — Input vector to layer (to); threshold=50 — Threshold for connecting nodes to the network (in map unit); output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.flow — v.net.flow

group: Vector (v.*)

Computes the maximum flow between two sets of nodes in the network.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (network)
points	source	yes	—	Input vector point layer (flow nodes)
threshold	number	no	50.0	Threshold for connecting centers to the network (in map unit)
source_cats	string	no	—	Source Category values
source_where	string	no	—	Source WHERE conditions of SQL statement without 'where' keyword
sink_cats	string	no	—	Sink Category values
sink_where	string	no	—	Sink WHERE conditions of SQL statement without 'where' keyword
arc_column *(adv)*	field	no	—	Arc forward/both direction(s) cost column (number)
arc_backward_column *(adv)*	field	no	—	Arc backward direction cost column (number)
node_column *(adv)*	field	no	—	Node cost column (number)
output	vectorDestination	yes	—	Network_Flow
cut	vectorDestination	yes	—	Network_Cut
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)

Parameter	Type	Required	Default	Description
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector), cut (outputVector)

Example usage

```
run grass:v.net.flow input=input.gpkg points=input.gpkg threshold=50 source_cats=value
↪  source_where=value sink_cats=value sink_where=value output=output.gpkg
```

This calls **v.net.flow**: input=input.gpkg — Input vector line layer (network); points=input.gpkg — Input vector point layer (flow nodes); threshold=50 — Threshold for connecting centers to the network (in map unit); source_cats=value — Source Category values; source_where=value — Source WHERE conditions of SQL statement without 'where' keyword; sink_cats=value — Sink Category values; sink_where=value — Sink WHERE conditions of SQL statement without 'where' keyword; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.iso — v.net.iso

group: Vector (v.*)

Splits network by cost isolines.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (arcs)
points	source	yes	—	Centers point layer (nodes)
threshold	number	no	50.0	Threshold for connecting centers to the network (in map unit)
arc_type *(adv)*	enum	no	[0, 1]	Arc type — options: 0=line, 1=boundary
center_cats *(adv)*	string	no	1-100000	Category values
costs	string	no	1000,2000,3000	Costs for isolines
arc_column *(adv)*	field	no	—	Arc forward/both direction(s) cost column (number)
arc_backward_column *(adv)*	field	no	—	Arc backward direction cost column (number)
node_column *(adv)*	field	no	—	Node cost column (number)
-g *(adv)*	boolean	no	False	Use geodesic calculation for longitude-latitude locations
output	vectorDestination	yes	—	Network_Iso
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.iso input=input.gpkg points=input.gpkg threshold=50 costs=1000,2000,3000
↪  output=output.gpkg
```

This calls **v.net.iso**: `input=input.gpkg` — Input vector line layer (arcs); `points=input.gpkg` — Centers point layer (nodes); `threshold=50` — Threshold for connecting centers to the network (in map unit); `costs=1000,2000,3000` — Costs for isolines; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.net.nreport` — v.net.nreport

group: Vector (v.*)

v.net.nreport - Reports nodes information of a network

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector line layer (arcs)
`output`	fileDestination	yes	—	NReport
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputHtml)

Example usage

```
run grass:v.net.nreport input=input.gpkg output=output.txt
```

This calls **v.net.nreport**: `input=input.gpkg` — Input vector line layer (arcs); `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.net.path` — v.net.path

group: Vector (v.*)

Finds shortest path on vector network

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector line layer (arcs)
`points`	source	yes	—	Centers point layer (nodes)
`file`	file	yes	—	Name of file containing start and end points
`threshold`	number	no	50.0	Threshold for connecting centers to the network (in map unit)
`arc_type *(adv)*`	enum	no	[0, 1]	Arc type — options: 0=line, 1=boundary
`arc_column *(adv)*`	field	no	—	Arc forward/both direction(s) cost column (number)
`arc_backward_column *(adv)*`	field	no	—	Arc backward direction cost column (number)
`node_column *(adv)*`	field	no	—	Node cost column (number)
`dmax *(adv)*`	number	no	1000.0	Maximum distance to the network
`-g *(adv)*`	boolean	no	False	Use geodesic calculation for longitude-latitude locations
`-s *(adv)*`	boolean	no	False	Write output as original input segments, not each path as one line
`output`	vectorDestination	yes	—	Network_Path
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)

Parameter	Type	Required	Default	Description
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.path input=input.gpkg points=input.gpkg file=input.dat threshold=50
↪  output=output.gpkg
```

This calls **v.net.path**: input=input.gpkg — Input vector line layer (arcs); points=input.gpkg — Centers point layer (nodes); file=input.dat — Name of file containing start and end points; threshold=50 — Threshold for connecting centers to the network (in map unit); output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.report — v.net.report

group: Vector (v.*)

v.net.report - Reports lines information of a network

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (arcs)
html	fileDestination	yes	—	Report
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: html (outputHtml)

Example usage

```
run grass:v.net.report input=input.gpkg html=html.txt
```

This calls **v.net.report**: input=input.gpkg — Input vector line layer (arcs); html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.salesman — v.net.salesman

group: Vector (v.*)

Creates a cycle connecting given nodes (Traveling salesman problem)

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (arcs)
points	source	yes	—	Centers point layer (nodes)
threshold	number	no	50.0	Threshold for connecting centers to the network (in map unit)
arc_type *(adv)*	enum	no	[0, 1]	Arc type — options: 0=line, 1=boundary

Parameter	Type	Required	Default	Description
`center_cats *(adv)*`	string	no	1-100000	Category values
`arc_column *(adv)*`	field	no	—	Arc forward/both direction(s) cost column (number)
`arc_backward_column *(adv)*`	field	no	—	Arc backward direction cost column (number)
`-g *(adv)*`	boolean	no	False	Use geodesic calculation for longitude-latitude locations
`output`	vectorDestination	yes	—	Network_Salesman
`sequence`	fileDestination	no	—	Output file holding node sequence
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector), `sequence` (outputFile)

Example usage

```
run grass:v.net.salesman input=input.gpkg points=input.gpkg threshold=50
 ↪  output=output.gpkg
```

This calls **v.net.salesman**: `input=input.gpkg` — Input vector line layer (arcs); `points=input.gpkg` — Centers point layer (nodes); `threshold=50` — Threshold for connecting centers to the network (in map unit); `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.net.spanningtree` — v.net.spanningtree

group: Vector (v.*)

Computes minimum spanning tree for the network.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector line layer (arcs)
`points`	source	no	—	Input point layer (nodes)
`threshold`	number	no	50.0	Threshold for connecting centers to the network (in map unit)
`arc_column *(adv)*`	field	no	—	Arc forward/both direction(s) cost column (number)
`node_column *(adv)*`	field	no	—	Node cost column (number)
`-g *(adv)*`	boolean	no	False	Use geodesic calculation for longitude-latitude locations
`output`	vectorDestination	yes	—	SpanningTree
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)

Parameter	Type	Required	Default	Description
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.spanningtree input=input.gpkg points=input.gpkg threshold=50
↪  output=output.gpkg
```

This calls **v.net.spanningtree**: input=input.gpkg — Input vector line layer (arcs); points=input.gpkg — Input point layer (nodes); threshold=50 — Threshold for connecting centers to the network (in map unit); output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.steiner — v.net.steiner

group: Vector (v.*)

Creates Steiner tree for the network and given terminals

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (arcs)
points	source	yes	—	Centers point layer (nodes)
threshold	number	no	50.0	Threshold for connecting centers to the network (in map unit)
arc_type *(adv)*	enum	no	[0, 1]	Arc type — options: 0=line, 1=boundary
terminal_cats *(adv)*	string	no	1-100000	Category values
acolumn *(adv)*	field	no	—	Arc forward/both direction(s) cost column (number)
npoints *(adv)*	number	no	-1.0	Number of Steiner points
-g *(adv)*	boolean	no	False	Use geodesic calculation for longitude-latitude locations
output	vectorDestination	yes	—	Network Steiner
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.steiner input=input.gpkg points=input.gpkg threshold=50 output=output.gpkg
```

This calls **v.net.steiner**: input=input.gpkg — Input vector line layer (arcs); points=input.gpkg — Centers point layer (nodes); threshold=50 — Threshold for connecting centers to the network (in map unit); output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.timetable — v.net.timetable

group: Vector (v.*)

Finds shortest path using timetables.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (arcs)
points	source	yes	—	Centers point layer (nodes)
walk_layer	source	no	—	Layer number or name with walking connections
threshold	number	no	50.0	Threshold for connecting centers to the network (in map unit)
arc_column *(adv)*	field	no	—	Arc forward/both direction(s) cost column (number)
arc_backward_column *(adv)*	field	no	—	Arc backward direction cost column (number)
node_column *(adv)*	field	no	—	Node cost column (number)
route_id *(adv)*	field	no	—	Name of column with route ids
stop_time *(adv)*	field	no	—	Name of column with stop timestamps
to_stop *(adv)*	field	no	—	Name of column with stop ids
walk_length *(adv)*	field	no	—	Name of column with walk lengths
output	vectorDestination	yes	—	Network Timetable
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.timetable input=input.gpkg points=input.gpkg walk_layer=input.gpkg
 ↪  threshold=50 output=output.gpkg
```

This calls **v.net.timetable**: input=input.gpkg — Input vector line layer (arcs); points=input.gpkg — Centers point layer (nodes); walk_layer=input.gpkg — Layer number or name with walking connections; threshold=50 — Threshold for connecting centers to the network (in map unit); output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.net.visibility — v.net.visibility

group: Vector (v.*)

Performs visibility graph construction.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector line layer (arcs)
coordinates	string	no	—	Coordinates
visibility	source	no	—	Input vector line layer containing visible points
output	vectorDestination	yes	—	Network Visibility

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.net.visibility input=input.gpkg coordinates=value visibility=input.gpkg
↪  output=output.gpkg
```

This calls **v.net.visibility**: input=input.gpkg — Input vector line layer (arcs); coordinates=value — Coordinates; visibility=input.gpkg — Input vector line layer containing visible points; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.normal — v.normal

group: Vector (v.*)

Tests for normality for points.

Parameter	Type	Required	Default	Description
map	source	yes	—	point vector defining sample points
tests	string	no	1-3	Lists of tests (1-15): e.g. 1,3-8,13
column	field	yes	—	Attribute column
-r	boolean	no	True	Use only points in current region
-l	boolean	no	False	lognormal
html	fileDestination	no	report.html	Normality
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: html (outputHtml)

Example usage

```
run grass:v.normal map=input.gpkg column=field1 tests=1-3 html=html.txt
```

This calls **v.normal**: map=input.gpkg — point vector defining sample points; column=field1 — Attribute column; tests=1-3 — Lists of tests (1-15): e.g. 1,3-8,13; html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.out.ascii — v.out.ascii

group: Vector (v.*)

Exports a vector map to a GRASS ASCII vector representation.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Name of input vector map
`type`	enum	no	[0, 1, 2, 3, 4, 5, 6]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area, 5=face, 6=kernel
`columns`	field	no	—	Name of attribute column(s) to be exported
`where`	string	no	—	WHERE conditions of SQL statement without 'where' keyword
`format`	enum	no	0	Output format — options: 0=point, 1=standard, 2=wkt
`separator`	enum	no	0	Field separator — options: 0=pipe, 1=comma, 2=space, 3=tab, 4=newline
`precision`	number	no	8.0	Number of significant digits (floating point only)
`-o *(adv)*`	boolean	no	False	Create old (version 4) ASCII file
`-c *(adv)*`	boolean	no	False	Include column names in output (points mode)
`output`	fileDestination	yes	—	Name for output ASCII file or ASCII vector name if '-o' is defined
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputFile)

Example usage

```
run grass:v.out.ascii input=input.gpkg type=0 columns=field1 where=value format=0
↪  separator=0 precision=8 output=output.txt
```

This calls **v.out.ascii**: `input=input.gpkg` — Name of input vector map; `type=0` selects *point*; `columns=field1` — Name of attribute column(s) to be exported; `where=value` — WHERE conditions of SQL statement without 'where' keyword; `format=0` selects *point*; `separator=0` selects *pipe*; `precision=8` — Number of significant digits (floating point only); `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.out.dxf` — v.out.dxf

group: Vector (v.*)

Exports GRASS vector map layers to DXF file format.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Name of input vector map
`output`	fileDestination	yes	—	DXF vector
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputFile)

Example usage

```
run grass:v.out.dxf input=input.gpkg output=output.txt
```

This calls **v.out.dxf**: `input=input.gpkg` — Name of input vector map; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.out.postgis` — v.out.postgis

group: Vector (v.*)

Exports a vector map layer to PostGIS feature table.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Name of input vector map
`type`	enum	no	[7]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area, 5=face, 6=kernel, 7=auto
`output`	string	no	PG:dbname=grass	Name for output PostGIS datasource
`output_layer`	string	no	—	Name for output PostGIS layer
`options`	string	no	—	Creation options
`-t *(adv)*`	boolean	no	False	Do not export attribute table
`-l *(adv)*`	boolean	no	False	Export PostGIS topology instead of simple features
`-2 *(adv)*`	boolean	no	False	Force 2D output even if input is 3D
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Example usage

```
run grass:v.out.postgis input=input.gpkg type=0 output=PG:dbname=grass output_layer=value
↪  options=value
```

This calls **v.out.postgis**: `input=input.gpkg` — Name of input vector map; `type=0` selects *point*; `output=PG:dbname=grass` — Name for output PostGIS datasource; `output_layer=value` — Name for output PostGIS layer; `options=value` — Creation options. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.out.pov` — v.out.pov

group: Vector (v.*)

Converts to POV-Ray format, GRASS x,y,z -> POV-Ray x,z,y

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Name of input vector map
`type`	enum	no	[0, 1, 4, 5]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area, 5=face, 6=kernel
`size`	number	no	10.0	Radius of sphere for points and tube for lines
`zmod`	string	no	—	Modifier for z coordinates, this string is appended to each z coordinate
`objmod`	string	no	—	Object modifier (OBJECT_MODIFIER in POV-Ray documentation)
`output`	fileDestination	yes	—	POV vector
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputFile)

Example usage

```
run grass:v.out.pov input=input.gpkg type=0 size=10 zmod=value objmod=value
↪  output=output.txt
```

This calls **v.out.pov**: `input=input.gpkg` — Name of input vector map; `type=0` selects *point*; `size=10` — Radius of sphere for points and tube for lines; `zmod=value` — Modifier for z coordinates, this string is appended to each z coordinate; `objmod=value` — Object modifier (OBJECT_MODIFIER in POV-Ray documentation); `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.out.svg` — v.out.svg

group: Vector (v.*)

Exports a vector map to SVG file.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Name of input vector map
`type`	enum	no	0	Output type — options: 0=poly, 1=line, 2=point
`precision`	number	no	6.0	Coordinate precision
`attribute`	field	no	—	Attribute(s) to include in output SVG
`output`	fileDestination	yes	—	SVG File
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputFile)

Example usage

```
run grass:v.out.svg input=input.gpkg type=0 precision=6 attribute=field1 output=output.txt
```

This calls **v.out.svg**: `input=input.gpkg` — Name of input vector map; `type=0` selects *poly*; `precision=6` — Coordinate precision; `attribute=field1` — Attribute(s) to include in output SVG; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.out.vtk` — v.out.vtk

group: Vector (v.*)

Converts a vector map to VTK ASCII output.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Name of input vector map
`type`	enum	no	[0, 1, 2, 3, 4, 5, 6]	Input feature type — options: 0=point, 1=kernel, 2=centroid, 3=line, 4=boundary, 5=area, 6=face
`precision`	number	no	2.0	Number of significant digits (floating point only)
`zscale`	number	no	1.0	Scale factor for elevation
`-c *(adv)*`	boolean	no	False	Correct the coordinates to fit the VTK-OpenGL precision

Parameter	Type	Required	Default	Description
`-n *(adv)*`	boolean	no	False	Export numeric attribute table fields as VTK scalar variables
`output`	fileDestination	yes	—	VTK File
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputFile)

Example usage

```
run grass:v.out.vtk input=input.gpkg type=0 precision=2 zscale=1 output=output.txt
```

This calls **v.out.vtk**: `input=input.gpkg` — Name of input vector map; `type=0` selects *point*; `precision=2` — Number of significant digits (floating point only); `zscale=1` — Scale factor for elevation; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.outlier` — v.outlier

group: Vector (v.*)

Removes outliers from vector point data.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector layer
`ew_step`	number	no	10.0	Interpolation spline step value in east direction
`ns_step`	number	no	10.0	Interpolation spline step value in north direction
`lambda`	number	no	0.1	Tykhonov regularization weight
`threshold`	number	no	50.0	Threshold for the outliers
`filter`	enum	no	0	Filtering option — options: 0=both, 1=positive, 2=negative
`-e`	boolean	no	False	Estimate point density and distance
`output`	vectorDestination	yes	—	Layer without outliers
`outlier`	vectorDestination	yes	—	Outliers
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector), `outlier` (outputVector)

Example usage

```
run grass:v.outlier input=input.gpkg ew_step=10 ns_step=10 lambda=0.1 threshold=50
 ↪  filter=0 output=output.gpkg
```

This calls **v.outlier**: `input=input.gpkg` — Input vector layer; `ew_step=10` — Interpolation spline step value in east direction; `ns_step=10` — Interpolation spline step value in north direction; `lambda=0.1` — Tykhonov regularization weight; `threshold=50` — Threshold for the outliers; `filter=0` selects *both*; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.overlay` — v.overlay

group: Vector (v.*)

Overlays two vector maps.

Parameter	Type	Required	Default	Description
`ainput`	source	yes	—	Input layer (A)
`atype`	enum	no	0	Input layer (A) Type — options: 0=area, 1=line, 2=auto
`binput`	source	yes	—	Input layer (B)
`btype`	enum	no	0	Input layer (B) Type — options: 0=area
`operator`	enum	no	0	Operator to use — options: 0=and, 1=or, 2=not, 3=xor
`snap`	number	no	1e-08	Snapping threshold for boundaries
`-t`	boolean	no	False	Do not create attribute table
`output`	vectorDestination	yes	—	Overlay
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.overlay ainput=input.gpkg binput=input.gpkg atype=0 btype=0 operator=0
↪  snap=1e-08 output=output.gpkg
```

This calls **v.overlay**: `ainput=input.gpkg` — Input layer (A); `binput=input.gpkg` — Input layer (B); `atype=0` selects *area*; `btype=0` selects *area*; `operator=0` selects *and*; `snap=1e-08` — Snapping threshold for boundaries; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.pack` — v.pack

group: Vector (v.*)

Exports a vector map as GRASS GIS specific archive file.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Name of input vector map to pack
`-c *(adv)*`	boolean	no	False	Switch the compression off
`output`	fileDestination	no	output.pack	Packed archive
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent

Parameter	Type	Required	Default	Description
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputFile)

Example usage

```
run grass:v.pack input=input.gpkg output=output.txt
```

This calls **v.pack**: `input=input.gpkg` — Name of input vector map to pack; `output=output.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.parallel — v.parallel

group: Vector (v.*)

Creates parallel line to input vector lines.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input lines
distance	number	no	1.0	Offset along major axis in map units
minordistance	number	no	—	Offset along minor axis in map units
angle	number	no	0.0	Angle of major axis in degrees
side	enum	no	0	Side — options: 0=left, 1=right, 2=both
tolerance	number	no	—	Tolerance of arc polylines in map units
-r	boolean	no	False	Make outside corners round
-b	boolean	no	False	Create buffer-like parallel lines
output	vectorDestination	yes	—	Parallel lines
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.parallel input=input.gpkg distance=1 minordistance=10 angle=0 side=0
↪  tolerance=10 output=output.gpkg
```

This calls **v.parallel**: `input=input.gpkg` — Input lines; `distance=1` — Offset along major axis in map units; `minordistance=10` — Offset along minor axis in map units; `angle=0` — Angle of major axis in degrees; `side=0` selects *left*; `tolerance=10` — Tolerance of arc polylines in map units; ou tput=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.patch — v.patch

group: Vector (v.*)

Create a new vector map layer by combining other vector map layers.

Parameter	Type	Required	Default	Description
input	multilayer	yes	—	Input layers
-e	boolean	no	True	Copy also attribute table
output	vectorDestination	yes	—	Combined
bbox	vectorDestination	yes	—	Bounding boxes
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector), bbox (outputVector)

Example usage

```
run grass:v.patch input="a.gpkg;b.gpkg" output=output.gpkg
```

This calls **v.patch**: input="a.gpkg;b.gpkg" — Input layers; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.perturb — v.perturb

group: Vector (v.*)

Random location perturbations of GRASS vector points

Parameter	Type	Required	Default	Description
input	source	yes	—	Vector points to be spatially perturbed
distribution	enum	no	0	Distribution of perturbation — options: 0=uniform, 1=normal
parameters	string	no	—	Parameter(s) of distribution (uniform: maximum; normal: mean and stddev)
minimum	number	no	0.0	Minimum deviation in map units
seed	number	no	0.0	Seed for random number generation
output	vectorDestination	yes	—	Perturbed
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.perturb input=input.gpkg distribution=0 parameters=value minimum=0 seed=0
↪  output=output.gpkg
```

This calls **v.perturb**: `input=input.gpkg` — Vector points to be spatially perturbed; `distribution=0` selects *uniform*; `parameters=value` — Parameter(s) of distribution (uniform: maximum; normal: mean and stddev); `minimum=0` — Minimum deviation in map units; `seed=0` — Seed for random number generation; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.proj` — v.proj

group: Vector (v.*)

Re-projects a vector layer to another coordinate reference system

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector to reproject
`crs`	crs	yes	—	New coordinate reference system
`smax`	number	no	10000.0	Maximum segment length in meters in output vector map
`-z *(adv)*`	boolean	no	False	Assume z coordinate is ellipsoidal height and transform if possible
`-w *(adv)*`	boolean	no	False	Disable wrapping to -180,180 for latlon output
`output`	vectorDestination	yes	—	Output vector map
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.proj input=input.gpkg crs=EPSG:6346 smax=10000 output=output.gpkg
```

This calls **v.proj**: `input=input.gpkg` — Input vector to reproject; `crs=EPSG:6346` — New coordinate reference system; `smax=10000` — Maximum segment length in meters in output vector map; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.qcount` — v.qcount

group: Vector (v.*)

Indices for quadrat counts of vector point lists.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Vector points layer
`nquadrats`	number	no	4.0	Number of quadrats
`radius`	number	no	10.0	Quadrat radius
`output`	vectorDestination	yes	—	Quadrats

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.qcount input=input.gpkg nquadrats=4 radius=10 output=output.gpkg
```

This calls **v.qcount**: `input=input.gpkg` — Vector points layer; `nquadrats=4` — Number of quadrats; `radius=10` — Quadrat radius; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.random` — v.random

group: Vector (v.*)

Randomly generate a 2D/3D vector points map.

Parameter	Type	Required	Default	Description
npoints	number	no	100.0	Number of points to be created
restrict	source	no	—	Restrict points to areas in input vector
where	string	no	—	WHERE conditions of SQL statement without 'where' keyword
zmin	number	no	0.0	Minimum z height for 3D output
zmax	number	no	0.0	Maximum z height for 3D output
seed	number	no	—	Seed for random number generation
column	string	no	z	Column for Z values
column_type	enum	no	0	Type of column for z values — options: 0=integer, 1=double precision
-z	boolean	no	False	Create 3D output
-a	boolean	no	False	Generate n points for each individual area
output	vectorDestination	yes	—	Random
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.random npoints=100 restrict=input.gpkg where=value zmin=0 zmax=0 seed=10
↪  column=z output=output.gpkg
```

This calls **v.random**: `npoints=100` — Number of points to be created; `restrict=input.gpkg` — Restrict points to areas in input vector; `where=value` — WHERE conditions of SQL statement without 'where' keyword; `zmin=0` — Minimum z height for 3D output; `zmax=0` — Maximum z height for 3D output; `seed=10` — Seed for random number generation; `column=z` — Column for Z values; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.rast.stats` — v.rast.stats

group: Vector (v.*)

Calculates univariate statistics from a raster map based on vector polygons and uploads statistics to new attribute columns.

Parameter	Type	Required	Default	Description
`map`	source	yes	—	Name of vector polygon map
`raster`	raster	yes	—	Name of raster map to calculate statistics from
`column_prefix`	string	yes	—	Column prefix for new attribute columns
`method`	enum	no	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]	The methods to use — options: 0=number, 1=minimum, 2=maximum, 3=range, 4=average, 5=stddev, 6=variance, 7=coeff_var, 8=sum, 9=first_quartile, 10=median, 11=third_quartile, 12=percentile
`percentile`	number	no	90.0	Percentile to calculate
`output`	vectorDestination	yes	—	Rast stats
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.rast.stats map=input.gpkg raster=dem.tif column_prefix=value method=0
↪  percentile=90 output=output.gpkg
```

This calls **v.rast.stats**: `map=input.gpkg` — Name of vector polygon map; `raster=dem.tif` — Name of raster map to calculate statistics from; `column_prefix=value` — Column prefix for new attribute columns; `method=0` selects *number*; `percentile=90` — Percentile to calculate; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.reclass — v.reclass

group: Vector (v.*)

Changes vector category values for an existing vector map according to results of SQL queries or a value in attribute table column.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input layer
type	enum	no	[0, 1, 2, 3]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid
column	field	no	—	The name of the column whose values are to be used as new categories
rules	file	no	—	Reclass rule file
output	vectorDestination	yes	—	Reclassified
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.reclass input=input.gpkg type=0 column=field1 rules=input.dat
 ↪  output=output.gpkg
```

This calls **v.reclass**: input=input.gpkg — Input layer; type=0 selects *point*; column=field1 — The name of the column whose values are to be used as new categories; rules=input.dat — Reclass rule file; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.rectify — v.rectify

group: Vector (v.*)

Rectifies a vector by computing a coordinate transformation for each object in the vector based on the control points.

Parameter	Type	Required	Default	Description
input	source	yes	—	Name of input vector map
inline_points	string	no	—	Inline control points
points	file	no	—	Name of input file with control points
order	number	no	1.0	Rectification polynomial order
separator	string	no	pipe	Field separator for RMS report
-3 *(adv)*	boolean	no	False	Perform 3D transformation
-o *(adv)*	boolean	no	False	Perform orthogonal 3D transformation
-b *(adv)*	boolean	no	False	Do not build topology
output	vectorDestination	yes	—	Rectified
rmsfile	fileDestination	yes	—	Root Mean Square errors file
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)

Parameter	Type	Required	Default	Description
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector), rmsfile (outputFile)

Example usage

```
run grass:v.rectify input=input.gpkg inline_points=value points=input.dat order=1
↪  separator=pipe output=output.gpkg
```

This calls **v.rectify**: input=input.gpkg — Name of input vector map; inline_points=value — Inline control points; points=input.dat — Name of input file with control points; order=1 — Rectification polynomial order; separator=pipe — Field separator for RMS report; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.report — v.report

group: Vector (v.*)

Reports geometry statistics for vectors.

Parameter	Type	Required	Default	Description
map	source	yes	—	Input layer
option	enum	no	0	Value to calculate — options: 0=area, 1=length, 2=coor
units	enum	no	2	units — options: 0=miles, 1=feet, 2=meters, 3=kilometers, 4=acres, 5=hectares, 6=percent
sort	enum	no	0	Sort the result (ascending, descending) — options: 0=asc, 1=desc
html	fileDestination	no	report.html	Report
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: html (outputHtml)

Example usage

```
run grass:v.report map=input.gpkg option=0 units=2 sort=0 html=html.txt
```

This calls **v.report**: map=input.gpkg — Input layer; option=0 selects *area*; units=2 selects *meters*; sort=0 selects *asc*; html=html.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.sample — v.sample

group: Vector (v.*)

Samples a raster layer at vector point locations.

Parameter	Type	Required	Default	Description
input	source	yes	—	Vector layer defining sample points
column	field	yes	—	Vector layer attribute column to use for comparison
raster	raster	yes	—	Raster map to be sampled
zscale	number	no	1.0	Sampled raster values will be multiplied by this factor
method	enum	no	0	Sampling interpolation method — options: 0=nearest, 1=bilinear, 2=bicubic
output	vectorDestination	no	—	Sampled
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.sample input=input.gpkg column=field1 raster=dem.tif zscale=1 method=0
↪  output=output.gpkg
```

This calls **v.sample**: input=input.gpkg — Vector layer defining sample points; column=field1 — Vector layer attribute column to use for comparison; raster=dem.tif — Raster map to be sampled; zscale=1 — Sampled raster values will be multiplied by this factor; method=0 selects *nearest*; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.segment — v.segment

group: Vector (v.*)

Creates points/segments from input vector lines and positions.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input lines layer
rules	file	yes	—	File containing segment rules
output	vectorDestination	yes	—	Segments
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.segment input=input.gpkg rules=input.dat output=output.gpkg
```

This calls **v.segment**: `input=input.gpkg` — Input lines layer; `rules=input.dat` — File containing segment rules; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.select` — v.select

group: Vector (v.*)

Selects features from vector map (A) by features from other vector map (B).

Parameter	Type	Required	Default	Description
`ainput`	source	yes	—	Input layer (A)
`atype`	enum	no	[0, 1, 4]	Input layer (A) Type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area
`binput`	source	yes	—	Input layer (B)
`btype`	enum	no	[0, 1, 4]	Input layer (B) Type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area
`operator`	enum	no	0	Operator to use — options: 0=overlap, 1=equals, 2=disjoint, 3=intersect, 4=touches, 5=crosses, 6=within, 7=contains, 8=overlaps, 9=relate
`relate`	string	no	—	Intersection Matrix Pattern used for 'relate' operator
`-t`	boolean	no	False	Do not create attribute table
`-c`	boolean	no	False	Do not skip features without category
`-r`	boolean	no	False	Reverse selection
`output`	vectorDestination	yes	—	Selected
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.select ainput=input.gpkg binput=input.gpkg atype=0 btype=0 operator=0
↪  relate=value output=output.gpkg
```

This calls **v.select**: `ainput=input.gpkg` — Input layer (A); `binput=input.gpkg` — Input layer (B); `atype=0` selects *point*; `btype=0` selects *point*; `operator=0` selects *overlap*; `relate=value` — Intersection Matrix Pattern used for 'relate' operator; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.split` — v.split

group: Vector (v.*)

Split lines to shorter segments by length.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input lines layer
`length`	number	no	—	Maximum segment length
`units`	enum	no	0	Length units — options: 0=map, 1=meters, 2=kilometers, 3=feet, 4=surveyfeet, 5=miles, 6=nautmiles
`vertices`	number	no	—	Maximum number of vertices in segment
`-n`	boolean	no	False	Add new vertices, but do not split
`-f`	boolean	no	False	Force segments to be exactly of given length, except for last one
`output`	vectorDestination	yes	—	Split by length
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.split input=input.gpkg length=10 units=0 vertices=10 output=output.gpkg
```

This calls **v.split**: `input=input.gpkg` — Input lines layer; `length=10` — Maximum segment length; `units=0` selects *map*; `vertices=10` — Maximum number of vertices in segment; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.surf.bspline` — v.surf.bspline

group: Vector (v.*)

Bicubic or bilinear spline interpolation with Tykhonov regularization.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input points layer
`column`	field	no	—	Attribute table column with values to interpolate
`sparse_input`	source	no	—	Sparse points layer
`ew_step`	number	no	4.0	Length of each spline step in the east-west direction
`ns_step`	number	no	4.0	Length of each spline step in the north-south direction
`method`	enum	no	0	Spline interpolation algorithm — options: 0=bilinear, 1=bicubic
`lambda_i`	number	no	0.01	Tykhonov regularization parameter (affects smoothing)
`solver`	enum	no	0	Type of solver which should solve the symmetric linear equation system — options: 0=cholesky, 1=cg
`maxit`	number	no	10000.0	Maximum number of iteration used to solve the linear equation system
`error`	number	no	1e-06	Error break criteria for iterative solver
`memory`	number	no	300.0	Maximum memory to be used (in MB)

Parameter	Type	Required	Default	Description
output	vectorDestination	no	—	Output vector
raster_output	rasterDestination	no	—	Interpolated spline
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector), raster_output (outputRaster)

Example usage

```
run grass:v.surf.bspline input=input.gpkg column=field1 sparse_input=input.gpkg ew_step=4
↪  ns_step=4 method=0 lambda_i=0.01 output=output.gpkg
```

This calls **v.surf.bspline**: input=input.gpkg — Input points layer; column=field1 — Attribute table column with values to interpolate; sparse_input=input.gpkg — Sparse points layer; ew_step=4 — Length of each spline step in the east-west direction; ns_step=4 — Length of each spline step in the north-south direction; method=0 selects *bilinear*; lambda_i=0.01 — Tykhonov regularization parameter (affects smoothing); output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.surf.idw — v.surf.idw

group: Vector (v.*)

Surface interpolation from vector point data by Inverse Distance Squared Weighting.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector layer
npoints	number	no	12.0	Number of interpolation points
power	number	no	2.0	Power parameter; greater values assign greater influence to closer points
column	field	yes	—	Attribute table column with values to interpolate
-n	boolean	no	False	Don't index points by raster cell
output	rasterDestination	yes	—	Interpolated IDW
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: output (outputRaster)

Example usage

```
run grass:v.surf.idw input=input.gpkg column=field1 npoints=12 power=2 output=output.tif
```

This calls **v.surf.idw**: `input=input.gpkg` — Input vector layer; `column=field1` — Attribute table column with values to interpolate; `npoints=12` — Number of interpolation points; `power=2` — Power parameter; greater values assign greater influence to closer points; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.surf.rst` — v.surf.rst

group: Vector (v.*)

Performs surface interpolation from vector points map by splines.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input points layer
`zcolumn`	field	no	—	Name of the attribute column with values to be used for approximation
`where`	string	no	—	WHERE conditions of SQL statement without 'where' keyword
`mask`	raster	no	—	Name of the raster map used as mask
`tension`	number	no	40.0	Tension parameter
`smooth`	number	no	—	Smoothing parameter
`smooth_column`	field	no	—	Name of the attribute column with smoothing parameters
`segmax`	number	no	40.0	Maximum number of points in a segment
`npmin`	number	no	300.0	Minimum number of points for approximation in a segment (>segmax)
`dmin`	number	no	—	Minimum distance between points (to remove almost identical points)
`dmax`	number	no	—	Maximum distance between points on isoline (to insert additional points)
`zscale`	number	no	1.0	Conversion factor for values used for approximation
`theta`	number	no	—	Anisotropy angle (in degrees counterclockwise from East)
`scalex`	number	no	—	Anisotropy scaling factor
`-t`	boolean	no	False	Use scale dependent tension
`-d`	boolean	no	False	Output partial derivatives instead of topographic parameters
`elevation`	rasterDestination	no	—	Interpolated RST
`slope`	rasterDestination	no	—	Slope
`aspect`	rasterDestination	no	—	Aspect
`pcurvature`	rasterDestination	no	—	Profile curvature
`tcurvature`	rasterDestination	no	—	Tangential curvature
`mcurvature`	rasterDestination	no	—	Mean curvature
`deviations`	vectorDestination	no	—	Deviations
`treeseg`	vectorDestination	no	—	Quadtree Segmentation
`overwin`	vectorDestination	no	—	Overlapping Windows
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_REGION_CELLSIZE_PARAMETER *(adv)*`	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
`GRASS_RASTER_FORMAT_OPT *(adv)*`	string	no	—	Output Rasters format options (createopt)
`GRASS_RASTER_FORMAT_META *(adv)*`	string	no	—	Output Rasters format metadata options (metaopt)
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Parameter	Type	Required	Default	Description
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: elevation (outputRaster), slope (outputRaster), aspect (outputRaster), pcurvature (outputRaster), tcurvature (outputRaster), mcurvature (outputRaster), deviations (outputVector), treeseg (outputVector), overwin (outputVector)

Example usage

```
run grass:v.surf.rst input=input.gpkg zcolumn=field1 where=value mask=dem.tif tension=40
 ↪  smooth=10 smooth_column=field1 elevation=elevation.tif
```

This calls **v.surf.rst**: input=input.gpkg — Input points layer; zcolumn=field1 — Name of the attribute column with values to be used for approximation; where=value — WHERE conditions of SQL statement without 'where' keyword; mask=dem.tif — Name of the raster map used as mask; tension=40 — Tension parameter; smooth=10 — Smoothing parameter; smooth_column=field1 — Name of the attribute column with smoothing parameters; elevation=elevation.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.surf.rst.cvdev — v.surf.rst.cvdev

group: Vector (v.*)

v.surf.rst.cvdev - Performs surface interpolation from vector points map by splines.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input points layer
zcolumn	field	no	—	Name of the attribute column with values to be used for approximation
where	string	no	—	WHERE conditions of SQL statement without 'where' keyword
mask	raster	no	—	Name of the raster map used as mask
tension	number	no	40.0	Tension parameter
smooth	number	no	—	Smoothing parameter
smooth_column	field	no	—	Name of the attribute column with smoothing parameters
segmax	number	no	40.0	Maximum number of points in a segment
npmin	number	no	300.0	Minimum number of points for approximation in a segment (>segmax)
dmin	number	no	—	Minimum distance between points (to remove almost identical points)
dmax	number	no	—	Maximum distance between points on isoline (to insert additional points)
zscale	number	no	1.0	Conversion factor for values used for approximation
theta	number	no	—	Anisotropy angle (in degrees counterclockwise from East)
scalex	number	no	—	Anisotropy scaling factor
-t	boolean	no	False	Use scale dependent tension
-c	boolean	no	True	Perform cross-validation procedure without raster approximation [leave this option as True]
cvdev	vectorDestination	no	—	Cross Validation Errors

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: cvdev (outputVector)

Example usage

```
run grass:v.surf.rst.cvdev input=input.gpkg zcolumn=field1 where=value mask=dem.tif
 ↪  tension=40 smooth=10 smooth_column=field1 cvdev=cvdev.gpkg
```

This calls **v.surf.rst.cvdev**: `input=input.gpkg` — Input points layer; `zcolumn=field1` — Name of the attribute column with values to be used for approximation; `where=value` — WHERE conditions of SQL statement without 'where' keyword; `mask=dem.tif` — Name of the raster map used as mask; `tension=40` — Tension parameter; `smooth=10` — Smoothing parameter; `smooth_column=field1` — Name of the attribute column with smoothing parameters; `cvdev=cvdev.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.to.3d` — v.to.3d

group: Vector (v.*)

Performs transformation of 2D vector features to 3D.

Parameter	Type	Required	Default	Description
input	source	yes	—	Name of input vector map
type	enum	no	[0, 1, 2, 3]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid
height	number	no	—	Fixed height for 3D vector features
column	field	no	—	Name of attribute column used for height
-r *(adv)*	boolean	no	False	Reverse transformation; 3D vector features to 2D
-t *(adv)*	boolean	no	False	Do not copy attribute table
output	vectorDestination	yes	—	3D
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.to.3d input=input.gpkg type=0 height=10 column=field1 output=output.gpkg
```

This calls **v.to.3d**: `input=input.gpkg` — Name of input vector map; `type=0` selects *point*; `height=10` — Fixed height for 3D vector features; `column=field1` — Name of attribute column used for height; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.to.lines` — v.to.lines

group: Vector (v.*)

Converts vector polygons or points to lines.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Name of input vector map
`method`	enum	no	0	Method used for point interpolation — options: 0=delaunay
`output`	vectorDestination	yes	—	Lines
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.to.lines input=input.gpkg method=0 output=output.gpkg
```

This calls **v.to.lines**: `input=input.gpkg` — Name of input vector map; `method=0` selects *delaunay*; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.to.points` — v.to.points

group: Vector (v.*)

Create points along input lines

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input lines layer
`type`	enum	no	[0, 1, 2, 3, 5]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area, 5=face, 6=kernel
`use`	enum	no	0	Use line nodes or vertices only — options: 0=node, 1=vertex
`dmax`	number	no	100.0	Maximum distance between points in map units
`-i`	boolean	no	False	Interpolate points between line vertices
`-t`	boolean	no	False	Do not create attribute table
`output`	vectorDestination	yes	—	Points along lines

Parameter	Type	Required	Default	Description
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.to.points input=input.gpkg type=0 use=0 dmax=100 output=output.gpkg
```

This calls **v.to.points**: `input=input.gpkg` — Input lines layer; `type=0` selects *point*; `use=0` selects *node*; `dmax=100` — Maximum distance between points in map units; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.to.rast` — v.to.rast

group: Vector (v.*)

Converts (rasterize) a vector layer into a raster layer.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input vector layer
type	enum	no	[0, 1, 3]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=area
where	string	no	—	WHERE conditions of SQL statement without 'where' keyword
use	enum	no	0	Source of raster values — options: 0=attr, 1=cat, 2=val, 3=z, 4=dir
attribute_column	field	no	—	Name of column for 'attr' parameter (data type must be numeric)
rgb_column	field	no	—	Name of color definition column (with RRR:GGG:BBB entries)
label_column	field	no	—	Name of column used as raster category labels
value	number	no	1.0	Raster value (for use=val)
memory	number	no	300.0	Maximum memory to be used (in MB)
output	rasterDestination	yes	—	Rasterized
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_RASTER_FORMAT_OPT *(adv)*	string	no	—	Output Rasters format options (createopt)
GRASS_RASTER_FORMAT_META *(adv)*	string	no	—	Output Rasters format metadata options (metaopt)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area

Outputs: `output` (outputRaster)

Example usage

```
run grass:v.to.rast input=input.gpkg type=0 where=value use=0 attribute_column=field1
 ↪  rgb_column=field1 label_column=field1 output=output.tif
```

This calls **v.to.rast**: `input=input.gpkg` — Input vector layer; `type=0` selects *point*; `where=value` — WHERE conditions of SQL statement without 'where' keyword; `use=0` selects *attr*; `attribute_column=field1` — Name of column for 'attr' parameter (data type must be numeric); `rgb_column=field1` — Name of color definition column (with RRR:GGG:BBB entries); `label_column=field1` — Name of column used as raster category labels; `output=output.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.transform` — v.transform

group: Vector (v.*)

Performs an affine transformation on a vector layer.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Input vector layer
`xshift`	number	no	0.0	X shift
`yshift`	number	no	0.0	Y shift
`zshift`	number	no	0.0	Z shift
`xscale`	number	no	1.0	X scale
`yscale`	number	no	1.0	Y scale
`zscale`	number	no	1.0	Z scale
`zrotation`	number	no	0.0	Rotation around z axis in degrees counterclockwise
`columns`	string	no	—	Name of attribute column(s) used as transformation parameters (Format: parameter:column, e.g. xshift:xs,yshift:ys,zrot:zr)
`-t *(adv)*`	boolean	no	False	Shift all z values to bottom=0
`-w *(adv)*`	boolean	no	False	Swap coordinates x, y and then apply other parameters
`-x *(adv)*`	boolean	no	False	Swap coordinates x, z and then apply other parameters
`-y *(adv)*`	boolean	no	False	Swap coordinates y, z and then apply other parameters
`-a *(adv)*`	boolean	no	False	Swap coordinates after the other transformations
`-b *(adv)*`	boolean	no	False	Do not build topology
`output`	vectorDestination	yes	—	Transformed
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.transform input=input.gpkg xshift=0 yshift=0 zshift=0 xscale=1 yscale=1
↪  zscale=1 output=output.gpkg
```

This calls **v.transform**: `input=input.gpkg` — Input vector layer; `xshift=0` — X shift; `yshift=0` — Y shift; `zshift=0` — Z shift; `xscale=1` — X scale; `yscale=1` — Y scale; `zscale=1` — Z scale; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.type` — v.type

group: Vector (v.*)

Change the type of geometry elements.

Parameter	Type	Required	Default	Description
`input`	source	yes	—	Name of existing vector map
`from_type`	enum	no	1	Feature type to convert from — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=face, 5=kernel
`to_type`	enum	no	2	Feature type to convert to — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=face, 5=kernel
`output`	vectorDestination	yes	—	Typed
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)
`GRASS_VECTOR_EXPORT_NOCAT *(adv)*`	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: `output` (outputVector)

Example usage

```
run grass:v.type input=input.gpkg from_type=1 to_type=2 output=output.gpkg
```

This calls **v.type**: `input=input.gpkg` — Name of existing vector map; `from_type=1` selects *line*; `to_type=2` selects *boundary*; `output=output.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.univar` — v.univar

group: Vector (v.*)

Calculates univariate statistics for attribute. Variance and standard deviation is calculated only for points if specified.

Parameter	Type	Required	Default	Description
`map`	source	yes	—	Name of input vector map
`type`	enum	no	[0, 1, 4]	Input feature type — options: 0=point, 1=line, 2=boundary, 3=centroid, 4=area
`where`	string	no	—	WHERE conditions of SQL statement without 'where' keyword

Parameter	Type	Required	Default	Description
`column`	field	yes	—	Column name
`percentile`	number	no	90.0	Percentile to calculate
`-g`	boolean	no	True	Print the stats in shell script style
`-e`	boolean	no	False	Calculate extended statistics
`-w`	boolean	no	False	Weigh by line length or area size
`-d`	boolean	no	False	Calculate geometric distances instead of attribute statistics
`html`	fileDestination	no	report.html	Statistics
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area

Outputs: `html` (outputHtml)

Example usage

```
run grass:v.univar map=input.gpkg column=field1 type=0 where=value percentile=90
↪  html=html.txt
```

This calls **v.univar**: `map=input.gpkg` — Name of input vector map; `column=field1` — Column name; `type=0` selects *point*; `where=value` — WHERE conditions of SQL statement without 'where' keyword; `percentile=90` — Percentile to calculate; `html=html.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`grass:v.vect.stats` — v.vect.stats

group: Vector (v.*)

Count points in areas and calculate statistics.

Parameter	Type	Required	Default	Description
`points`	source	yes	—	Name of existing vector map with points
`areas`	source	yes	—	Name of existing vector map with areas
`type`	enum	no	[0]	Input feature type — options: 0=point, 1=centroid
`method`	enum	no	0	Method for aggregate statistics — options: 0=sum, 1=average, 2=median, 3=mode, 4=minimum, 5=min_cat, 6=maximum, 7=max_cat, 8=range, 9=stddev, 10=variance, 11=diversity
`points_column`	field	yes	—	Column name of points map to use for statistics
`count_column`	string	yes	—	Column name to upload points count (integer, created if doesn't exists)
`stats_column`	string	yes	—	Column name to upload statistics (double, created if doesn't exists)
`output`	vectorDestination	yes	—	Updated
`GRASS_REGION_PARAMETER *(adv)*`	extent	no	—	GRASS GIS region extent
`GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*`	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
`GRASS_MIN_AREA_PARAMETER *(adv)*`	number	no	0.0001	v.in.ogr min area
`GRASS_OUTPUT_TYPE_PARAMETER *(adv)*`	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
`GRASS_VECTOR_DSCO *(adv)*`	string	no	—	v.out.ogr output data source options (dsco)
`GRASS_VECTOR_LCO *(adv)*`	string	no	—	v.out.ogr output layer options (lco)

Parameter	Type	Required	Default	Description
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.vect.stats points=input.gpkg areas=input.gpkg points_column=field1
 ↪  count_column=value stats_column=value type=0 method=0 output=output.gpkg
```

This calls **v.vect.stats**: points=input.gpkg — Name of existing vector map with points; areas=input.gpkg — Name of existing vector map with areas; points_column=field1 — Column name of points map to use for statistics; count_column=value — Column name to upload points count (integer, created if doesn't exists); stats_column=value — Column name to upload statistics (double, created if doesn't exists); type=0 selects *point*; method=0 selects *sum*; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.voronoi — v.voronoi

group: Vector (v.*)

v.voronoi - Creates a Voronoi diagram from an input vector layer containing points.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input points layer
-l *(adv)*	boolean	no	False	Output tessellation as a graph (lines), not areas
-t *(adv)*	boolean	no	False	Do not create attribute table
output	vectorDestination	yes	—	Voronoi
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.voronoi input=input.gpkg output=output.gpkg
```

This calls **v.voronoi**: input=input.gpkg — Input points layer; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.voronoi.skeleton — v.voronoi.skeleton

group: Vector (v.*)

v.voronoi.skeleton - Creates a Voronoi diagram for polygons or compute the center line/skeleton of polygons.

Parameter	Type	Required	Default	Description
input	source	yes	—	Input polygons layer
smoothness	number	no	0.25	Factor for output smoothness
thin	number	no	-1.0	Maximum dangle length of skeletons (-1 will extract the center line)
-a *(adv)*	boolean	no	False	Create Voronoi diagram for input areas
-s *(adv)*	boolean	no	True	Extract skeletons for input areas
-l *(adv)*	boolean	no	False	Output tessellation as a graph (lines), not areas
-t *(adv)*	boolean	no	False	Do not create attribute table
output	vectorDestination	yes	—	Voronoi
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.voronoi.skeleton input=input.gpkg smoothness=0.25 thin=-1 output=output.gpkg
```

This calls **v.voronoi.skeleton**: input=input.gpkg — Input polygons layer; smoothness=0.25 — Factor for output smoothness; thin=-1 — Maximum dangle length of skeletons (-1 will extract the center line); output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.what.rast — v.what.rast

group: Vector (v.*)

Uploads raster values at positions of vector centroids to the table.

Parameter	Type	Required	Default	Description
map	source	yes	—	Name of vector points map for which to edit attributes
raster	raster	yes	—	Raster map to be sampled
type	enum	no	0	Input feature type — options: 0=point, 1=centroid
column	field	yes	—	Name of attribute column to be updated with the query result
where	string	no	—	WHERE conditions of SQL statement without 'where' keyword
-i *(adv)*	boolean	no	False	Interpolate values from the nearest four cells
output	vectorDestination	yes	—	Sampled
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_REGION_CELLSIZE_PARAMETER *(adv)*	number	no	0.0	GRASS GIS region cellsize (leave 0 for default)
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area

Parameter	Type	Required	Default	Description
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.what.rast map=input.gpkg raster=dem.tif column=field1 type=0 where=value
↪  output=output.gpkg
```

This calls **v.what.rast**: map=input.gpkg — Name of vector points map for which to edit attributes; raster=dem.tif — Raster map to be sampled; column=field1 — Name of attribute column to be updated with the query result; type=0 selects *point*; where=value — WHERE conditions of SQL statement without 'where' keyword; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

grass:v.what.vect — v.what.vect

group: Vector (v.*)

Uploads vector values at positions of vector points to the table.

Parameter	Type	Required	Default	Description
map	source	yes	—	Name of vector points map for which to edit attributes
column	field	yes	—	Column to be updated with the query result
query_map	source	yes	—	Vector map to be queried
query_column	field	yes	—	Column to be queried
dmax	number	no	0.0	Maximum query distance in map units
output	vectorDestination	yes	—	Updated
GRASS_REGION_PARAMETER *(adv)*	extent	no	—	GRASS GIS region extent
GRASS_SNAP_TOLERANCE_PARAMETER *(adv)*	number	no	-1.0	v.in.ogr snap tolerance (-1 = no snap)
GRASS_MIN_AREA_PARAMETER *(adv)*	number	no	0.0001	v.in.ogr min area
GRASS_OUTPUT_TYPE_PARAMETER *(adv)*	enum	no	0	v.out.ogr output type — options: 0=auto, 1=point, 2=line, 3=area
GRASS_VECTOR_DSCO *(adv)*	string	no	—	v.out.ogr output data source options (dsco)
GRASS_VECTOR_LCO *(adv)*	string	no	—	v.out.ogr output layer options (lco)
GRASS_VECTOR_EXPORT_NOCAT *(adv)*	boolean	no	False	Also export features without category (not labeled). Otherwise only features with category are exported

Outputs: output (outputVector)

Example usage

```
run grass:v.what.vect map=input.gpkg column=field1 query_map=input.gpkg
↪  query_column=field1 dmax=0 output=output.gpkg
```

This calls **v.what.vect**: map=input.gpkg — Name of vector points map for which to edit attributes; column=field1 — Column to be updated with the query result; query_map=input.gpkg — Vector map to be queried; query_column=field1 — Column to be queried; dmax=0 — Maximum query distance in map units; output=output.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb: algorithms

otb — Orfeo ToolBox algorithms (optional provider plugin; see docs/guide/pdal-lastools-qgis4.md) — 109 algorithms (QGIS 4.0.3). Auto-generated by `scripts/gen_algorithms.py`; see the index.

Call any of these with `run <id> KEY=value …` (the **Name** column is the KEY); each entry includes a worked **Example usage**. A ★ marks an algorithm with a friendly niva alias verb.

otb:BandMath — BandMath

group: Image Manipulation

Parameter	Type	Required	Default	Description
`il`	multilayer	yes	—	Input image list
`out`	rasterDestination	yes	—	Output Image
`exp`	string	yes	—	Expression
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:BandMath il="a.gpkg;b.gpkg" exp=value out=out.tif
```

This calls **BandMath**: `il="a.gpkg;b.gpkg"` — Input image list; `exp=value` — Expression; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:BandMathX — BandMathX

group: Image Manipulation

Parameter	Type	Required	Default	Description
`il`	multilayer	yes	—	Input image list
`out`	rasterDestination	yes	—	Output Image
`exp`	string	yes	—	Expressions
`incontext`	file	no	—	Import context
`outcontext`	fileDestination	no	—	Export context
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster), `outcontext` (outputFile)

Example usage

```
run otb:BandMathX il="a.gpkg;b.gpkg" exp=value incontext=input.dat out=out.tif
```

This calls **BandMathX**: `il="a.gpkg;b.gpkg"` — Input image list; `exp=value` — Expressions; `incontext=input.dat` — Import context; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:BinaryMorphologicalOperation` — BinaryMorphologicalOperation

group: Feature Extraction

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Output Image
`channel`	number	no	1.0	Selected Channel
`structype`	OTBParameterChoice	no	0	Type of structuring element
`xradius`	number	no	5.0	Structuring element X radius
`yradius`	number	no	5.0	Structuring element Y radius
`foreval`	number	no	1.0	Foreground value
`backval`	number	no	0.0	Background value
`filter`	OTBParameterChoice	no	0	Morphological Operation
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:BinaryMorphologicalOperation in=dem.tif channel=1 structype=0 xradius=5 yradius=5
↪  foreval=1 backval=0 out=out.tif
```

This calls **BinaryMorphologicalOperation**: `in=dem.tif` — Input Image; `channel=1` — Selected Channel; `structype=0` — Type of structuring element; `xradius=5` — Structuring element X radius; `yradius=5` — Structuring element Y radius; `foreval=1` — Foreground value; `backval=0` — Background value; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:BlockMatching` — BlockMatching

group: Stereo

Parameter	Type	Required	Default	Description
`io.inleft`	raster	yes	—	Left input image
`io.inright`	raster	yes	—	Right input image
`io.out`	rasterDestination	yes	—	The output disparity map
`io.outmask`	rasterDestination	no	—	The output mask corresponding to all criterions
`io.outmetric`	boolean	no	False	Flag to output optimal metric values as well
`mask.inleft`	raster	no	—	Mask to discard left pixels
`mask.inright`	raster	no	—	Mask to discard right pixels
`mask.nodata`	number	no	0.0	Discard pixels with no-data value
`mask.variancet`	number	no	100.0	Discard pixels with low local variance
`bm.metric`	OTBParameterChoice	no	0	Block-matching metric
`bm.metric.lp.p`	number	no	1.0	p value
`bm.radius`	number	no	3.0	Radius of blocks
`bm.minhd`	number	no	0.0	Minimum horizontal disparity
`bm.maxhd`	number	no	0.0	Maximum horizontal disparity
`bm.minvd`	number	no	0.0	Minimum vertical disparity
`bm.maxvd`	number	no	0.0	Maximum vertical disparity

Parameter	Type	Required	Default	Description
bm.subpixel	OTBParameterChoice	no	0	Sub-pixel interpolation
bm.step	number	no	1.0	Computation step
bm.startx	number	no	0.0	X start index
bm.starty	number	no	0.0	Y start index
bm.medianfilter.radius	number	no	0.0	Radius
bm.medianfilter.incoherence	number	no	0.0	Incoherence threshold
bm.initdisp	OTBParameterChoice	no	0	Initial disparities
bm.initdisp.uniform.hdisp	number	no	0.0	Horizontal initial disparity
bm.initdisp.uniform.vdisp	number	no	0.0	Vertical initial disparity
bm.initdisp.uniform.hrad	number	no	0.0	Horizontal exploration radius
bm.initdisp.uniform.vrad	number	no	0.0	Vertical exploration radius
bm.initdisp.maps.hmap	raster	yes	—	Horizontal initial disparity map
bm.initdisp.maps.vmap	raster	yes	—	Vertical initial disparity map
bm.initdisp.maps.hrad	number	no	0.0	Horizontal exploration radius
bm.initdisp.maps.vrad	number	no	0.0	Vertical exploration radius
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: io.out (outputRaster), io.outmask (outputRaster)

Example usage

```
run otb:BlockMatching io.inleft=dem.tif io.inright=dem.tif bm.initdisp.maps.hmap=dem.tif
↪ bm.initdisp.maps.vmap=dem.tif io.outmetric=false mask.inleft=dem.tif
↪ mask.inright=dem.tif io.out=io.out.tif
```

This calls **BlockMatching**: io.inleft=dem.tif — Left input image; io.inright=dem.tif — Right input image; bm.initdisp.maps.hmap=dem.tif — Horizontal initial disparity map; bm.initdisp.maps.vmap=dem.tif — Vertical initial disparity map; io.outmetric=false — Flag to output optimal metric values as well; mask.inleft=dem.tif — Mask to discard left pixels; mask.inright=dem.tif — Mask to discard right pixels; io.out=io.out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:BundleToPerfectSensor — BundleToPerfectSensor

group: Geometry

Parameter	Type	Required	Default	Description
inp	raster	yes	—	Input PAN Image
inxs	raster	yes	—	Input XS Image
out	rasterDestination	yes	—	Output image
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
mode	OTBParameterChoice	no	0	Mode
method	OTBParameterChoice	no	0	Algorithm
method.rcs.radiusx	number	no	9.0	X radius
method.rcs.radiusy	number	no	9.0	Y radius
method.lmvm.radiusx	number	no	3.0	X radius
method.lmvm.radiusy	number	no	3.0	Y radius
method.bayes.lambda	number	no	0.9999	Weight
method.bayes.s	number	no	1.0	S coefficient
lms	number	no	4.0	Spacing of the deformation field
interpolator	OTBParameterChoice	no	0	Interpolation
interpolator.bco.radius	number	no	2.0	Radius for bicubic interpolation
fv	number	no	0.0	Fill Value

Parameter	Type	Required	Default	Description
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:BundleToPerfectSensor inp=dem.tif inxs=dem.tif elev.dem=input.dat
↪ elev.geoid=input.dat elev.default=0 mode=0 method=0 out=out.tif
```

This calls **BundleToPerfectSensor**: inp=dem.tif — Input PAN Image; inxs=dem.tif — Input XS Image; elev.dem=input.dat — DEM directory; elev.geoid=input.dat — Geoid File; elev.default=0 — Default elevation; mode=0 — Mode; method=0 — Algorithm; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ClassificationMapRegularization — ClassificationMapRegularization

group: Learning

Parameter	Type	Required	Default	Description
io.in	raster	yes	—	Input classification image
io.out	rasterDestination	yes	—	Output regularized image
ip.radius	number	no	1.0	Structuring element radius (in pixels)
ip.suvbool	boolean	no	False	Set tie pixels to undecided
ip.nodatalabel	number	no	0.0	Label for the NoData class
ip.undecidedlabel	number	no	0.0	Label for the Undecided class
ip.onlyisolatedpixels	boolean	no	False	Process isolated pixels only
ip.isolatedthreshold	number	no	1.0	Threshold for isolated pixels
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: io.out (outputRaster)

Example usage

```
run otb:ClassificationMapRegularization io.in=dem.tif ip.radius=1 ip.suvbool=false
↪ ip.nodatalabel=0 ip.undecidedlabel=0 ip.onlyisolatedpixels=false
↪ ip.isolatedthreshold=1 io.out=io.out.tif
```

This calls **ClassificationMapRegularization**: io.in=dem.tif — Input classification image; ip.radius=1 — Structuring element radius (in pixels); ip.suvbool=false — Set tie pixels to undecided; ip.nodatalabel=0 — Label for the NoData class; ip.undecidedlabel=0 — Label for the Undecided class; ip.onlyisolatedpixels=false — Process isolated pixels only; ip.isolatedthreshold=1 — Threshold for isolated pixels; io.out=io.out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ColorMapping — ColorMapping

group: Image Manipulation

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	rasterDestination	yes	—	Output Image
op	OTBParameterChoice	no	0	Operation

Parameter	Type	Required	Default	Description
op.colortolabel.notfound	number	no	404.0	Not Found Label
method	OTBParameterChoice	no	0	Color mapping method
method.custom.lut	file	yes	—	Look-up table file
method.continuous.lut	OTBParameterChoice	no	0	Look-up tables
method.continuous.min	number	no	0.0	Mapping range lower value
method.continuous.max	number	no	255.0	Mapping range higher value
method.optimal.background	number	no	0.0	Background label
method.image.in	raster	yes	—	Support Image
method.image.nodatavalue	number	no	0.0	NoData value
method.image.low	number	no	2.0	lower quantile
method.image.up	number	no	2.0	upper quantile
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:ColorMapping in=dem.tif method.custom.lut=input.dat method.image.in=dem.tif op=0
↪  op.colortolabel.notfound=404 method=0 method.continuous.lut=0 out=out.tif
```

This calls **ColorMapping**: in=dem.tif — Input Image; method.custom.lut=input.dat — Look-up table file; method.image.in=dem.tif — Support Image; op=0 — Operation; op.colortolabel.notfound=404 — Not Found Label; method=0 — Color mapping method; method.continuous.lut=0 — Look-up tables; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:CompareImages — CompareImages

group: Image Manipulation

Parameter	Type	Required	Default	Description
ref.in	raster	yes	—	Reference image
ref.channel	number	no	1.0	Reference image channel
meas.in	raster	yes	—	Measured image
meas.channel	number	no	1.0	Measured image channel
roi.startx	number	no	0.0	Start X
roi.starty	number	no	0.0	Start Y
roi.sizex	number	no	0.0	Size X
roi.sizey	number	no	0.0	Size Y

Example usage

```
run otb:CompareImages ref.in=dem.tif meas.in=dem.tif ref.channel=1 meas.channel=1
↪  roi.startx=0 roi.starty=0 roi.sizex=0
```

This calls **CompareImages**: ref.in=dem.tif — Reference image; meas.in=dem.tif — Measured image; ref.channel=1 — Reference image channel; meas.channel=1 — Measured image channel; roi.startx=0 — Start X; roi.starty=0 — Start Y; roi.sizex=0 — Size X. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ComputeConfusionMatrix — ComputeConfusionMatrix

group: Learning

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	fileDestination	yes	—	Matrix output
`format`	OTBParameterChoice	no	0	set the output format to contingency table or confusion matrix
`ref`	OTBParameterChoice	no	0	Ground truth
`ref.raster.in`	raster	yes	—	Input reference image
`ref.raster.nodata`	number	no	0.0	Value for nodata pixels in the reference raster
`ref.vector.in`	vector	yes	—	Input reference vector data
`ref.vector.field`	field	no	—	Field name
`ref.vector.nodata`	number	no	0.0	Value for nodata pixels in the reference vector
`nodatalabel`	number	no	0.0	Value for nodata pixels in the input image

Outputs: `out` (outputFile)

Example usage

```
run otb:ComputeConfusionMatrix in=dem.tif ref.raster.in=dem.tif ref.vector.in=input.gpkg
↪  format=0 ref=0 ref.raster.nodata=0 ref.vector.field=field1 out=out.txt
```

This calls **ComputeConfusionMatrix**: `in=dem.tif` — Input Image; `ref.raster.in=dem.tif` — Input reference image; `ref.vector.in=input.gpkg` — Input reference vector data; `format=0` — set the output format to contingency table or confusion matrix; `ref=0` — Ground truth; `ref.raster.nodata=0` — Value for nodata pixels in the reference raster; `ref.vector.field=field1` — Field name; `out=out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ComputeImagesStatistics` — ComputeImagesStatistics

group: Learning

Parameter	Type	Required	Default	Description
`il`	multilayer	yes	—	Input images
`bv`	number	no	0.0	Background Value
`out.xml`	fileDestination	no	—	Output XML file

Outputs: `out.xml` (outputFile)

Example usage

```
run otb:ComputeImagesStatistics il="a.gpkg;b.gpkg" bv=0 out.xml=out.xml.txt
```

This calls **ComputeImagesStatistics**: `il="a.gpkg;b.gpkg"` — Input images; `bv=0` — Background Value; `out.xml=out.xml.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ComputeModulusAndPhase` — ComputeModulusAndPhase

group: SAR

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`modulus`	rasterDestination	yes	—	Modulus
`phase`	rasterDestination	yes	—	Phase
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `modulus` (outputRaster), `phase` (outputRaster)

Example usage

```
run otb:ComputeModulusAndPhase in=dem.tif modulus=modulus.tif
```

This calls **ComputeModulusAndPhase**: `in=dem.tif` — Input Image; `modulus=modulus.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ComputeOGRLayersFeaturesStatistics` — ComputeOGRLayersFeaturesStatistics

group: Segmentation

Parameter	Type	Required	Default	Description
`inshp`	vector	yes	—	Vector Data
`outstats`	fileDestination	yes	—	Output XML file
`feat`	field	no	—	Feature

Outputs: `outstats` (outputFile)

Example usage

```
run otb:ComputeOGRLayersFeaturesStatistics inshp=input.gpkg feat=field1
↪  outstats=outstats.txt
```

This calls **ComputeOGRLayersFeaturesStatistics**: `inshp=input.gpkg` — Vector Data; `feat=field1` — Feature; `outstats=outstats.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ConcatenateImages` — ConcatenateImages

group: Image Manipulation

Parameter	Type	Required	Default	Description
`il`	multilayer	yes	—	Input images list
`out`	rasterDestination	yes	—	Output Image
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:ConcatenateImages il="a.gpkg;b.gpkg" out=out.tif
```

This calls **ConcatenateImages**: `il="a.gpkg;b.gpkg"` — Input images list; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ConcatenateVectorData` — ConcatenateVectorData

group: Vector Data Manipulation

Parameter	Type	Required	Default	Description
vd	multilayer	yes	—	Input vector files
out	vectorDestination	yes	—	Concatenated output

Outputs: out (outputVector)

Example usage

```
run otb:ConcatenateVectorData vd="a.gpkg;b.gpkg" out=out.gpkg
```

This calls **ConcatenateVectorData**: vd="a.gpkg;b.gpkg" — Input vector files; out=out.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ConnectedComponentSegmentation — ConnectedComponentSegmentation

group: Segmentation

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	vectorDestination	yes	—	Output Shape
mask	string	no	—	Mask expression
expr	string	yes	—	Connected Component Expression
minsize	number	no	2.0	Minimum Object Size
obia	string	no	—	OBIA Expression
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation

Outputs: out (outputVector)

Example usage

```
run otb:ConnectedComponentSegmentation in=dem.tif expr=value mask=value minsize=2
↪  obia=value elev.dem=input.dat elev.geoid=input.dat out=out.gpkg
```

This calls **ConnectedComponentSegmentation**: in=dem.tif — Input Image; expr=value — Connected Component Expression; mask=value — Mask expression; minsize=2 — Minimum Object Size; obia=value — OBIA Expression; elev.dem=input.dat — DEM directory; elev.geoid=input.dat — Geoid File; out=out.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ContrastEnhancement — ContrastEnhancement

group: Image Filtering

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	rasterDestination	yes	—	Output Image
bins	number	no	256.0	Number of bins
hfact	number	no	0.0	Contrast Limitation
nodata	number	no	0.0	Nodata Value
spatial	OTBParameterChoice	no	0	Spatial parameters for the histogram computation
spatial.local.h	number	no	256.0	Thumbnail height
spatial.local.w	number	no	256.0	Thumbnail width
minmax	OTBParameterChoice	no	0	Minimum and maximum settings
minmax.auto.global	boolean	no	False	Global

Parameter	Type	Required	Default	Description
minmax.manual.min	number	no	0.0	Minimum value
minmax.manual.max	number	no	0.0	Maximum value
mode	OTBParameterChoice	no	0	What to equalized
mode.lum.red.ch	number	no	0.0	Red channel
mode.lum.red.coef	number	no	0.21	Value for luminance computation for the red channel
mode.lum.green.ch	number	no	1.0	Green channel
mode.lum.green.coef	number	no	0.71	Value for luminance computation of the green channel
mode.lum.blue.ch	number	no	2.0	Blue channel
mode.lum.blue.coef	number	no	0.08	Value for luminance computation of the blue channel
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:ContrastEnhancement in=dem.tif bins=256 hfact=0 nodata=0 spatial=0
↪  spatial.local.h=256 spatial.local.w=256 out=out.tif
```

This calls **ContrastEnhancement**: `in=dem.tif` — Input Image; `bins=256` — Number of bins; `hfact=0` — Contrast Limitation; `nodata=0` — Nodata Value; `spatial=0` — Spatial parameters for the histogram computation; `spatial.local.h=256` — Thumbnail height; `spatial.local.w=256` — Thumbnail width; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ConvertCartoToGeoPoint — ConvertCartoToGeoPoint

group: Geometry

Parameter	Type	Required	Default	Description
carto.x	number	no	0.0	X cartographic coordinates
carto.y	number	no	0.0	Y cartographic coordinates
mapproj	OTBParameterChoice	no	0	Map Projection
mapproj.utm.zone	number	no	31.0	Zone number
mapproj.utm.northhem	boolean	no	False	Northern Hemisphere
mapproj.epsg.code	crs	no	ProjectCrs	EPSG Code

Example usage

```
run otb:ConvertCartoToGeoPoint carto.x=0 carto.y=0 mapproj=0 mapproj.utm.zone=31
↪  mapproj.utm.northhem=false mapproj.epsg.code=EPSG:6346
```

This calls **ConvertCartoToGeoPoint**: `carto.x=0` — X cartographic coordinates; `carto.y=0` — Y cartographic coordinates; `mapproj=0` — Map Projection; `mapproj.utm.zone=31` — Zone number; `mapproj.utm.northhem=false` — Northern Hemisphere; `mapproj.epsg.code=EPSG:6346` — EPSG Code. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ConvertSensorToGeoPoint — ConvertSensorToGeoPoint

group: Geometry

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Sensor image
`input.idx`	number	no	0.0	X value of desired point
`input.idy`	number	no	0.0	Y value of desired point

Example usage

```
run otb:ConvertSensorToGeoPoint in=dem.tif input.idx=0 input.idy=0
```

This calls **ConvertSensorToGeoPoint**: `in=dem.tif` — Sensor image; `input.idx=0` — X value of desired point; `input.idy=0` — Y value of desired point. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:Despeckle` — Despeckle

group: Image Filtering

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Output Image
`filter`	OTBParameterChoice	no	0	Speckle filtering method
`filter.lee.rad`	number	no	1.0	Radius
`filter.lee.nblooks`	number	no	1.0	Number of looks
`filter.frost.rad`	number	no	1.0	Radius
`filter.frost.deramp`	number	no	0.1	Deramp factor
`filter.gammamap.rad`	number	no	1.0	Radius
`filter.gammamap.nblooks`	number	no	1.0	Number of looks
`filter.kuan.rad`	number	no	0.0	Radius
`filter.kuan.nblooks`	number	no	0.0	Number of looks
`outputpixeltype` *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:Despeckle in=dem.tif filter=0 filter.lee.rad=1 filter.lee.nblooks=1
↪  filter.frost.rad=1 filter.frost.deramp=0.1 filter.gammamap.rad=1 out=out.tif
```

This calls **Despeckle**: `in=dem.tif` — Input Image; `filter=0` — Speckle filtering method; `filter.lee.rad=1` — Radius; `filter.lee.nblooks=1` — Number of looks; `filter.frost.rad=1` — Radius; `filter.frost.deramp=0.1` — Deramp factor; `filter.gammamap.rad=1` — Radius; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:DimensionalityReduction` — DimensionalityReduction

group: Image Filtering

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	no	—	Output Image
`rescale`	OTBParameterChoice	no	0	Rescale Output
`rescale.minmax.outmin`	number	no	0.0	Output min value
`rescale.minmax.outmax`	number	no	255.0	Output max value
`outinv`	rasterDestination	no	—	Inverse Output Image
`method`	OTBParameterChoice	no	0	Algorithm

Parameter	Type	Required	Default	Description
method.pca.outeigenvalues	fileDestination	no	—	Output file containing eigenvalues (txt format)
method.pca.whiten	boolean	no	True	Perform pca whitening
method.napca.radiusx	number	no	1.0	Set the x radius of the sliding window
method.napca.radiusy	number	no	1.0	Set the y radius of the sliding window
method.ica.iter	number	no	20.0	number of iterations
method.ica.mu	number	no	1.0	Give the increment weight of W in [0, 1]
method.ica.g	OTBParameterChoice	no	0	Nonlinearity
nbcomp	number	no	0.0	Number of Components
normalize	boolean	no	False	Center and reduce data
outmatrix	fileDestination	no	—	Transformation matrix output (text format)
bv	number	no	0.0	Background Value

Outputs: out (outputRaster), outinv (outputRaster), method.pca.outeigenvalues (outputFile), outmatrix (outputFile)

Example usage

```
run otb:DimensionalityReduction in=dem.tif rescale=0 rescale.minmax.outmin=0
↪  rescale.minmax.outmax=255 method=0 method.pca.whiten=true method.napca.radiusx=1
↪  out=out.tif
```

This calls **DimensionalityReduction**: in=dem.tif — Input Image; rescale=0 — Rescale Output; rescale.minmax.outmin=0 — Output min value; rescale.minmax.outmax=255 — Output max value; method=0 — Algorithm; method.pca.whiten=true — Perform pca whitening; method.napca.radiusx=1 — Set the x radius of the sliding window; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:DisparityMapToElevationMap — DisparityMapToElevationMap

group: Stereo

Parameter	Type	Required	Default	Description
io.in	raster	yes	—	Input disparity map
io.left	raster	yes	—	Left sensor image
io.right	raster	yes	—	Right sensor image
io.lgrid	raster	yes	—	Left Grid
io.rgrid	raster	yes	—	Right Grid
io.out	rasterDestination	yes	—	Output elevation map
io.mask	raster	no	—	Disparity mask
step	number	no	5.0	DEM step
hmin	number	no	0.0	Minimum elevation expected
hmax	number	no	100.0	Maximum elevation expected
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: io.out (outputRaster)

Example usage

```
run otb:DisparityMapToElevationMap io.in=dem.tif io.left=dem.tif io.right=dem.tif
↪  io.lgrid=dem.tif io.rgrid=dem.tif io.mask=dem.tif step=5 io.out=io.out.tif
```

This calls **DisparityMapToElevationMap**: `io.in=dem.tif` — Input disparity map; `io.left=dem.tif` — Left sensor image; `io.right=dem.tif` — Right sensor image; `io.lgrid=dem.tif` — Left Grid; `io.rgrid=dem.tif` — Right Grid; `io.mask=dem.tif` — Disparity mask; `step=5` — DEM step; `io.out=io.out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:DomainTransform` — DomainTransform

group: Image Filtering

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Output Image
`mode`	OTBParameterChoice	no	1	Mode
`mode.fft.shift`	boolean	no	False	Shift fft transform
`mode.wavelet.form`	OTBParameterChoice	no	0	Select wavelet form
`mode.wavelet.nlevels`	number	no	2.0	Number of decomposition levels
`direction`	OTBParameterChoice	no	0	Direction
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:DomainTransform in=dem.tif mode=1 mode.fft.shift=false mode.wavelet.form=0
↪  mode.wavelet.nlevels=2 direction=0 out=out.tif
```

This calls **DomainTransform**: `in=dem.tif` — Input Image; `mode=1` — Mode; `mode.fft.shift=false` — Shift fft transform; `mode.wavelet.form=0` — Select wavelet form; `mode.wavelet.nlevels=2` — Number of decomposition levels; `direction=0` — Direction; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:DynamicConvert` — DynamicConvert

group: Image Manipulation

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input image
`out`	rasterDestination	yes	—	Output Image
`type`	OTBParameterChoice	no	0	Rescale type
`type.linear.gamma`	number	no	1.0	Gamma correction factor
`mask`	raster	no	—	Input mask
`quantile.high`	number	no	2.0	High cut quantile
`quantile.low`	number	no	2.0	Low cut quantile
`channels`	OTBParameterChoice	no	0	Channels selection
`channels.grayscale.channel`	number	no	1.0	Grayscale channel
`channels.rgb.red`	number	no	0.0	Red Channel
`channels.rgb.green`	number	no	0.0	Green Channel
`channels.rgb.blue`	number	no	0.0	Blue Channel
`outmin`	number	no	0.0	Output min value
`outmax`	number	no	255.0	Output max value
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:DynamicConvert in=dem.tif type=0 type.linear.gamma=1 mask=dem.tif quantile.high=2
↪  quantile.low=2 channels=0 out=out.tif
```

This calls **DynamicConvert**: `in=dem.tif` — Input image; `type=0` — Rescale type; `type.linear.gamma=1` — Gamma correction factor; `mask=dem.tif` — Input mask; `quantile.high=2` — High cut quantile; `quantile.low=2` — Low cut quantile; `channels=0` — Channels selection; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:EdgeExtraction` — EdgeExtraction

group: Feature Extraction

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Feature Output Image
`channel`	number	no	1.0	Selected Channel
`filter`	OTBParameterChoice	no	0	Edge feature
`filter.touzi.xradius`	number	no	1.0	X radius of the neighborhood
`filter.touzi.yradius`	number	no	1.0	Y radius of the neighborhood
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:EdgeExtraction in=dem.tif channel=1 filter=0 filter.touzi.xradius=1
↪  filter.touzi.yradius=1 out=out.tif
```

This calls **EdgeExtraction**: `in=dem.tif` — Input Image; `channel=1` — Selected Channel; `filter=0` — Edge feature; `filter.touzi.xradius=1` — X radius of the neighborhood; `filter.touzi.yradius=1` — Y radius of the neighborhood; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:EndmemberNumberEstimation` — EndmemberNumberEstimation

group: Hyperspectral

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image Filename
`algo`	OTBParameterChoice	no	0	Unmixing algorithm
`algo.vd.far`	number	no	0.001	False alarm rate

Example usage

```
run otb:EndmemberNumberEstimation in=dem.tif algo=0 algo.vd.far=0.001
```

This calls **EndmemberNumberEstimation**: `in=dem.tif` — Input Image Filename; `algo=0` — Unmixing algorithm; `algo.vd.far=0.001` — False alarm rate. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ExtractROI — ExtractROI

group: Image Manipulation

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	rasterDestination	yes	—	Output Image
mode	OTBParameterChoice	no	0	Extraction mode
mode.fit.im	raster	no	—	Reference image
mode.fit.vect	vector	no	—	Reference vector
mode.extent.ulx	number	no	0.0	X coordinate of the Upper left corner
mode.extent.uly	number	no	0.0	Y coordinate of Upper Left corner point
mode.extent.lrx	number	no	0.0	X coordinate of Lower Right corner point
mode.extent.lry	number	no	0.0	Y coordinate of Lower Right corner point
mode.extent.unit	OTBParameterChoice	no	0	Unit
mode.radius.r	number	no	0.0	Radius
mode.radius.unitr	OTBParameterChoice	no	0	Radius unit
mode.radius.cx	number	no	0.0	X coordinate of the center
mode.radius.cy	number	no	0.0	Y coordinate of the center
mode.radius.unitc	OTBParameterChoice	no	0	Center unit
startx	number	no	0.0	Start X
starty	number	no	0.0	Start Y
sizex	number	no	0.0	Size X
sizey	number	no	0.0	Size Y
cl	band	no	—	Output Image channels
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:ExtractROI in=dem.tif mode=0 mode.fit.im=dem.tif mode.fit.vect=input.gpkg
 ↪  mode.extent.ulx=0 mode.extent.uly=0 mode.extent.lrx=0 out=out.tif
```

This calls **ExtractROI**: `in=dem.tif` — Input Image; `mode=0` — Extraction mode; `mode.fit.im=dem.tif` — Reference image; `mode.fit.vect=input.gpkg` — Reference vector; `mode.extent.ulx=0` — X coordinate of the Upper left corner; `mode.extent.uly=0` — Y coordinate of Upper Left corner point; `mode.extent.lrx=0` — X coordinate of Lower Right corner point; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:FastNLMeans — FastNLMeans

group: Image Filtering

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input image
out	rasterDestination	yes	—	Output Image
patchradius	number	no	2.0	Patch radius (patch is a square)
searchradius	number	no	7.0	Search window radius (search window is a square)
sig	number	no	0.0	Standard deviation in image
thresh	number	no	1.0	Similarity threshold

Parameter	Type	Required	Default	Description
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:FastNLMeans in=dem.tif patchradius=2 searchradius=7 sig=0 thresh=1 out=out.tif
```

This calls **FastNLMeans**: in=dem.tif — Input image; patchradius=2 — Patch radius (patch is a square); searchradius=7 — Search window radius (search window is a square); sig=0 — Standard deviation in image; thresh=1 — Similarity threshold; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:FineRegistration — FineRegistration

group: Stereo

Parameter	Type	Required	Default	Description
ref	raster	yes	—	Reference Image
sec	raster	yes	—	Secondary Image
out	rasterDestination	yes	—	Output Image
erx	number	no	0.0	Exploration Radius X
ery	number	no	0.0	Exploration Radius Y
mrx	number	no	0.0	Metric Radius X
mry	number	no	0.0	Metric Radius Y
w	raster	no	—	Image To Warp
wo	rasterDestination	no	—	Output Warped Image
cox	number	no	0.0	Coarse Offset X
coy	number	no	0.0	Coarse Offset Y
ssrx	number	no	1.0	Sub-Sampling Rate X
ssry	number	no	1.0	Sub-Sampling Rate Y
rgsx	number	no	0.0	Reference Gaussian Smoothing X
rgsy	number	no	0.0	Reference Gaussian Smoothing Y
sgsx	number	no	0.0	Secondary Gaussian Smoothing X
sgsy	number	no	0.0	Secondary Gaussian Smoothing Y
m	string	no	—	Metric
spa	number	no	0.01	SubPixelAccuracy
cva	number	no	0.01	ConvergenceAccuracy
vmlt	number	no	0.0	Validity Mask Lower Threshold
vmut	number	no	0.0	Validity Mask Upper Threshold
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster), wo (outputRaster)

Example usage

```
run otb:FineRegistration ref=dem.tif sec=dem.tif erx=0 ery=0 mrx=0 mry=0 w=dem.tif
↪  out=out.tif
```

This calls **FineRegistration**: ref=dem.tif — Reference Image; sec=dem.tif — Secondary Image; erx=0 — Exploration Radius X; ery=0 — Exploration Radius Y; mrx=0 — Metric Radius X; mry=0 — Metric Radius Y; w=dem.tif — Image To Warp; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:FusionOfClassifications — FusionOfClassifications

group: Learning

Parameter	Type	Required	Default	Description
il	multilayer	yes	—	Input classifications
out	rasterDestination	yes	—	The output classification image
method	OTBParameterChoice	no	0	Fusion method
method.dempstershafer.cmfl	multilayer	yes	—	Confusion Matrices
method.dempstershafer.mob	OTBParameterChoice	no	0	Mass of belief measurement
nodatalabel	number	no	0.0	Label for the NoData class
undecidedlabel	number	no	0.0	Label for the Undecided class
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:FusionOfClassifications il="a.gpkg;b.gpkg"
 ↪  method.dempstershafer.cmfl="a.gpkg;b.gpkg" method=0 method.dempstershafer.mob=0
 ↪  nodatalabel=0 undecidedlabel=0 out=out.tif
```

This calls **FusionOfClassifications**: `il="a.gpkg;b.gpkg"` — Input classifications; `method.dempstershafer.cmfl="a.gpkg;b.gpkg"` — Confusion Matrices; `method=0` — Fusion method; `method.dempstershafer.mob=0` — Mass of belief measurement; `nodatalabel=0` — Label for the NoData class; `undecidedlabel=0` — Label for the Undecided class; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:GeneratePlyFile — GeneratePlyFile

group: Geometry

Parameter	Type	Required	Default	Description
indem	raster	yes	—	The input DEM image
mode	OTBParameterChoice	no	0	Conversion Mode
map	OTBParameterChoice	no	0	Map Projection
map.utm.zone	number	no	31.0	Zone number
map.utm.northhem	boolean	no	False	Northern Hemisphere
map.epsg.code	crs	no	ProjectCrs	EPSG Code
incolor	raster	yes	—	The input color image
out	fileDestination	yes	—	The output Ply file

Outputs: out (outputFile)

Example usage

```
run otb:GeneratePlyFile indem=dem.tif incolor=dem.tif mode=0 map=0 map.utm.zone=31
 ↪  map.utm.northhem=false map.epsg.code=EPSG:6346 out=out.txt
```

This calls **GeneratePlyFile**: `indem=dem.tif` — The input DEM image; `incolor=dem.tif` — The input color image; `mode=0` — Conversion Mode; `map=0` — Map Projection; `map.utm.zone=31` — Zone number; `map.utm.northhem=false` — Northern Hemisphere; `map.epsg.code=EPSG:6346` — EPSG Code; `out=out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:GrayScaleMorphologicalOperation — GrayScaleMorphologicalOperation

group: Feature Extraction

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	rasterDestination	yes	—	Feature Output Image
channel	number	no	1.0	Selected Channel
structype	OTBParameterChoice	no	0	Structuring Element Type
xradius	number	no	5.0	Structuring element X radius
yradius	number	no	5.0	Structuring element Y radius
filter	OTBParameterChoice	no	0	Morphological Operation
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:GrayScaleMorphologicalOperation in=dem.tif channel=1 structype=0 xradius=5
↪  yradius=5 filter=0 out=out.tif
```

This calls **GrayScaleMorphologicalOperation**: in=dem.tif — Input Image; channel=1 — Selected Channel; structype=0 — Structuring Element Type; xradius=5 — Structuring element X radius; yradius=5 — Structuring element Y radius; filter=0 — Morphological Operation; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:GridBasedImageResampling — GridBasedImageResampling

group: Geometry

Parameter	Type	Required	Default	Description
io.in	raster	yes	—	Input image
io.out	rasterDestination	yes	—	Output Image
grid.in	raster	yes	—	Input resampling grid
grid.type	OTBParameterChoice	no	0	Grid Type
out.ulx	number	no	0.0	Upper Left X
out.uly	number	no	0.0	Upper Left Y
out.sizex	number	no	0.0	Size X
out.sizey	number	no	0.0	Size Y
out.spacingx	number	no	1.0	Pixel Size X
out.spacingy	number	no	1.0	Pixel Size Y
out.default	number	no	0.0	Default value
interpolator	OTBParameterChoice	no	2	Interpolation
interpolator.bco.radius	number	no	2.0	Radius for bicubic interpolation
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: io.out (outputRaster)

Example usage

```
run otb:GridBasedImageResampling io.in=dem.tif grid.in=dem.tif grid.type=0 out.ulx=0
↪  out.uly=0 out.sizex=0 out.sizey=0 io.out=io.out.tif
```

This calls **GridBasedImageResampling**: `io.in=dem.tif` — Input image; `grid.in=dem.tif` — Input resampling grid; `grid.type=0` — Grid Type; `out.ulx=0` — Upper Left X; `out.uly=0` — Upper Left Y; `out.sizex=0` — Size X; `out.sizey=0` — Size Y; `io.out=io.out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:HaralickTextureExtraction` — HaralickTextureExtraction

group: Feature Extraction

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`channel`	number	no	1.0	Selected Channel
`step`	number	no	1.0	Computation step
`parameters.xrad`	number	no	2.0	X Radius
`parameters.yrad`	number	no	2.0	Y Radius
`parameters.xoff`	number	no	1.0	X Offset
`parameters.yoff`	number	no	1.0	Y Offset
`parameters.min`	number	no	0.0	Image Minimum
`parameters.max`	number	no	255.0	Image Maximum
`parameters.nbbin`	number	no	8.0	Histogram number of bin
`texture`	OTBParameterChoice	no	0	Texture Set Selection
`out`	rasterDestination	no	—	Output Image

Outputs: `out` (outputRaster)

Example usage

```
run otb:HaralickTextureExtraction in=dem.tif channel=1 step=1 parameters.xrad=2
↪  parameters.yrad=2 parameters.xoff=1 parameters.yoff=1 out=out.tif
```

This calls **HaralickTextureExtraction**: `in=dem.tif` — Input Image; `channel=1` — Selected Channel; `step=1` — Computation step; `parameters.xrad=2` — X Radius; `parameters.yrad=2` — Y Radius; `parameters.xoff=1` — X Offset; `parameters.yoff=1` — Y Offset; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:HomologousPointsExtraction` — HomologousPointsExtraction

group: Feature Extraction

Parameter	Type	Required	Default	Description
`in1`	raster	yes	—	Input Image 1
`band1`	number	no	1.0	Input band 1
`in2`	raster	yes	—	Input Image 2
`band2`	number	no	1.0	Input band 2
`out`	fileDestination	yes	—	Output file with tie points
`outvector`	fileDestination	no	—	Output vector file with tie points
`algorithm`	OTBParameterChoice	no	0	Keypoints detection algorithm
`threshold`	number	no	0.6	Distance threshold for matching
`backmatching`	boolean	no	False	Use back-matching to filter matches
`mode`	OTBParameterChoice	no	0	Keypoints search mode
`mode.geobins.binsize`	number	no	256.0	Size of bin
`mode.geobins.binsizey`	number	no	0.0	Size of bin (y direction)
`mode.geobins.binstep`	number	no	256.0	Steps between bins
`mode.geobins.binstepy`	number	no	0.0	Steps between bins (y direction)

Parameter	Type	Required	Default	Description
mode.geobins.margin	number	no	10.0	Margin from image border to start/end bins (in pixels)
precision	number	no	0.0	Estimated precision of the colocalisation function (in pixels)
mfilter	boolean	no	False	Filter points according to geographical or sensor based colocalisation
2wgs84	boolean	no	False	If enabled, points from second image will be exported in WGS84
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation

Outputs: out (outputFile), outvector (outputFile)

Example usage

```
run otb:HomologousPointsExtraction in1=dem.tif in2=dem.tif band1=1 band2=1 algorithm=0
↪  threshold=0.6 backmatching=false out=out.txt
```

This calls **HomologousPointsExtraction**: in1=dem.tif — Input Image 1; in2=dem.tif — Input Image 2; band1=1 — Input band 1; band2=1 — Input band 2; algorithm=0 — Keypoints detection algorithm; threshold=0.6 — Distance threshold for matching; backmatching=false — Use backmatching to filter matches; out=out.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:HooverCompareSegmentation — HooverCompareSegmentation

group: Segmentation

Parameter	Type	Required	Default	Description
ingt	raster	yes	—	Input ground truth
inms	raster	yes	—	Input machine segmentation
bg	number	no	0.0	Background label
th	number	no	0.75	Overlapping threshold
outgt	rasterDestination	no	—	Colored ground truth output
outms	rasterDestination	no	—	Colored machine segmentation output

Outputs: outgt (outputRaster), outms (outputRaster)

Example usage

```
run otb:HooverCompareSegmentation ingt=dem.tif inms=dem.tif bg=0 th=0.75 outgt=outgt.tif
```

This calls **HooverCompareSegmentation**: ingt=dem.tif — Input ground truth; inms=dem.tif — Input machine segmentation; bg=0 — Background label; th=0.75 — Overlapping threshold; outgt=outgt.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:HyperspectralUnmixing — HyperspectralUnmixing

group: Hyperspectral

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image Filename
out	rasterDestination	yes	—	Output Image
ie	raster	yes	—	Input endmembers
ua	OTBParameterChoice	no	0	Unmixing algorithm

Parameter	Type	Required	Default	Description
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:HyperspectralUnmixing in=dem.tif ie=dem.tif ua=0 out=out.tif
```

This calls **HyperspectralUnmixing**: in=dem.tif — Input Image Filename; ie=dem.tif — Input endmembers; ua=0 — Unmixing algorithm; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ImageClassifier — ImageClassifier

group: Learning

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
mask	raster	no	—	Input Mask
model	file	yes	—	Model file
imstat	file	no	—	Statistics file
nodatalabel	number	no	0.0	Label mask value
out	rasterDestination	yes	—	Output Image
confmap	rasterDestination	no	—	Confidence map
probamap	rasterDestination	no	—	Probability map
nbclasses	number	no	20.0	Number of classes in the model
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster), confmap (outputRaster), probamap (outputRaster)

Example usage

```
run otb:ImageClassifier in=dem.tif model=input.dat mask=dem.tif imstat=input.dat
↪  nodatalabel=0 nbclasses=20 out=out.tif
```

This calls **ImageClassifier**: in=dem.tif — Input Image; model=input.dat — Model file; mask=dem.tif — Input Mask; imstat=input.dat — Statistics file; nodatalabel=0 — Label mask value; nbclasses=20 — Number of classes in the model; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ImageDimensionalityReduction — ImageDimensionalityReduction

group: Learning

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
mask	raster	no	—	Input Mask
model	file	yes	—	Model file
imstat	file	no	—	Statistics file
out	rasterDestination	yes	—	Output Image
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:ImageDimensionalityReduction in=dem.tif model=input.dat mask=dem.tif
↪  imstat=input.dat out=out.tif
```

This calls **ImageDimensionalityReduction**: `in=dem.tif` — Input Image; `model=input.dat` — Model file; `mask=dem.tif` — Input Mask; `imstat=input.dat` — Statistics file; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ImageEnvelope` — ImageEnvelope

group: Geometry

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	vectorDestination	yes	—	Output Vector Data
`sr`	number	no	0.0	Sampling Rate
`elev.dem`	file	no	—	DEM directory
`elev.geoid`	file	no	—	Geoid File
`elev.default`	number	no	0.0	Default elevation
`proj`	string	no	—	Projection

Outputs: out (outputVector)

Example usage

```
run otb:ImageEnvelope in=dem.tif sr=0 elev.dem=input.dat elev.geoid=input.dat
↪  elev.default=0 proj=value out=out.gpkg
```

This calls **ImageEnvelope**: `in=dem.tif` — Input Image; `sr=0` — Sampling Rate; `elev.dem=input.dat` — DEM directory; `elev.geoid=input.dat` — Geoid File; `elev.default=0` — Default elevation; `proj=value` — Projection; `out=out.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ImageRegression` — ImageRegression

group: Learning

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`mask`	raster	no	—	Input Mask
`model`	file	yes	—	Model file
`imstat`	file	no	—	Statistics file
`out`	rasterDestination	yes	—	Output Image
`outputpixeltype` *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:ImageRegression in=dem.tif model=input.dat mask=dem.tif imstat=input.dat
↪  out=out.tif
```

This calls **ImageRegression**: `in=dem.tif` — Input Image; `model=input.dat` — Model file; `mask=dem.tif` — Input Mask; `imstat=input.dat` — Statistics file; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:KMeansClassification` — KMeansClassification

group: Learning

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Output Image
`nc`	number	no	5.0	Number of classes
`ts`	number	no	100.0	Training set size
`maxit`	number	no	1000.0	Maximum number of iterations
`centroids.in`	file	no	—	input centroids text file
`centroids.out`	fileDestination	no	—	Output centroids text file
`sampler`	OTBParameterChoice	no	0	Sampler type
`sampler.periodic.jitter`	number	no	0.0	Jitter amplitude
`vm`	raster	no	—	Validity Mask
`nodatalabel`	number	no	0.0	Label mask value
`cleanup`	boolean	no	True	Clean-up of temporary files
`rand`	number	no	0.0	Random seed
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster), `centroids.out` (outputFile)

Example usage

```
run otb:KMeansClassification in=dem.tif nc=5 ts=100 maxit=1000 centroids.in=input.dat
 ↪  sampler=0 sampler.periodic.jitter=0 out=out.tif
```

This calls **KMeansClassification**: `in=dem.tif` — Input Image; `nc=5` — Number of classes; `ts=100` — Training set size; `maxit=1000` — Maximum number of iterations; `centroids.in=input.dat` — input centroids text file; `sampler=0` — Sampler type; `sampler.periodic.jitter=0` — Jitter amplitude; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:LSMSSegmentation` — LSMSSegmentation

group: Segmentation

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Filtered image
`inpos`	raster	no	—	Filtered position image
`out`	rasterDestination	yes	—	Output labeled Image
`spatialr`	number	no	5.0	Spatial radius
`ranger`	number	no	15.0	Range radius
`minsize`	number	no	0.0	Minimum Segment Size
`tilesizex`	number	no	500.0	Size of tiles in pixel (X-axis)
`tilesizey`	number	no	500.0	Size of tiles in pixel (Y-axis)
`tmpdir`	file	no	—	Directory where to write temporary files
`cleanup`	boolean	no	True	Temporary files cleaning
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:LSMSSegmentation in=dem.tif inpos=dem.tif spatialr=5 ranger=15 minsize=0
↪  tilesizex=500 tilesizey=500 out=out.tif
```

This calls **LSMSSegmentation**: `in=dem.tif` — Filtered image; `inpos=dem.tif` — Filtered position image; `spatialr=5` — Spatial radius; `ranger=15` — Range radius; `minsize=0` — Minimum Segment Size; `tilesizex=500` — Size of tiles in pixel (X-axis); `tilesizey=500` — Size of tiles in pixel (Y-axis); `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:LSMSSmallRegionsMerging` — LSMSSmallRegionsMerging

group: Segmentation

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input image
`inseg`	raster	yes	—	Segmented image
`out`	rasterDestination	yes	—	Output Image
`minsize`	number	no	50.0	Minimum Segment Size
`tilesizex`	number	no	500.0	Size of tiles in pixel (X-axis)
`tilesizey`	number	no	500.0	Size of tiles in pixel (Y-axis)
`outputpixeltype` *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:LSMSSmallRegionsMerging in=dem.tif inseg=dem.tif minsize=50 tilesizex=500
↪  tilesizey=500 out=out.tif
```

This calls **LSMSSmallRegionsMerging**: `in=dem.tif` — Input image; `inseg=dem.tif` — Segmented image; `minsize=50` — Minimum Segment Size; `tilesizex=500` — Size of tiles in pixel (X-axis); `tilesizey=500` — Size of tiles in pixel (Y-axis); `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:LSMSVectorization` — LSMSVectorization

group: Segmentation

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`inseg`	raster	yes	—	Segmented image
`out`	fileDestination	yes	—	Output GIS vector file
`tilesizex`	number	no	500.0	Size of tiles in pixel (X-axis)
`tilesizey`	number	no	500.0	Size of tiles in pixel (Y-axis)

Outputs: out (outputFile)

Example usage

```
run otb:LSMSVectorization in=dem.tif inseg=dem.tif tilesizex=500 tilesizey=500 out=out.txt
```

This calls **LSMSVectorization**: `in=dem.tif` — Input Image; `inseg=dem.tif` — Segmented image; `tilesizex=500` — Size of tiles in pixel (X-axis); `tilesizey=500` — Size of tiles in pixel (Y-axis); `out=out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:LargeScaleMeanShift` — LargeScaleMeanShift

group: Segmentation

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`spatialr`	number	no	5.0	Spatial radius
`ranger`	number	no	15.0	Range radius
`minsize`	number	no	50.0	Minimum Segment Size
`tilesizex`	number	no	500.0	Size of tiles in pixel (X-axis)
`tilesizey`	number	no	500.0	Size of tiles in pixel (Y-axis)
`mode`	OTBParameterChoice	no	0	Output mode
`mode.vector.imfield`	raster	no	—	Support image for field computation
`mode.vector.out`	fileDestination	yes	—	Output GIS vector file
`mode.raster.out`	rasterDestination	yes	—	The output raster image
`cleanup`	boolean	no	True	Temporary files cleaning
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `mode.vector.out` (outputFile), `mode.raster.out` (outputRaster)

Example usage

```
run otb:LargeScaleMeanShift in=dem.tif spatialr=5 ranger=15 minsize=50 tilesizex=500
↪  tilesizey=500 mode=0 mode.vector.out=mode.vector.out.txt
```

This calls **LargeScaleMeanShift**: `in=dem.tif` — Input Image; `spatialr=5` — Spatial radius; `ranger=15` — Range radius; `minsize=50` — Minimum Segment Size; `tilesizex=500` — Size of tiles in pixel (X-axis); `tilesizey=500` — Size of tiles in pixel (Y-axis); `mode=0` — Output mode; `mode.vector.out=mode.vector.out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:LineSegmentDetection` — LineSegmentDetection

group: Feature Extraction

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	vectorDestination	yes	—	Output Detected lines
`elev.dem`	file	no	—	DEM directory
`elev.geoid`	file	no	—	Geoid File
`elev.default`	number	no	0.0	Default elevation
`norescale`	boolean	no	False	No rescaling in [0, 255]

Outputs: `out` (outputVector)

Example usage

```
run otb:LineSegmentDetection in=dem.tif elev.dem=input.dat elev.geoid=input.dat
↪  elev.default=0 norescale=false out=out.gpkg
```

This calls **LineSegmentDetection**: `in=dem.tif` — Input Image; `elev.dem=input.dat` — DEM directory; `elev.geoid=input.dat` — Geoid File; `elev.default=0` — Default elevation; `norescale=false` — No rescaling in [0, 255]; `out=out.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:LocalRxDetection` — LocalRxDetection

group: Hyperspectral

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	no	—	Output Image
`ir`	number	no	1.0	Internal radius
`er`	number	no	5.0	External radius

Outputs: `out` (outputRaster)

Example usage

```
run otb:LocalRxDetection in=dem.tif ir=1 er=5 out=out.tif
```

This calls **LocalRxDetection**: `in=dem.tif` — Input Image; `ir=1` — Internal radius; `er=5` — External radius; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:LocalStatisticExtraction` — LocalStatisticExtraction

group: Feature Extraction

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Feature Output Image
`channel`	number	no	1.0	Selected Channel
`radius`	number	no	3.0	Neighborhood radius
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:LocalStatisticExtraction in=dem.tif channel=1 radius=3 out=out.tif
```

This calls **LocalStatisticExtraction**: `in=dem.tif` — Input Image; `channel=1` — Selected Channel; `radius=3` — Neighborhood radius; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ManageNoData` — ManageNoData

group: Image Manipulation

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input image
`out`	rasterDestination	yes	—	Output Image
`usenan`	boolean	no	False	Consider NaN as no-data
`mode`	OTBParameterChoice	no	0	No-data handling mode

Parameter	Type	Required	Default	Description
mode.buildmask.inv	number	no	1.0	Inside Value
mode.buildmask.outv	number	no	0.0	Outside Value
mode.changevalue.newv	number	no	0.0	The new no-data value
mode.apply.mask	raster	yes	—	Mask image
mode.apply.ndval	number	no	0.0	Nodata value used
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:ManageNoData in=dem.tif mode.apply.mask=dem.tif usenan=false mode=0
↪  mode.buildmask.inv=1 mode.buildmask.outv=0 mode.changevalue.newv=0 out=out.tif
```

This calls **ManageNoData**: in=dem.tif — Input image; mode.apply.mask=dem.tif — Mask image; usenan=false — Consider NaN as no-data; mode=0 — No-data handling mode; mode.buildmask.inv=1 — Inside Value; mode.buildmask.outv=0 — Outside Value; mode.changevalue.newv=0 — The new no-data value; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:MeanShiftSmoothing — MeanShiftSmoothing

group: Image Filtering

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
fout	rasterDestination	yes	—	Spectral filtered output
foutpos	rasterDestination	no	—	Spatial filtered displacement output
spatialr	number	no	5.0	Spatial radius
ranger	number	no	15.0	Range radius
thres	number	no	0.1	Mode convergence threshold
maxiter	number	no	100.0	Maximum number of iterations
rangeramp	number	no	0.0	Range radius ramp coefficient
modesearch	boolean	no	False	Mode search
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: fout (outputRaster), foutpos (outputRaster)

Example usage

```
run otb:MeanShiftSmoothing in=dem.tif spatialr=5 ranger=15 thres=0.1 maxiter=100
↪  rangeramp=0 modesearch=false fout=fout.tif
```

This calls **MeanShiftSmoothing**: in=dem.tif — Input Image; spatialr=5 — Spatial radius; ranger=15 — Range radius; thres=0.1 — Mode convergence threshold; maxiter=100 — Maximum number of iterations; rangeramp=0 — Range radius ramp coefficient; modesearch=false — Mode search; fout=fout.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:MorphologicalClassification — MorphologicalClassification

group: Feature Extraction

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Output Image
`channel`	number	no	1.0	Selected Channel
`structype`	OTBParameterChoice	no	0	Structuring Element Type
`radius`	number	no	5.0	Radius
`sigma`	number	no	0.5	Sigma value for leveling tolerance
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:MorphologicalClassification in=dem.tif channel=1 structype=0 radius=5 sigma=0.5
↪  out=out.tif
```

This calls **MorphologicalClassification**: `in=dem.tif` — Input Image; `channel=1` — Selected Channel; `structype=0` — Structuring Element Type; `radius=5` — Radius; `sigma=0.5` — Sigma value for leveling tolerance; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:MorphologicalMultiScaleDecomposition` — MorphologicalMultiScaleDecomposition

group: Feature Extraction

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`outconvex`	rasterDestination	yes	—	Output Convex Image
`outconcave`	rasterDestination	yes	—	Output Concave Image
`outleveling`	rasterDestination	yes	—	Output Image
`channel`	number	no	1.0	Selected Channel
`structype`	OTBParameterChoice	no	0	Structuring Element Type
`radius`	number	no	5.0	Initial radius
`step`	number	no	1.0	Radius step
`levels`	number	no	1.0	Number of levels use for multi scale
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `outconvex` (outputRaster), `outconcave` (outputRaster), `outleveling` (outputRaster)

Example usage

```
run otb:MorphologicalMultiScaleDecomposition in=dem.tif channel=1 structype=0 radius=5
↪  step=1 levels=1 outconvex=outconvex.tif
```

This calls **MorphologicalMultiScaleDecomposition**: `in=dem.tif` — Input Image; `channel=1` — Selected Channel; `structype=0` — Structuring Element Type; `radius=5` — Initial radius; `step=1` — Radius step; `levels=1` — Number of levels use for multi scale; `outconvex=outconvex.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:MorphologicalProfilesAnalysis` — MorphologicalProfilesAnalysis

group: Feature Extraction

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Output Image
`channel`	number	no	1.0	Selected Channel
`structype`	OTBParameterChoice	no	0	Structuring Element Type
`size`	number	no	5.0	Profile Size
`radius`	number	no	5.0	Initial radius
`step`	number	no	1.0	Radius step
`profile`	OTBParameterChoice	no	0	Profile
`profile.classification.sigma`	number	no	1.0	Sigma value for leveling tolerance
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:MorphologicalProfilesAnalysis in=dem.tif channel=1 structype=0 size=5 radius=5
 ↪  step=1 profile=0 out=out.tif
```

This calls **MorphologicalProfilesAnalysis**: `in=dem.tif` — Input Image; `channel=1` — Selected Channel; `structype=0` — Structuring Element Type; `size=5` — Profile Size; `radius=5` — Initial radius; `step=1` — Radius step; `profile=0` — Profile; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:Mosaic` — Mosaic

group: Image Manipulation

Parameter	Type	Required	Default	Description
`il`	multilayer	yes	—	Input Images
`vdcut`	multilayer	no	—	Input VectorData for composition
`vdstats`	multilayer	no	—	Input VectorData for statistics
`comp.feather`	OTBParameterChoice	no	0	Feathering method
`comp.feather.slim.exponent`	number	no	1.0	Transition smoothness (1.0 = linear transition)
`comp.feather.slim.length`	number	no	0.0	Transition length (In cartographic units)
`harmo.method`	OTBParameterChoice	no	0	harmonization method
`harmo.cost`	OTBParameterChoice	no	0	harmonization cost function
`out`	rasterDestination	yes	—	Output image
`interpolator`	OTBParameterChoice	no	0	Interpolation
`interpolator.bco.radius`	number	no	2.0	Radius for bicubic interpolation
`output.spacingx`	number	no	0.0	Pixel Size (X)
`output.spacingy`	number	no	0.0	Pixel Size (Y)
`tmpdir`	file	no	—	Directory for temporary files
`distancemap.sr`	number	no	10.0	Distance maps sampling ratio
`nodata`	number	no	0.0	no-data value
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:Mosaic il="a.gpkg;b.gpkg" vdcut="a.gpkg;b.gpkg" vdstats="a.gpkg;b.gpkg"
 ↪  comp.feather=0 comp.feather.slim.exponent=1 comp.feather.slim.length=0 harmo.method=0
 ↪  out=out.tif
```

This calls **Mosaic**: `il="a.gpkg;b.gpkg"` — Input Images; `vdcut="a.gpkg;b.gpkg"` — Input VectorData for composition; `vdstats="a.gpkg;b.gpkg"` — Input VectorData for statistics; `comp.feather=0` — Feathering method; `comp.feather.slim.exponent=1` — Transition smoothness (1.0 = linear transition); `comp.feather.slim.length=0` — Transition length (In cartographic units); `harmo.method=0` — harmonization method; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:MultiImageSamplingRate` — MultiImageSamplingRate

group: Learning

Parameter	Type	Required	Default	Description
`il`	multilayer	yes	—	Input statistics
`out`	fileDestination	yes	—	Output sampling rates
`strategy`	OTBParameterChoice	no	2	Sampling strategy
`strategy.byclass.in`	multilayer	yes	—	Number of samples by class
`strategy.constant.nb`	string	yes	—	Number of samples for all classes
`strategy.percent.p`	string	yes	—	The percentage(s) to use
`strategy.total.v`	string	yes	—	The number of samples to generate
`mim`	OTBParameterChoice	no	0	Multi-Image Mode

Outputs: `out` (outputFile)

Example usage

```
run otb:MultiImageSamplingRate il="a.gpkg;b.gpkg" strategy.byclass.in="a.gpkg;b.gpkg"
 ↪  strategy.constant.nb=value strategy.percent.p=value strategy.total.v=value strategy=2
 ↪  mim=0 out=out.txt
```

This calls **MultiImageSamplingRate**: `il="a.gpkg;b.gpkg"` — Input statistics; `strategy.byclass.in="a.gpkg;b.gpkg"` — Number of samples by class; `strategy.constant.nb=value` — Number of samples for all classes; `strategy.percent.p=value` — The percentage(s) to use; `strategy.total.v=value` — The number of samples to generate; `strategy=2` — Sampling strategy; `mim=0` — Multi-Image Mode; `out=out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:MultiResolutionPyramid` — MultiResolutionPyramid

group: Image Manipulation

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Output Image
`level`	number	no	1.0	Number Of Levels
`sfactor`	number	no	2.0	Subsampling factor
`vfactor`	number	no	0.6	Variance factor
`fast`	boolean	no	False	Use Fast Scheme
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:MultiResolutionPyramid in=dem.tif level=1 sfactor=2 vfactor=0.6 fast=false
 ↪  out=out.tif
```

This calls **MultiResolutionPyramid**: `in=dem.tif` — Input Image; `level=1` — Number Of Levels; `sfactor=2` — Subsampling factor; `vfactor=0.6` — Variance factor; `fast=false` — Use Fast Scheme; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:MultivariateAlterationDetector` — MultivariateAlterationDetector

group: Change Detection

Parameter	Type	Required	Default	Description
`in1`	raster	yes	—	Input Image 1
`in2`	raster	yes	—	Input Image 2
`out`	rasterDestination	yes	—	Change Map
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:MultivariateAlterationDetector in1=dem.tif in2=dem.tif out=out.tif
```

This calls **MultivariateAlterationDetector**: `in1=dem.tif` — Input Image 1; `in2=dem.tif` — Input Image 2; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:OGRLayerClassifier` — OGRLayerClassifier

group: Segmentation

Parameter	Type	Required	Default	Description
`inshp`	vector	yes	—	Name of the input shapefile
`instats`	file	yes	—	XML file containing mean and variance of each feature
`insvm`	fileDestination	yes	—	Input model filename
`feat`	field	no	—	Features
`cfield`	string	no	—	Field containing the predicted class

Outputs: `insvm` (outputFile)

Example usage

```
run otb:OGRLayerClassifier inshp=input.gpkg instats=input.dat feat=field1 cfield=value
↪  insvm=insvm.txt
```

This calls **OGRLayerClassifier**: `inshp=input.gpkg` — Name of the input shapefile; `instats=input.dat` — XML file containing mean and variance of each feature; `feat=field1` — Features; `cfield=value` — Field containing the predicted class; `insvm=insvm.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ObtainUTMZoneFromGeoPoint` — ObtainUTMZoneFromGeoPoint

group: Geometry

Parameter	Type	Required	Default	Description
`lat`	number	no	0.0	Latitude

Parameter	Type	Required	Default	Description
lon	number	no	0.0	Longitude

Example usage

```
run otb:ObtainUTMZoneFromGeoPoint lat=0 lon=0
```

This calls **ObtainUTMZoneFromGeoPoint**: `lat=0` — Latitude; `lon=0` — Longitude. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:OpticalCalibration — OpticalCalibration

group: Calibration

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input
out	rasterDestination	yes	—	Output
level	OTBParameterChoice	no	0	Calibration Level
milli	boolean	no	False	Convert to milli reflectance
clamp	boolean	no	True	Clamp of reflectivity values between [0, 1]
acqui.minute	number	no	0.0	Minute
acqui.hour	number	no	12.0	Hour
acqui.day	number	no	1.0	Day
acqui.month	number	no	1.0	Month
acqui.year	number	no	2000.0	Year
acqui.fluxnormcoeff	number	no	0.0	Flux Normalization
acqui.solardistance	number	no	0.0	Solar distance
acqui.sun.elev	number	no	90.0	Sun elevation angle (deg)
acqui.sun.azim	number	no	0.0	Sun azimuth angle (deg)
acqui.view.elev	number	no	90.0	Viewing elevation angle (deg)
acqui.view.azim	number	no	0.0	Viewing azimuth angle (deg)
acqui.gainbias	file	no	—	Gains and biases
acqui.solarilluminations	file	no	—	Solar illuminations
atmo.aerosol	OTBParameterChoice	no	0	Aerosol Model
atmo.oz	number	no	0.0	Ozone Amount (cm-atm)
atmo.wa	number	no	2.5	Water Vapor Amount (g/cm2)
atmo.pressure	number	no	1030.0	Atmospheric Pressure (hPa)
atmo.opt	number	no	0.2	Aerosol Optical Thickness
atmo.aeronet	file	no	—	Aeronet File
atmo.rsr	file	no	—	Relative Spectral Response File
atmo.radius	number	no	2.0	Window radius (adjacency effects)
atmo.pixsize	number	no	1.0	Pixel size (in km)
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:OpticalCalibration in=dem.tif level=0 milli=false clamp=true acqui.minute=0
↪  acqui.hour=12 acqui.day=1 out=out.tif
```

This calls **OpticalCalibration**: `in=dem.tif` — Input; `level=0` — Calibration Level; `milli=false` — Convert to milli reflectance; `clamp=true` — Clamp of reflectivity values between [0, 1]; `acqui.minute=0` — Minute; `acqui.hour=12` — Hour; `acqui.day=1` — Day; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:OrthoRectification — OrthoRectification

group: Geometry

Parameter	Type	Required	Default	Description
io.in	raster	yes	—	Input Image
io.out	rasterDestination	yes	—	Output Image
map	OTBParameterChoice	no	0	Map Projection
map.utm.zone	number	no	31.0	Zone number
map.utm.northhem	boolean	no	False	Northern Hemisphere
map.epsg.code	crs	no	ProjectCrs	EPSG Code
outputs.mode	OTBParameterChoice	no	0	Parameters estimation modes
outputs.ulx	number	no	0.0	Upper Left X
outputs.uly	number	no	0.0	Upper Left Y
outputs.sizex	number	no	0.0	Size X
outputs.sizey	number	no	0.0	Size Y
outputs.spacingx	number	no	0.0	Pixel Size X
outputs.spacingy	number	no	0.0	Pixel Size Y
outputs.lrx	number	no	0.0	Lower right X
outputs.lry	number	no	0.0	Lower right Y
outputs.ortho	raster	no	—	Model ortho-image
outputs.isotropic	boolean	no	True	Force isotropic spacing by default
outputs.default	number	no	0.0	Default pixel value
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
interpolator	OTBParameterChoice	no	0	Interpolation
interpolator.bco.radius	number	no	2.0	Radius for bicubic interpolation
opt.rpc	number	no	10.0	RPC modeling (points per axis)
opt.gridspacing	number	no	4.0	Resampling grid spacing
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: io.out (outputRaster)

Example usage

```
run otb:OrthoRectification io.in=dem.tif map=0 map.utm.zone=31 map.utm.northhem=false
↪  map.epsg.code=EPSG:6346 outputs.mode=0 outputs.ulx=0 io.out=io.out.tif
```

This calls **OrthoRectification**: io.in=dem.tif — Input Image; map=0 — Map Projection; map.utm.zone=31 — Zone number; map.utm.northhem=false — Northern Hemisphere; map.epsg.code=EPSG:6346 — EPSG Code; outputs.mode=0 — Parameters estimation modes; outputs.ulx=0 — Upper Left X; io.out=io.out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:Pansharpening — Pansharpening

group: Geometry

Parameter	Type	Required	Default	Description
inp	raster	yes	—	Input PAN Image
inxs	raster	yes	—	Input XS Image
out	rasterDestination	yes	—	Output image
method	OTBParameterChoice	no	0	Algorithm
method.rcs.radiusx	number	no	9.0	X radius
method.rcs.radiusy	number	no	9.0	Y radius
method.lmvm.radiusx	number	no	3.0	X radius

Parameter	Type	Required	Default	Description
method.lmvm.radiusy	number	no	3.0	Y radius
method.bayes.lambda	number	no	0.9999	Weight
method.bayes.s	number	no	1.0	S coefficient
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:Pansharpening inp=dem.tif inxs=dem.tif method=0 method.rcs.radiusx=9
↪  method.rcs.radiusy=9 method.lmvm.radiusx=3 method.lmvm.radiusy=3 out=out.tif
```

This calls **Pansharpening**: inp=dem.tif — Input PAN Image; inxs=dem.tif — Input XS Image; method=0 — Algorithm; method.rcs.radiusx=9 — X radius; method.rcs.radiusy=9 — Y radius; method.lmvm.radiusx=3 — X radius; method.lmvm.radiusy=3 — Y radius; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:PantexTextureExtraction — PantexTextureExtraction

group: Feature Extraction

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
channel	number	no	1.0	Selected Channel
out	rasterDestination	yes	—	Output Image
min	number	no	0.0	Image minimum
max	number	no	0.0	Image maximum
sradx	number	no	4.0	Window radius (x direction)
srady	number	no	4.0	Window radius (y direction)
nbin	number	no	8.0	Number of bins per axis for histogram generation
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:PantexTextureExtraction in=dem.tif channel=1 min=0 max=0 sradx=4 srady=4 nbin=8
↪  out=out.tif
```

This calls **PantexTextureExtraction**: in=dem.tif — Input Image; channel=1 — Selected Channel; min=0 — Image minimum; max=0 — Image maximum; sradx=4 — Window radius (x direction); srady=4 — Window radius (y direction); nbin=8 — Number of bins per axis for histogram generation; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:PixelValue — PixelValue

group: Image Manipulation

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
coordx	number	no	0.0	X coordinate

Parameter	Type	Required	Default	Description
coordy	number	no	0.0	Y coordinate
mode	OTBParameterChoice	no	0	Coordinate system used to designate the pixel
mode.epsg.code	crs	no	ProjectCrs	EPSG code
cl	band	no	—	Channels

Example usage

```
run otb:PixelValue in=dem.tif coordx=0 coordy=0 mode=0 mode.epsg.code=EPSG:6346 cl=1
```

This calls **PixelValue**: `in=dem.tif` — Input Image; `coordx=0` — X coordinate; `coordy=0` — Y coordinate; `mode=0` — Coordinate system used to designate the pixel; `mode.epsg.code=EPSG:6346` — EPSG code; `cl=1` — Channels. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:PolygonClassStatistics` — PolygonClassStatistics

group: Learning

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input image
mask	raster	no	—	Input validity mask
vec	vector	yes	—	Input vectors
out	fileDestination	yes	—	Output XML statistics file
field	field	no	—	Field Name
layer	number	no	0.0	Layer Index
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation

Outputs: `out` (outputFile)

Example usage

```
run otb:PolygonClassStatistics in=dem.tif vec=input.gpkg mask=dem.tif field=field1 layer=0
↪  elev.dem=input.dat elev.geoid=input.dat out=out.txt
```

This calls **PolygonClassStatistics**: `in=dem.tif` — Input image; `vec=input.gpkg` — Input vectors; `mask=dem.tif` — Input validity mask; `field=field1` — Field Name; `layer=0` — Layer Index; `elev.dem=input.dat` — DEM directory; `elev.geoid=input.dat` — Geoid File; `out=out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:Quicklook` — Quicklook

group: Image Manipulation

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	rasterDestination	yes	—	Output Image
cl	band	no	—	Channel List
rox	number	no	0.0	ROI Origin X
roy	number	no	0.0	ROI Origin Y
rsx	number	no	0.0	ROI Size X
rsy	number	no	0.0	ROI Size Y
sr	number	no	2.0	Sampling ratio
sx	number	no	0.0	Size X

Parameter	Type	Required	Default	Description
sy	number	no	0.0	Size Y
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:Quicklook in=dem.tif cl=1 rox=0 roy=0 rsx=0 rsy=0 sr=2 out=out.tif
```

This calls **Quicklook**: in=dem.tif — Input Image; cl=1 — Channel List; rox=0 — ROI Origin X; roy=0 — ROI Origin Y; rsx=0 — ROI Size X; rsy=0 — ROI Size Y; sr=2 — Sampling ratio; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:RadiometricIndices — RadiometricIndices

group: Feature Extraction

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	rasterDestination	yes	—	Output Image
channels.blue	number	no	1.0	Blue Channel
channels.green	number	no	1.0	Green Channel
channels.red	number	no	1.0	Red Channel
channels.nir	number	no	1.0	NIR Channel
channels.mir	number	no	1.0	Mir Channel
list	enum	no	—	Available Radiometric Indices — options: 0=Vegetation:NDVI, 1=Vegetation:TNDVI, 2=Vegetation:RVI, 3=Vegetation:SAVI, 4=Vegetation:TSAVI, 5=Vegetation:MSAVI, 6=Vegetation:MSAVI2, 7=Vegetation:GEMI, 8=Vegetation:IPVI, 9=Vegetation:LAIFromNDVILog, 10=Vegetation:LAIFromReflLinear, 11=Vegetation:LAIFromNDVIFormo, 12=Water:NDWI, 13=Water:NDWI2, 14=Water:MNDWI, 15=Water:NDTI, 16=Soil:RI, 17=Soil:CI, 18=Soil:BI, 19=Soil:BI2, 20=BuiltUp:ISU
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:RadiometricIndices in=dem.tif channels.blue=1 channels.green=1 channels.red=1
↪  channels.nir=1 channels.mir=1 list=0 out=out.tif
```

This calls **RadiometricIndices**: in=dem.tif — Input Image; channels.blue=1 — Blue Channel; channels.green=1 — Green Channel; channels.red=1 — Red Channel; channels.nir=1 — NIR Channel; channels.mir=1 — Mir Channel; list=0 selects *Vegetation:NDVI*; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:Rasterization — Rasterization

group: Vector Data Manipulation

Parameter	Type	Required	Default	Description
`in`	vector	yes	—	Input vector dataset
`out`	rasterDestination	yes	—	Output image
`im`	raster	no	—	Input reference image
`szx`	number	no	0.0	Output size x
`szy`	number	no	0.0	Output size y
`epsg`	crs	no	ProjectCrs	Output EPSG code
`orx`	number	no	0.0	Output Upper-left x
`ory`	number	no	0.0	Output Upper-left y
`spx`	number	no	0.0	Spacing (GSD) x
`spy`	number	no	0.0	Spacing (GSD) y
`background`	number	no	0.0	Background value
`mode`	OTBParameterChoice	no	0	Rasterization mode
`mode.binary.foreground`	number	no	255.0	Foreground value
`mode.attribute.field`	string	no	—	The attribute field to burn
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:Rasterization in=input.gpkg im=dem.tif szx=0 szy=0 epsg=EPSG:6346 orx=0 ory=0
↪  out=out.tif
```

This calls **Rasterization**: `in=input.gpkg` — Input vector dataset; `im=dem.tif` — Input reference image; `szx=0` — Output size x; `szy=0` — Output size y; `epsg=EPSG:6346` — Output EPSG code; `orx=0` — Output Upper-left x; `ory=0` — Output Upper-left y; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ReadImageInfo` — ReadImageInfo

group: Image Manipulation

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`imagemetadata`	boolean	no	False	Display the image metadata
`outgeom`	fileDestination	no	—	Write the image metadata to a geom file

Outputs: `outgeom` (outputFile)

Example usage

```
run otb:ReadImageInfo in=dem.tif imagemetadata=false outgeom=outgeom.txt
```

This calls **ReadImageInfo**: `in=dem.tif` — Input Image; `imagemetadata=false` — Display the image metadata; `outgeom=outgeom.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:RefineSensorModel` — RefineSensorModel

group: Geometry

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input image file
`out`	rasterDestination	yes	—	Output image file

Parameter	Type	Required	Default	Description
inpoints	file	yes	—	Input file containing tie points
outstat	fileDestination	no	—	Output file containing output precision statistics
outvector	fileDestination	no	—	Output vector file with residues
map	OTBParameterChoice	no	0	Map Projection
map.utm.zone	number	no	31.0	Zone number
map.utm.northhem	boolean	no	False	Northern Hemisphere
map.epsg.code	crs	no	ProjectCrs	EPSG Code
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster), outstat (outputFile), outvector (outputFile)

Example usage

```
run otb:RefineSensorModel in=dem.tif inpoints=input.dat map=0 map.utm.zone=31
 ↪  map.utm.northhem=false map.epsg.code=EPSG:6346 elev.dem=input.dat out=out.tif
```

This calls **RefineSensorModel**: in=dem.tif — Input image file; inpoints=input.dat — Input file containing tie points; map=0 — Map Projection; map.utm.zone=31 — Zone number; map.utm.northhem=false — Northern Hemisphere; map.epsg.code=EPSG:6346 — EPSG Code; elev.dem=input.dat — DEM directory; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:ResetMargin — ResetMargin

group: Image Manipulation

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input image
out	rasterDestination	yes	—	Output Image
threshold.y.start	number	no	0.0	Top line index threshold
threshold.y.end	number	no	0.0	Bottom line index threshold
threshold.x	number	no	0.0	Column index threshold
roi.startx	number	no	0.0	X start index
roi.starty	number	no	0.0	Y start index
roi.sizex	number	no	0.0	X size
roi.sizey	number	no	0.0	Y size
margin.top	number	no	0.0	Top margin
margin.bottom	number	no	0.0	Bottom margin
margin.left	number	no	0.0	Left margin
margin.right	number	no	0.0	Right margin
mode	OTBParameterChoice	no	0	Region mode
fillval	number	no	0.0	Padding fill value
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:ResetMargin in=dem.tif threshold.y.start=0 threshold.y.end=0 threshold.x=0
 ↪  roi.startx=0 roi.starty=0 roi.sizex=0 out=out.tif
```

This calls **ResetMargin**: `in=dem.tif` — Input image; `threshold.y.start=0` — Top line index threshold; `threshold.y.end=0` — Bottom line index threshold; `threshold.x=0` — Column index threshold; `roi.startx=0` — X start index; `roi.starty=0` — Y start index; `roi.sizex=0` — X size; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:RigidTransformResample` — RigidTransformResample

group: Geometry

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input image
`out`	rasterDestination	yes	—	Output image
`transform.type`	OTBParameterChoice	no	0	Type of transformation
`transform.type.id.scalex`	number	no	1.0	X scaling
`transform.type.id.scaley`	number	no	1.0	Y scaling
`transform.type.translation.tx`	number	no	0.0	The X translation (in physical units)
`transform.type.translation.ty`	number	no	0.0	The Y translation (in physical units)
`transform.type.translation.scalex`	number	no	1.0	X scaling
`transform.type.translation.scaley`	number	no	1.0	Y scaling
`transform.type.rotation.angle`	number	no	0.0	Rotation angle
`transform.type.rotation.scalex`	number	no	1.0	X scaling
`transform.type.rotation.scaley`	number	no	1.0	Y scaling
`interpolator`	OTBParameterChoice	no	2	Interpolation
`interpolator.bco.radius`	number	no	2.0	Radius for bicubic interpolation
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:RigidTransformResample in=dem.tif transform.type=0 transform.type.id.scalex=1
↪ transform.type.id.scaley=1 transform.type.translation.tx=0
↪ transform.type.translation.ty=0 transform.type.translation.scalex=1 out=out.tif
```

This calls **RigidTransformResample**: `in=dem.tif` — Input image; `transform.type=0` — Type of transformation; `transform.type.id.scalex=1` — X scaling; `transform.type.id.scaley=1` — Y scaling; `transform.type.translation.tx=0` — The X translation (in physical units); `transform.type.translation.ty=0` — The Y translation (in physical units); `transform.type.translation.scalex=1` — X scaling; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:SARBurstExtraction` — SARBurstExtraction

group: SAR

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Sentinel1 IW SLC Image
`burstindex`	number	no	-1.0	Index of Burst
`out`	rasterDestination	yes	—	Output Image
`allpixels`	boolean	no	False	Select the modes for output image
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:SARBurstExtraction in=dem.tif burstindex=-1 allpixels=false out=out.tif
```

This calls **SARBurstExtraction**: `in=dem.tif` — Input Sentinel1 IW SLC Image; `burstindex=-1` — Index of Burst; `allpixels=false` — Select the modes for output image; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:SARCalibration` — SARCalibration

group: Calibration

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`out`	rasterDestination	yes	—	Output Image
`removenoise`	boolean	no	False	Remove Noise
`lut`	OTBParameterChoice	no	0	Lookup table
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:SARCalibration in=dem.tif removenoise=false lut=0 out=out.tif
```

This calls **SARCalibration**: `in=dem.tif` — Input Image; `removenoise=false` — Remove Noise; `lut=0` — Lookup table; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:SARConcatenateBursts` — SARConcatenateBursts

group: SAR

Parameter	Type	Required	Default	Description
`il`	multilayer	yes	—	Input bursts list
`insar`	raster	yes	—	Input Sentinel1 IW SLC Image (only metadata used)
`out`	rasterDestination	yes	—	Output Image
`burstindex`	number	no	0.0	Index of the first Burst
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:SARConcatenateBursts il="a.gpkg;b.gpkg" insar=dem.tif burstindex=0 out=out.tif
```

This calls **SARConcatenateBursts**: `il="a.gpkg;b.gpkg"` — Input bursts list; `insar=dem.tif` — Input Sentinel1 IW SLC Image (only metadata used); `burstindex=0` — Index of the first Burst; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SARDeburst — SARDeburst

group: SAR

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Sentinel1 IW SLC Image
out	rasterDestination	yes	—	Output Image
onlyvalidsamples	boolean	no	False	Select the modes for output image
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:SARDeburst in=dem.tif onlyvalidsamples=false out=out.tif
```

This calls **SARDeburst**: in=dem.tif — Input Sentinel1 IW SLC Image; onlyvalidsamples=false — Select the modes for output image; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SARDecompositions — SARDecompositions

group: SAR

Parameter	Type	Required	Default	Description
inhh	raster	yes	—	Input Image
inhv	raster	no	—	Input Image
invh	raster	no	—	Input Image
invv	raster	yes	—	Input Image
out	rasterDestination	yes	—	Output Image
decomp	OTBParameterChoice	no	0	Decompositions
inco.kernelsize	number	no	3.0	Kernel size for spatial incoherent averaging
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:SARDecompositions inhh=dem.tif invv=dem.tif inhv=dem.tif invh=dem.tif decomp=0
↪  inco.kernelsize=3 out=out.tif
```

This calls **SARDecompositions**: inhh=dem.tif — Input Image; invv=dem.tif — Input Image; inhv=dem.tif — Input Image; invh=dem.tif — Input Image; decomp=0 — Decompositions; inco.kernelsize=3 — Kernel size for spatial incoherent averaging; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SARPolarMatrixConvert — SARPolarMatrixConvert

group: SAR

Parameter	Type	Required	Default	Description
inc	raster	no	—	Input multi-band complex image
inf	raster	no	—	Input multi-band real image
inhh	raster	no	—	Input one-band complex image (HH)

Parameter	Type	Required	Default	Description
inhv	raster	no	—	Input one-band complex image (HV)
invh	raster	no	—	Input one-band complex image (VH)
invv	raster	no	—	Input one-band complex image (VV)
outc	rasterDestination	no	—	Output Complex Image
outf	rasterDestination	no	—	Output Real Image
conv	OTBParameterChoice	no	0	Conversion

Outputs: outc (outputRaster), outf (outputRaster)

Example usage

```
run otb:SARPolarMatrixConvert inc=dem.tif inf=dem.tif inhh=dem.tif inhv=dem.tif
↪  invh=dem.tif invv=dem.tif conv=0 outc=outc.tif
```

This calls **SARPolarMatrixConvert**: inc=dem.tif — Input multi-band complex image; inf=dem.tif — Input multi-band real image; inhh=dem.tif — Input one-band complex image (HH); inhv=dem.tif — Input one-band complex image (HV); invh=dem.tif — Input one-band complex image (VH); invv=dem.tif — Input one-band complex image (VV); conv=0 — Conversion; outc=outc.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SARPolarSynth — SARPolarSynth

group: SAR

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	rasterDestination	yes	—	Output Image
psii	number	no	0.0	psii
khii	number	no	0.0	khii
psir	number	no	0.0	psir
khir	number	no	0.0	khir
emissionh	number	no	0.0	Emission H
emissionv	number	no	0.0	Emission V
mode	OTBParameterChoice	no	0	Forced mode
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:SARPolarSynth in=dem.tif psii=0 khii=0 psir=0 khir=0 emissionh=0 emissionv=0
↪  out=out.tif
```

This calls **SARPolarSynth**: in=dem.tif — Input Image; psii=0 — psii; khii=0 — khii; psir=0 — psir; khir=0 — khir; emissionh=0 — Emission H; emissionv=0 — Emission V; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SFSTextureExtraction — SFSTextureExtraction

group: Feature Extraction

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image

Parameter	Type	Required	Default	Description
channel	number	no	1.0	Selected Channel
parameters.spethre	number	no	50.0	Spectral Threshold
parameters.spathre	number	no	100.0	Spatial Threshold
parameters.nbdir	number	no	20.0	Number of Direction
parameters.alpha	number	no	1.0	Alpha
parameters.maxcons	number	no	5.0	Ratio Maximum Consideration Number
out	rasterDestination	yes	—	Feature Output Image
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:SFSTextureExtraction in=dem.tif channel=1 parameters.spethre=50
↪  parameters.spathre=100 parameters.nbdir=20 parameters.alpha=1 parameters.maxcons=5
↪  out=out.tif
```

This calls **SFSTextureExtraction**: in=dem.tif — Input Image; channel=1 — Selected Channel; parameters.spethre=50 — Spectral Threshold; parameters.spathre=100 — Spatial Threshold; parameters.nbdir=20 — Number of Direction; parameters.alpha=1 — Alpha; parameters.maxcons=5 — Ratio Maximum Consideration Number; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SOMClassification — SOMClassification

group: Learning

Parameter	Type	Required	Default	Description
in	raster	yes	—	InputImage
out	rasterDestination	yes	—	OutputImage
vm	raster	no	—	ValidityMask
tp	number	no	1.0	TrainingProbability
ts	number	no	0.0	TrainingSetSize
som	rasterDestination	no	—	SOM Map
sx	number	no	32.0	SizeX
sy	number	no	32.0	SizeY
nx	number	no	10.0	NeighborhoodX
ny	number	no	10.0	NeighborhoodY
ni	number	no	5.0	NumberIteration
bi	number	no	1.0	BetaInit
bf	number	no	0.1	BetaFinal
iv	number	no	0.0	InitialValue
rand	number	no	0.0	Random seed
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster), som (outputRaster)

Example usage

```
run otb:SOMClassification in=dem.tif vm=dem.tif tp=1 ts=0 sx=32 sy=32 nx=10 out=out.tif
```

This calls **SOMClassification**: in=dem.tif — InputImage; vm=dem.tif — ValidityMask; tp=1 — TrainingProbability; ts=0 — TrainingSetSize; sx=32 — SizeX; sy=32 — SizeY; nx=10 — NeighborhoodX; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:SampleAugmentation` — SampleAugmentation

group: Learning

Parameter	Type	Required	Default	Description
`in`	vector	yes	—	Input samples
`out`	fileDestination	yes	—	Output samples
`field`	field	no	—	Field Name
`layer`	number	no	0.0	Layer Index
`label`	number	no	1.0	Label of the class to be augmented
`samples`	number	no	100.0	Number of generated samples
`exclude`	field	no	—	Field names for excluded features
`strategy`	OTBParameterChoice	no	0	Augmentation strategy
`strategy.jitter.stdfactor`	number	no	10.0	Factor for dividing the standard deviation of each feature
`strategy.smote.neighbors`	number	no	5.0	Number of nearest neighbors
`seed`	number	no	0.0	Random seed

Outputs: `out` (outputFile)

Example usage

```
run otb:SampleAugmentation in=input.gpkg field=field1 layer=0 label=1 samples=100
↪  exclude=field1 strategy=0 out=out.txt
```

This calls **SampleAugmentation**: `in=input.gpkg` — Input samples; `field=field1` — Field Name; `layer=0` — Layer Index; `label=1` — Label of the class to be augmented; `samples=100` — Number of generated samples; `exclude=field1` — Field names for excluded features; `strategy=0` — Augmentation strategy; `out=out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:SampleExtraction` — SampleExtraction

group: Learning

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	InputImage
`vec`	vector	yes	—	Input sampling positions
`out`	fileDestination	no	—	Output samples
`outfield`	OTBParameterChoice	no	0	Output field names
`outfield.prefix.name`	string	no	—	Output field prefix
`outfield.list.names`	string	no	—	Output field names
`field`	field	no	—	Field Name
`layer`	number	no	0.0	Layer Index

Outputs: `out` (outputFile)

Example usage

```
run otb:SampleExtraction in=dem.tif vec=input.gpkg outfield=0 outfield.prefix.name=value
↪  outfield.list.names=value field=field1 layer=0 out=out.txt
```

This calls **SampleExtraction**: `in=dem.tif` — InputImage; `vec=input.gpkg` — Input sampling positions; `outfield=0` — Output field names; `outfield.prefix.name=value` — Output field prefix; `outfield.list.names=value` — Output field names; `field=field1` — Field Name; `layer=0` — Layer Index; `out=out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SampleSelection — SampleSelection

group: Learning

Parameter	Type	Required	Default	Description
in	raster	yes	—	InputImage
mask	raster	no	—	InputMask
vec	vector	yes	—	Input vectors
out	fileDestination	yes	—	Output vectors
instats	file	yes	—	Input Statistics
outrates	fileDestination	no	—	Output rates
sampler	OTBParameterChoice	no	0	Sampler type
sampler.periodic.jitter	number	no	0.0	Jitter amplitude
strategy	OTBParameterChoice	no	4	Sampling strategy
strategy.byclass.in	file	yes	—	Number of samples by class
strategy.constant.nb	number	no	0.0	Number of samples for all classes
strategy.percent.p	number	no	0.5	The percentage to use
strategy.total.v	number	no	1000.0	The number of samples to generate
field	field	no	—	Field Name
layer	number	no	0.0	Layer Index
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
rand	number	no	0.0	Random seed

Outputs: out (outputFile), outrates (outputFile)

Example usage

```
run otb:SampleSelection in=dem.tif vec=input.gpkg instats=input.dat
 ↪  strategy.byclass.in=input.dat mask=dem.tif sampler=0 sampler.periodic.jitter=0
 ↪  out=out.txt
```

This calls **SampleSelection**: in=dem.tif — InputImage; vec=input.gpkg — Input vectors; instats=input.dat — Input Statistics; strategy.byclass.in=input.dat — Number of samples by class; mask=dem.tif — InputMask; sampler=0 — Sampler type; sampler.periodic.jitter=0 — Jitter amplitude; out=out.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:Segmentation — Segmentation

group: Segmentation

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
filter	OTBParameterChoice	no	0	Segmentation algorithm
filter.meanshift.spatialr	number	no	5.0	Spatial radius
filter.meanshift.ranger	number	no	15.0	Range radius
filter.meanshift.thres	number	no	0.1	Mode convergence threshold
filter.meanshift.maxiter	number	no	100.0	Maximum number of iterations
filter.meanshift.minsize	number	no	100.0	Minimum region size
filter.cc.expr	string	yes	—	Condition
filter.watershed.threshold	number	no	0.01	Depth Threshold
filter.watershed.level	number	no	0.1	Flood Level
filter.mprofiles.size	number	no	5.0	Profile Size
filter.mprofiles.start	number	no	1.0	Initial radius
filter.mprofiles.step	number	no	1.0	Radius step
filter.mprofiles.sigma	number	no	1.0	Threshold of the final decision rule

Parameter	Type	Required	Default	Description
mode	OTBParameterChoice	no	0	Processing mode
mode.vector.out	fileDestination	yes	—	Output vector file
mode.vector.outmode	OTBParameterChoice	no	0	Writing mode for the output vector file
mode.vector.inmask	raster	no	—	Mask Image
mode.vector.neighbor	boolean	no	False	8-neighbor connectivity
mode.vector.stitch	boolean	no	True	Stitch polygons
mode.vector.minsize	number	no	1.0	Minimum object size
mode.vector.simplify	number	no	0.1	Simplify polygons
mode.vector.layername	string	no	—	Layer name
mode.vector.fieldname	string	no	—	Geometry index field name
mode.vector.tilesize	number	no	1024.0	Tiles size
mode.vector.startlabel	number	no	1.0	Starting geometry index
mode.vector.ogroptions	string	no	—	OGR options for layer creation
mode.raster.out	rasterDestination	yes	—	Output labeled image
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: mode.vector.out (outputFile), mode.raster.out (outputRaster)

Example usage

```
run otb:Segmentation in=dem.tif filter.cc.expr=value filter=0 filter.meanshift.spatialr=5
↪ filter.meanshift.ranger=15 filter.meanshift.thres=0.1 filter.meanshift.maxiter=100
↪ mode.vector.out=mode.vector.out.txt
```

This calls **Segmentation**: in=dem.tif — Input Image; filter.cc.expr=value — Condition; filter=0 — Segmentation algorithm; filter.meanshift.spatialr=5 — Spatial radius; filter.meanshift.ranger=15 — Range radius; filter.meanshift.thres=0.1 — Mode convergence threshold; filter.meanshift.maxiter=100 — Maximum number of iterations; mode.vector.out=mode.vector.out.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SmallRegionsMerging — SmallRegionsMerging

group: Segmentation

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input image
inseg	raster	yes	—	Segmented image
out	rasterDestination	yes	—	Output Image
minsize	number	no	50.0	Minimum Segment Size
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:SmallRegionsMerging in=dem.tif inseg=dem.tif minsize=50 out=out.tif
```

This calls **SmallRegionsMerging**: in=dem.tif — Input image; inseg=dem.tif — Segmented image; minsize=50 — Minimum Segment Size; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:Smoothing — Smoothing

group: Image Filtering

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	rasterDestination	yes	—	Output Image
type	OTBParameterChoice	no	2	Smoothing Type
type.mean.radius	number	no	2.0	Radius
type.gaussian.stdev	number	no	2.0	Standard deviation
type.gaussian.maxerror	number	no	0.01	Maximum error
type.gaussian.maxwidth	number	no	32.0	Maximum kernel width
type.anidif.timestep	number	no	0.125	Time Step
type.anidif.nbiter	number	no	10.0	Nb Iterations
type.anidif.conductance	number	no	1.0	Conductance
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:Smoothing in=dem.tif type=2 type.mean.radius=2 type.gaussian.stdev=2
↪  type.gaussian.maxerror=0.01 type.gaussian.maxwidth=32 type.anidif.timestep=0.125
↪  out=out.tif
```

This calls **Smoothing**: in=dem.tif — Input Image; type=2 — Smoothing Type; type.mean.radius=2 — Radius; type.gaussian.stdev=2 — Standard deviation; type.gaussian.maxerror=0.01 — Maximum error; type.gaussian.maxwidth=32 — Maximum kernel width; type.anidif.timestep=0.125 — Time Step; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SpectralAngleClassification — SpectralAngleClassification

group: Hyperspectral

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image Filename
ie	raster	yes	—	Input endmembers
measure	rasterDestination	no	—	Output spectral angle values
out	rasterDestination	no	—	Output classified image
mode	OTBParameterChoice	no	0	Measure used for classification
threshold	number	no	0.0	Classification threshold
bv	number	no	0.0	Background value

Outputs: measure (outputRaster), out (outputRaster)

Example usage

```
run otb:SpectralAngleClassification in=dem.tif ie=dem.tif mode=0 threshold=0 bv=0
↪  measure=measure.tif
```

This calls **SpectralAngleClassification**: in=dem.tif — Input Image Filename; ie=dem.tif — Input endmembers; mode=0 — Measure used for classification; threshold=0 — Classification threshold; bv=0 — Background value; measure=measure.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:SplitImage — SplitImage

group: Image Manipulation

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
out	rasterDestination	yes	—	Output Image
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:SplitImage in=dem.tif out=out.tif
```

This calls **SplitImage**: in=dem.tif — Input Image; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:StereoFramework — StereoFramework

group: Stereo

Parameter	Type	Required	Default	Description
input.il	multilayer	yes	—	Input images list
input.co	string	no	—	Couples list
input.channel	number	no	1.0	Input Image channel
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
output.res	number	no	1.0	Output resolution
output.nodata	number	no	-32768.0	NoData value
output.fusionmethod	OTBParameterChoice	no	0	Method to fuse measures in each DSM cell
output.out	rasterDestination	yes	—	Output DSM
output.mode	OTBParameterChoice	no	0	Parameters estimation modes
output.mode.user.ulx	number	no	0.0	Upper Left X
output.mode.user.uly	number	no	0.0	Upper Left Y
output.mode.user.sizex	number	no	0.0	Size X
output.mode.user.sizey	number	no	0.0	Size Y
output.mode.user.spacingx	number	no	0.0	Pixel Size X
output.mode.user.spacingy	number	no	0.0	Pixel Size Y
map	OTBParameterChoice	no	3	Map Projection
map.utm.zone	number	no	31.0	Zone number
map.utm.northhem	boolean	no	False	Northern Hemisphere
map.epsg.code	crs	no	ProjectCrs	EPSG Code
stereorect.fwdgridstep	number	no	16.0	Step of the displacement grid (in pixels)
stereorect.invgridssrate	number	no	10.0	Sub-sampling rate for epipolar grid inversion
bm.metric	OTBParameterChoice	no	0	Block-matching metric
bm.metric.lp.p	number	no	1.0	p value
bm.radius	number	no	2.0	Correlation window radius (in pixels)
bm.minhoffset	number	no	-20.0	Minimum altitude offset (in meters)
bm.maxhoffset	number	no	20.0	Maximum altitude offset (in meters)
postproc.bij	boolean	no	True	Use bijection consistency in block matching strategy
postproc.med	boolean	no	False	Use median disparities filtering
postproc.metrict	number	no	0.6	Correlation metric threshold
mask.left	raster	no	—	Input left mask
mask.right	raster	no	—	Input right mask

Parameter	Type	Required	Default	Description
mask.variancet	number	no	50.0	Discard pixels with low local variance
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: output.out (outputRaster)

Example usage

```
run otb:StereoFramework input.il="a.gpkg;b.gpkg" input.co=value input.channel=1
 ↪  elev.dem=input.dat elev.geoid=input.dat elev.default=0 output.res=1
 ↪  output.out=output.out.tif
```

This calls **StereoFramework**: input.il="a.gpkg;b.gpkg" — Input images list; input.co=value — Couples list; input.channel=1 — Input Image channel; elev.dem=input.dat — DEM directory; elev.geoid=input.dat — Geoid File; elev.default=0 — Default elevation; output.res=1 — Output resolution; output.out=output.out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:StereoRectificationGridGenerator — StereoRectificationGridGenerator

group: Stereo

Parameter	Type	Required	Default	Description
io.inleft	raster	yes	—	Left input image
io.inright	raster	yes	—	Right input image
io.outleft	rasterDestination	yes	—	Left output deformation grid
io.outright	rasterDestination	yes	—	Right output deformation grid
epi.elevation.dem	file	no	—	DEM directory
epi.elevation.geoid	file	no	—	Geoid File
epi.elevation.default	number	no	0.0	Default elevation
epi.elevation.avgdem.step	number	no	1.0	Sub-sampling step
epi.scale	number	no	1.0	Scale of epipolar images
epi.step	number	no	1.0	Step of the deformation grid (in number of pixels)
inverse.outleft	rasterDestination	no	—	Left inverse deformation grid
inverse.outright	rasterDestination	no	—	Right inverse deformation grid
inverse.ssrate	number	no	16.0	Sub-sampling rate for inversion
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: io.outleft (outputRaster), io.outright (outputRaster), inverse.outleft (outputRaster), inverse.outright (outputRaster)

Example usage

```
run otb:StereoRectificationGridGenerator io.inleft=dem.tif io.inright=dem.tif
 ↪  epi.elevation.dem=input.dat epi.elevation.geoid=input.dat epi.elevation.default=0
 ↪  epi.elevation.avgdem.step=1 epi.scale=1 io.outleft=io.outleft.tif
```

This calls **StereoRectificationGridGenerator**: io.inleft=dem.tif — Left input image; io.inright=dem.tif — Right input image; epi.elevation.dem=input.dat — DEM directory; epi.elevation.geoid=input.dat — Geoid File; epi.elevation.default=0 — Default elevation; epi.elevation.avgdem.step=1 — Sub-sampling step; epi.scale=1 — Scale of epipolar images; io.outleft=io.outleft.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:Superimpose — Superimpose

group: Geometry

Parameter	Type	Required	Default	Description
inr	raster	yes	—	Reference input
inm	raster	yes	—	The image to reproject
out	rasterDestination	yes	—	Output image
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
lms	number	no	4.0	Spacing of the deformation field
fv	number	no	0.0	Fill Value
mode	OTBParameterChoice	no	0	Mode
interpolator	OTBParameterChoice	no	0	Interpolation
interpolator.bco.radius	number	no	2.0	Radius for bicubic interpolation
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:Superimpose inr=dem.tif inm=dem.tif elev.dem=input.dat elev.geoid=input.dat
↪  elev.default=0 lms=4 fv=0 out=out.tif
```

This calls **Superimpose**: inr=dem.tif — Reference input; inm=dem.tif — The image to reproject; elev.dem=input.dat — DEM directory; elev.geoid=input.dat — Geoid File; elev.default=0 — Default elevation; lms=4 — Spacing of the deformation field; fv=0 — Fill Value; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:Synthetize — Synthetize

group: Image Manipulation

Parameter	Type	Required	Default	Description
il	string	no	—	Input images list
out	rasterDestination	yes	—	Output Image
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: out (outputRaster)

Example usage

```
run otb:Synthetize il=value out=out.tif
```

This calls **Synthetize**: il=value — Input images list; out=out.tif is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:TileFusion — TileFusion

group: Image Manipulation

Parameter	Type	Required	Default	Description
`il`	multilayer	yes	—	Input Tile Images
`cols`	number	no	0.0	Number of tile columns
`rows`	number	no	0.0	Number of tile rows
`out`	rasterDestination	yes	—	Output Image
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out` (outputRaster)

Example usage

```
run otb:TileFusion il="a.gpkg;b.gpkg" cols=0 rows=0 out=out.tif
```

This calls **TileFusion**: `il="a.gpkg;b.gpkg"` — Input Tile Images; `cols=0` — Number of tile columns; `rows=0` — Number of tile rows; `out=out.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:TrainDimensionalityReduction` — TrainDimensionalityReduction

group: Learning

Parameter	Type	Required	Default	Description
`io.vd`	vector	yes	—	Input Vector Data
`io.out`	fileDestination	yes	—	Output model
`io.stats`	file	no	—	Input XML image statistics file
`feat`	field	no	—	Field names to be used for training
`algorithm`	OTBParameterChoice	no	0	algorithm to use for the training
`algorithm.som.s`	string	no	—	Map size
`algorithm.som.n`	string	no	—	Neighborhood sizes
`algorithm.som.ni`	number	no	5.0	NumberIteration
`algorithm.som.bi`	number	no	1.0	BetaInit
`algorithm.som.bf`	number	no	0.1	BetaFinal
`algorithm.som.iv`	number	no	10.0	InitialValue
`algorithm.autoencoder.nbiter`	number	no	100.0	Maximum number of iterations during training
`algorithm.autoencoder.nbiterfinetuning`	number	no	0.0	Maximum number of iterations during training
`algorithm.autoencoder.epsilon`	number	no	0.0	Epsilon
`algorithm.autoencoder.initfactor`	number	no	1.0	Weight initialization factor
`algorithm.autoencoder.nbneuron`	string	no	—	Size
`algorithm.autoencoder.regularization`	string	no	—	Strength of the regularization
`algorithm.autoencoder.noise`	string	no	—	Strength of the noise
`algorithm.autoencoder.rho`	string	no	—	Sparsity parameter
`algorithm.autoencoder.beta`	string	no	—	Sparsity regularization strength
`algorithm.autoencoder.learningcurve`	fileDestination	no	—	Learning curve
`algorithm.pca.dim`	number	no	10.0	Dimension of the output of the pca transformation

Outputs: `io.out` (outputFile), `algorithm.autoencoder.learningcurve` (outputFile)

Example usage

```
run otb:TrainDimensionalityReduction io.vd=input.gpkg io.stats=input.dat feat=field1
 ↪  algorithm=0 algorithm.som.s=value algorithm.som.n=value algorithm.som.ni=5
 ↪  io.out=io.out.txt
```

This calls **TrainDimensionalityReduction**: `io.vd=input.gpkg` — Input Vector Data; `io.stats=input.dat` — Input XML image statistics file; `feat=field1` — Field names to be used for

training; `algorithm=0` — algorithm to use for the training; `algorithm.som.s=value` — Map size; `algorithm.som.n=value` — Neighborhood sizes; `algorithm.som.ni=5` — NumberIteration; `io.out=io.out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:TrainImagesClassifier` — TrainImagesClassifier

group: Learning

Parameter	Type	Required	Default	Description
io.il	multilayer	yes	—	Input Image List
io.vd	multilayer	yes	—	Input Vector Data List
io.valid	multilayer	no	—	Validation Vector Data List
io.imstat	file	no	—	Input XML image statistics file
io.out	fileDestination	yes	—	Output model
io.confmatout	fileDestination	no	—	Output confusion matrix or contingency table
cleanup	boolean	no	True	Temporary files cleaning
sample.mt	number	no	1000.0	Maximum training sample size per class
sample.mv	number	no	1000.0	Maximum validation sample size per class
sample.bm	number	no	1.0	Bound sample number by minimum
sample.vtr	number	no	0.5	Training and validation sample ratio
sample.vfn	field	no	—	Field containing the class integer label for supervision
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
classifier	OTBParameterChoice	no	0	Classifier to use for the training
classifier.libsvm.k	OTBParameterChoice	no	0	SVM Kernel Type
classifier.libsvm.m	OTBParameterChoice	no	0	SVM Model Type
classifier.libsvm.c	number	no	1.0	Cost parameter C
classifier.libsvm.gamma	number	no	1.0	Gamma parameter
classifier.libsvm.coef0	number	no	0.0	Coefficient parameter
classifier.libsvm.degree	number	no	3.0	Degree parameter
classifier.libsvm.nu	number	no	0.5	Cost parameter Nu
classifier.libsvm.opt	boolean	no	False	Parameters optimization
classifier.libsvm.prob	boolean	no	False	Probability estimation
classifier.boost.t	OTBParameterChoice	no	1	Boost Type
classifier.boost.w	number	no	100.0	Weak count
classifier.boost.r	number	no	0.95	Weight Trim Rate
classifier.boost.m	number	no	1.0	Maximum depth of the tree
classifier.dt.max	number	no	10.0	Maximum depth of the tree
classifier.dt.min	number	no	10.0	Minimum number of samples in each node
classifier.dt.ra	number	no	0.01	Termination criteria for regression tree
classifier.dt.cat	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split
classifier.dt.r	boolean	no	False	Set Use1seRule flag to false
classifier.dt.t	boolean	no	False	Set TruncatePrunedTree flag to false
classifier.ann.t	OTBParameterChoice	no	1	Train Method Type
classifier.ann.sizes	string	no	—	Number of neurons in each intermediate layer
classifier.ann.f	OTBParameterChoice	no	1	Neuron activation function type
classifier.ann.a	number	no	1.0	Alpha parameter of the activation function
classifier.ann.b	number	no	1.0	Beta parameter of the activation function
classifier.ann.bpdw	number	no	0.1	Strength of the weight gradient term in the BACKPROP method
classifier.ann.bpms	number	no	0.1	Strength of the momentum term (the difference between weights on the 2 previous iterations)
classifier.ann.rdw	number	no	0.1	Initial value Delta_0 of update-values Delta_{ij} in RPROP method

Parameter	Type	Required	Default	Description
classifier.ann.rdwm	number	no	1e-07	Update-values lower limit Delta_{min} in RPROP method
classifier.ann.term	OTBParameterChoice	no	2	Termination criteria
classifier.ann.eps	number	no	0.01	Epsilon value used in the Termination criteria
classifier.ann.iter	number	no	1000.0	Maximum number of iterations used in the Termination criteria
classifier.rf.max	number	no	5.0	Maximum depth of the tree
classifier.rf.min	number	no	10.0	Minimum number of samples in each node
classifier.rf.ra	number	no	0.0	Termination Criteria for regression tree
classifier.rf.cat	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split
classifier.rf.var	number	no	0.0	Size of the randomly selected subset of features at each tree node
classifier.rf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.rf.acc	number	no	0.01	Sufficient accuracy (OOB error)
classifier.knn.k	number	no	32.0	Number of Neighbors
classifier.sharkrf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.sharkrf.nodesize	number	no	25.0	Min size of the node for a split
classifier.sharkrf.mtry	number	no	0.0	Number of features tested at each node
classifier.sharkrf.oobr	number	no	0.66	Out of bound ratio
classifier.sharkkm.maxiter	number	no	10.0	Maximum number of iterations for the kmeans algorithm
classifier.sharkkm.k	number	no	2.0	Number of classes for the kmeans algorithm
classifier.sharkkm.incentroids	file	no	—	User defined input centroids
classifier.sharkkm.cstats	file	no	—	Statistics file
classifier.sharkkm.outcentroids	fileDestination	no	—	Output centroids text file
rand	number	no	0.0	Random seed

Outputs: io.out (outputFile), io.confmatout (outputFile), classifier.sharkkm.outcentroids (outputFile)

Example usage

```
run otb:TrainImagesClassifier io.il="a.gpkg;b.gpkg" io.vd="a.gpkg;b.gpkg"
 ↪  io.valid="a.gpkg;b.gpkg" io.imstat=input.dat cleanup=true sample.mt=1000
 ↪  sample.mv=1000 io.out=io.out.txt
```

This calls **TrainImagesClassifier**: io.il="a.gpkg;b.gpkg" — Input Image List; io.vd="a.gpkg;b.gpkg" — Input Vector Data List; io.valid="a.gpkg;b.gpkg" — Validation Vector Data List; io.imstat=input.dat — Input XML image statistics file; cleanup=true — Temporary files cleaning; sample.mt=1000 — Maximum training sample size per class; sample.mv=1000 — Maximum validation sample size per class; io.out=io.out.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:TrainImagesRegression — TrainImagesRegression

group: Learning

Parameter	Type	Required	Default	Description
io.il	multilayer	yes	—	Input predictor Image List
io.ip	multilayer	yes	—	Input label Image List
io.vd	multilayer	no	—	Input Vector Data List
io.valid	multilayer	no	—	Validation Vector Data List
io.imstat	file	no	—	Input XML image statistics file
io.out	fileDestination	yes	—	Output model
sample.nt	number	no	0.0	Number of training samples
sample.nv	number	no	0.0	Number of validation samples
sample.ratio	number	no	0.5	Training and validation sample ratio

Parameter	Type	Required	Default	Description
sample.type	OTBParameterChoice	no	0	Sampler type
sample.type.periodic.jitter	number	no	0.0	Jitter amplitude
rand	number	no	0.0	Random seed
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation
classifier	OTBParameterChoice	no	0	Classifier to use for the training
classifier.libsvm.k	OTBParameterChoice	no	0	SVM Kernel Type
classifier.libsvm.m	OTBParameterChoice	no	0	SVM Model Type
classifier.libsvm.c	number	no	1.0	Cost parameter C
classifier.libsvm.gamma	number	no	1.0	Gamma parameter
classifier.libsvm.coef0	number	no	0.0	Coefficient parameter
classifier.libsvm.degree	number	no	3.0	Degree parameter
classifier.libsvm.nu	number	no	0.5	Cost parameter Nu
classifier.libsvm.opt	boolean	no	False	Parameters optimization
classifier.libsvm.prob	boolean	no	False	Probability estimation
classifier.libsvm.eps	number	no	0.001	Epsilon
classifier.dt.max	number	no	10.0	Maximum depth of the tree
classifier.dt.min	number	no	10.0	Minimum number of samples in each node
classifier.dt.ra	number	no	0.01	Termination criteria for regression tree
classifier.dt.cat	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split
classifier.dt.r	boolean	no	False	Set Use1seRule flag to false
classifier.dt.t	boolean	no	False	Set TruncatePrunedTree flag to false
classifier.ann.t	OTBParameterChoice	no	1	Train Method Type
classifier.ann.sizes	string	no	—	Number of neurons in each intermediate layer
classifier.ann.f	OTBParameterChoice	no	1	Neuron activation function type
classifier.ann.a	number	no	1.0	Alpha parameter of the activation function
classifier.ann.b	number	no	1.0	Beta parameter of the activation function
classifier.ann.bpdw	number	no	0.1	Strength of the weight gradient term in the BACK-PROP method
classifier.ann.bpms	number	no	0.1	Strength of the momentum term (the difference between weights on the 2 previous iterations)
classifier.ann.rdw	number	no	0.1	Initial value Delta_0 of update-values Delta_{ij} in RPROP method
classifier.ann.rdwm	number	no	1e-07	Update-values lower limit Delta_{min} in RPROP method
classifier.ann.term	OTBParameterChoice	no	2	Termination criteria
classifier.ann.eps	number	no	0.01	Epsilon value used in the Termination criteria
classifier.ann.iter	number	no	1000.0	Maximum number of iterations used in the Termination criteria
classifier.rf.max	number	no	5.0	Maximum depth of the tree
classifier.rf.min	number	no	10.0	Minimum number of samples in each node
classifier.rf.ra	number	no	0.0	Termination Criteria for regression tree
classifier.rf.cat	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split
classifier.rf.var	number	no	0.0	Size of the randomly selected subset of features at each tree node
classifier.rf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.rf.acc	number	no	0.01	Sufficient accuracy (OOB error)
classifier.knn.k	number	no	32.0	Number of Neighbors
classifier.knn.rule	OTBParameterChoice	no	0	Decision rule
classifier.sharkrf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.sharkrf.nodesize	number	no	25.0	Min size of the node for a split
classifier.sharkrf.mtry	number	no	0.0	Number of features tested at each node
classifier.sharkrf.oobr	number	no	0.66	Out of bound ratio
cleanup	boolean	no	True	Temporary files cleaning

Outputs: io.out (outputFile)

Example usage

```
run otb:TrainImagesRegression io.il="a.gpkg;b.gpkg" io.ip="a.gpkg;b.gpkg"
 ↪  io.vd="a.gpkg;b.gpkg" io.valid="a.gpkg;b.gpkg" io.imstat=input.dat sample.nt=0
 ↪  sample.nv=0 io.out=io.out.txt
```

This calls **TrainImagesRegression**: `io.il="a.gpkg;b.gpkg"` — Input predictor Image List; `io.ip="a.gpkg;b.gpkg"` — Input label Image List; `io.vd="a.gpkg;b.gpkg"` — Input Vector Data List; `io.valid="a.gpkg;b.gpkg"` — Validation Vector Data List; `io.imstat=input.dat` — Input XML image statistics file; `sample.nt=0` — Number of training samples; `sample.nv=0` — Number of validation samples; `io.out=io.out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:TrainRegression` — TrainRegression

group: Deprecated

Parameter	Type	Required	Default	Description
`io.il`	multilayer	yes	—	Input Image List
`io.csv`	file	no	—	Input CSV file
`io.imstat`	file	no	—	Input XML image statistics file
`io.out`	fileDestination	yes	—	Output regression model
`sample.mt`	number	no	1000.0	Maximum training predictors
`sample.mv`	number	no	1000.0	Maximum validation predictors
`sample.vtr`	number	no	0.5	Training and validation sample ratio
`classifier`	OTBParameterChoice	no	0	Classifier to use for the training
`classifier.libsvm.k`	OTBParameterChoice	no	0	SVM Kernel Type
`classifier.libsvm.m`	OTBParameterChoice	no	0	SVM Model Type
`classifier.libsvm.c`	number	no	1.0	Cost parameter C
`classifier.libsvm.gamma`	number	no	1.0	Gamma parameter
`classifier.libsvm.coef0`	number	no	0.0	Coefficient parameter
`classifier.libsvm.degree`	number	no	3.0	Degree parameter
`classifier.libsvm.nu`	number	no	0.5	Cost parameter Nu
`classifier.libsvm.opt`	boolean	no	False	Parameters optimization
`classifier.libsvm.prob`	boolean	no	False	Probability estimation
`classifier.libsvm.eps`	number	no	0.001	Epsilon
`classifier.dt.max`	number	no	10.0	Maximum depth of the tree
`classifier.dt.min`	number	no	10.0	Minimum number of samples in each node
`classifier.dt.ra`	number	no	0.01	Termination criteria for regression tree
`classifier.dt.cat`	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split
`classifier.dt.r`	boolean	no	False	Set Use1seRule flag to false
`classifier.dt.t`	boolean	no	False	Set TruncatePrunedTree flag to false
`classifier.ann.t`	OTBParameterChoice	no	1	Train Method Type
`classifier.ann.sizes`	string	no	—	Number of neurons in each intermediate layer
`classifier.ann.f`	OTBParameterChoice	no	1	Neuron activation function type
`classifier.ann.a`	number	no	1.0	Alpha parameter of the activation function
`classifier.ann.b`	number	no	1.0	Beta parameter of the activation function
`classifier.ann.bpdw`	number	no	0.1	Strength of the weight gradient term in the BACK-PROP method
`classifier.ann.bpms`	number	no	0.1	Strength of the momentum term (the difference between weights on the 2 previous iterations)
`classifier.ann.rdw`	number	no	0.1	Initial value Delta_0 of update-values Delta_{ij} in RPROP method
`classifier.ann.rdwm`	number	no	1e-07	Update-values lower limit Delta_{min} in RPROP method
`classifier.ann.term`	OTBParameterChoice	no	2	Termination criteria
`classifier.ann.eps`	number	no	0.01	Epsilon value used in the Termination criteria
`classifier.ann.iter`	number	no	1000.0	Maximum number of iterations used in the Termination criteria

Parameter	Type	Required	Default	Description
classifier.rf.max	number	no	5.0	Maximum depth of the tree
classifier.rf.min	number	no	10.0	Minimum number of samples in each node
classifier.rf.ra	number	no	0.0	Termination Criteria for regression tree
classifier.rf.cat	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split
classifier.rf.var	number	no	0.0	Size of the randomly selected subset of features at each tree node
classifier.rf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.rf.acc	number	no	0.01	Sufficient accuracy (OOB error)
classifier.knn.k	number	no	32.0	Number of Neighbors
classifier.knn.rule	OTBParameterChoice	no	0	Decision rule
classifier.sharkrf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.sharkrf.nodesize	number	no	25.0	Min size of the node for a split
classifier.sharkrf.mtry	number	no	0.0	Number of features tested at each node
classifier.sharkrf.oobr	number	no	0.66	Out of bound ratio
rand	number	no	0.0	Random seed

Outputs: io.out (outputFile)

Example usage

```
run otb:TrainRegression io.il="a.gpkg;b.gpkg" io.csv=input.dat io.imstat=input.dat
↪  sample.mt=1000 sample.mv=1000 sample.vtr=0.5 classifier=0 io.out=io.out.txt
```

This calls **TrainRegression**: io.il="a.gpkg;b.gpkg" — Input Image List; io.csv=input.dat — Input CSV file; io.imstat=input.dat — Input XML image statistics file; sample.mt=1000 — Maximum training predictors; sample.mv=1000 — Maximum validation predictors; sample.vtr=0.5 — Training and validation sample ratio; classifier=0 — Classifier to use for the training; io.out=io.out.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:TrainVectorClassifier — TrainVectorClassifier

group: Learning

Parameter	Type	Required	Default	Description
io.vd	multilayer	yes	—	Input Vector Data
io.stats	file	no	—	Input XML image statistics file
io.out	fileDestination	yes	—	Output model
io.confmatout	fileDestination	no	—	Output confusion matrix or contingency table
layer	number	no	0.0	Layer Index
feat	field	no	—	Field names for training features
valid.vd	multilayer	no	—	Validation Vector Data
valid.layer	number	no	0.0	Layer Index
cfield	field	no	—	Field containing the class integer label for supervision
v	boolean	no	True	Verbose mode
classifier	OTBParameterChoice	no	0	Classifier to use for the training
classifier.libsvm.k	OTBParameterChoice	no	0	SVM Kernel Type
classifier.libsvm.m	OTBParameterChoice	no	0	SVM Model Type
classifier.libsvm.c	number	no	1.0	Cost parameter C
classifier.libsvm.gamma	number	no	1.0	Gamma parameter
classifier.libsvm.coef0	number	no	0.0	Coefficient parameter
classifier.libsvm.degree	number	no	3.0	Degree parameter
classifier.libsvm.nu	number	no	0.5	Cost parameter Nu
classifier.libsvm.opt	boolean	no	False	Parameters optimization
classifier.libsvm.prob	boolean	no	False	Probability estimation
classifier.boost.t	OTBParameterChoice	no	1	Boost Type

Parameter	Type	Required	Default	Description
classifier.boost.w	number	no	100.0	Weak count
classifier.boost.r	number	no	0.95	Weight Trim Rate
classifier.boost.m	number	no	1.0	Maximum depth of the tree
classifier.dt.max	number	no	10.0	Maximum depth of the tree
classifier.dt.min	number	no	10.0	Minimum number of samples in each node
classifier.dt.ra	number	no	0.01	Termination criteria for regression tree
classifier.dt.cat	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split
classifier.dt.r	boolean	no	False	Set Use1seRule flag to false
classifier.dt.t	boolean	no	False	Set TruncatePrunedTree flag to false
classifier.ann.t	OTBParameterChoice	no	1	Train Method Type
classifier.ann.sizes	string	no	—	Number of neurons in each intermediate layer
classifier.ann.f	OTBParameterChoice	no	1	Neuron activation function type
classifier.ann.a	number	no	1.0	Alpha parameter of the activation function
classifier.ann.b	number	no	1.0	Beta parameter of the activation function
classifier.ann.bpdw	number	no	0.1	Strength of the weight gradient term in the BACKPROP method
classifier.ann.bpms	number	no	0.1	Strength of the momentum term (the difference between weights on the 2 previous iterations)
classifier.ann.rdw	number	no	0.1	Initial value Delta_0 of update-values Delta_{ij} in RPROP method
classifier.ann.rdwm	number	no	1e-07	Update-values lower limit Delta_{min} in RPROP method
classifier.ann.term	OTBParameterChoice	no	2	Termination criteria
classifier.ann.eps	number	no	0.01	Epsilon value used in the Termination criteria
classifier.ann.iter	number	no	1000.0	Maximum number of iterations used in the Termination criteria
classifier.rf.max	number	no	5.0	Maximum depth of the tree
classifier.rf.min	number	no	10.0	Minimum number of samples in each node
classifier.rf.ra	number	no	0.0	Termination Criteria for regression tree
classifier.rf.cat	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split
classifier.rf.var	number	no	0.0	Size of the randomly selected subset of features at each tree node
classifier.rf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.rf.acc	number	no	0.01	Sufficient accuracy (OOB error)
classifier.knn.k	number	no	32.0	Number of Neighbors
classifier.sharkrf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.sharkrf.nodesize	number	no	25.0	Min size of the node for a split
classifier.sharkrf.mtry	number	no	0.0	Number of features tested at each node
classifier.sharkrf.oobr	number	no	0.66	Out of bound ratio
classifier.sharkkm.maxiter	number	no	10.0	Maximum number of iterations for the kmeans algorithm
classifier.sharkkm.k	number	no	2.0	Number of classes for the kmeans algorithm
classifier.sharkkm.incentroids	file	no	—	User defined input centroids
classifier.sharkkm.cstats	file	no	—	Statistics file
classifier.sharkkm.outcentroids	fileDestination	no	—	Output centroids text file
rand	number	no	0.0	Random seed

Outputs: io.out (outputFile), io.confmatout (outputFile), classifier.sharkkm.outcentroids (outputFile)

Example usage

```
run otb:TrainVectorClassifier io.vd="a.gpkg;b.gpkg" io.stats=input.dat layer=0 feat=field1
↪  valid.vd="a.gpkg;b.gpkg" valid.layer=0 cfield=field1 io.out=io.out.txt
```

This calls **TrainVectorClassifier**: io.vd="a.gpkg;b.gpkg" — Input Vector Data; io.stats=input.dat — Input XML image statistics file; layer=0 — Layer Index; feat=field1 — Field names for training features; valid.vd="a.gpkg;b.gpkg" — Validation Vector Data; valid.layer=0

— Layer Index; `cfield=field1` — Field containing the class integer label for supervision; `io.out=io.out.txt` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:TrainVectorRegression` — TrainVectorRegression

group: Learning

Parameter	Type	Required	Default	Description
`io.vd`	multilayer	yes	—	Input Vector Data
`io.stats`	file	no	—	Input XML image statistics file
`io.out`	fileDestination	yes	—	Output model
`layer`	number	no	0.0	Layer Index
`feat`	field	no	—	Field names for training features
`valid.vd`	multilayer	no	—	Validation Vector Data
`valid.layer`	number	no	0.0	Layer Index
`cfield`	field	no	—	Field containing the class integer label for supervision
`v`	boolean	no	True	Verbose mode
`classifier`	OTBParameterChoice	no	0	Classifier to use for the training
`classifier.libsvm.k`	OTBParameterChoice	no	0	SVM Kernel Type
`classifier.libsvm.m`	OTBParameterChoice	no	0	SVM Model Type
`classifier.libsvm.c`	number	no	1.0	Cost parameter C
`classifier.libsvm.gamma`	number	no	1.0	Gamma parameter
`classifier.libsvm.coef0`	number	no	0.0	Coefficient parameter
`classifier.libsvm.degree`	number	no	3.0	Degree parameter
`classifier.libsvm.nu`	number	no	0.5	Cost parameter Nu
`classifier.libsvm.opt`	boolean	no	False	Parameters optimization
`classifier.libsvm.prob`	boolean	no	False	Probability estimation
`classifier.libsvm.eps`	number	no	0.001	Epsilon
`classifier.dt.max`	number	no	10.0	Maximum depth of the tree
`classifier.dt.min`	number	no	10.0	Minimum number of samples in each node
`classifier.dt.ra`	number	no	0.01	Termination criteria for regression tree
`classifier.dt.cat`	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split
`classifier.dt.r`	boolean	no	False	Set Use1seRule flag to false
`classifier.dt.t`	boolean	no	False	Set TruncatePrunedTree flag to false
`classifier.ann.t`	OTBParameterChoice	no	1	Train Method Type
`classifier.ann.sizes`	string	no	—	Number of neurons in each intermediate layer
`classifier.ann.f`	OTBParameterChoice	no	1	Neuron activation function type
`classifier.ann.a`	number	no	1.0	Alpha parameter of the activation function
`classifier.ann.b`	number	no	1.0	Beta parameter of the activation function
`classifier.ann.bpdw`	number	no	0.1	Strength of the weight gradient term in the BACKPROP method
`classifier.ann.bpms`	number	no	0.1	Strength of the momentum term (the difference between weights on the 2 previous iterations)
`classifier.ann.rdw`	number	no	0.1	Initial value Delta_0 of update-values Delta_{ij} in RPROP method
`classifier.ann.rdwm`	number	no	1e-07	Update-values lower limit Delta_{min} in RPROP method
`classifier.ann.term`	OTBParameterChoice	no	2	Termination criteria
`classifier.ann.eps`	number	no	0.01	Epsilon value used in the Termination criteria
`classifier.ann.iter`	number	no	1000.0	Maximum number of iterations used in the Termination criteria
`classifier.rf.max`	number	no	5.0	Maximum depth of the tree
`classifier.rf.min`	number	no	10.0	Minimum number of samples in each node
`classifier.rf.ra`	number	no	0.0	Termination Criteria for regression tree
`classifier.rf.cat`	number	no	10.0	Cluster possible values of a categorical variable into K <= cat clusters to find a suboptimal split

Parameter	Type	Required	Default	Description
classifier.rf.var	number	no	0.0	Size of the randomly selected subset of features at each tree node
classifier.rf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.rf.acc	number	no	0.01	Sufficient accuracy (OOB error)
classifier.knn.k	number	no	32.0	Number of Neighbors
classifier.knn.rule	OTBParameterChoice	no	0	Decision rule
classifier.sharkrf.nbtrees	number	no	100.0	Maximum number of trees in the forest
classifier.sharkrf.nodesize	number	no	25.0	Min size of the node for a split
classifier.sharkrf.mtry	number	no	0.0	Number of features tested at each node
classifier.sharkrf.oobr	number	no	0.66	Out of bound ratio
rand	number	no	0.0	Random seed

Outputs: io.out (outputFile)

Example usage

```
run otb:TrainVectorRegression io.vd="a.gpkg;b.gpkg" io.stats=input.dat layer=0 feat=field1
 ↪  valid.vd="a.gpkg;b.gpkg" valid.layer=0 cfield=field1 io.out=io.out.txt
```

This calls **TrainVectorRegression**: io.vd="a.gpkg;b.gpkg" — Input Vector Data; io.stats=input.dat — Input XML image statistics file; layer=0 — Layer Index; feat=field1 — Field names for training features; valid.vd="a.gpkg;b.gpkg" — Validation Vector Data; valid.layer=0 — Layer Index; cfield=field1 — Field containing the class integer label for supervision; io.out=io.out.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:VectorClassifier — VectorClassifier

group: Learning

Parameter	Type	Required	Default	Description
in	vector	yes	—	Name of the input vector data
instat	file	no	—	Statistics file
model	file	yes	—	Model file
cfield	string	no	—	Output field
feat	field	no	—	Field names to be calculated
confmap	boolean	no	False	Confidence map
out	fileDestination	no	—	Output vector data file

Outputs: out (outputFile)

Example usage

```
run otb:VectorClassifier in=input.gpkg model=input.dat instat=input.dat cfield=value
 ↪  feat=field1 confmap=false out=out.txt
```

This calls **VectorClassifier**: in=input.gpkg — Name of the input vector data; model=input.dat — Model file; instat=input.dat — Statistics file; cfield=value — Output field; feat=field1 — Field names to be calculated; confmap=false — Confidence map; out=out.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:VectorDataExtractROI — VectorDataExtractROI

group: Vector Data Manipulation

Parameter	Type	Required	Default	Description
io.vd	vector	yes	—	Input Vector data
io.in	raster	yes	—	Support image
io.out	vectorDestination	yes	—	Output Vector data
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation

Outputs: io.out (outputVector)

Example usage

```
run otb:VectorDataExtractROI io.vd=input.gpkg io.in=dem.tif elev.dem=input.dat
 ↪  elev.geoid=input.dat elev.default=0 io.out=io.out.gpkg
```

This calls **VectorDataExtractROI**: io.vd=input.gpkg — Input Vector data; io.in=dem.tif — Support image; elev.dem=input.dat — DEM directory; elev.geoid=input.dat — Geoid File; elev.default=0 — Default elevation; io.out=io.out.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:VectorDataReprojection — VectorDataReprojection

group: Vector Data Manipulation

Parameter	Type	Required	Default	Description
in.vd	file	yes	—	Input vector data
in.kwl	raster	no	—	Use image keywords list
out.vd	fileDestination	yes	—	Output vector data
out.proj	OTBParameterChoice	no	0	Output Projection choice
out.proj.image.in	raster	yes	—	Image used to get projection map
out.proj.user.map	OTBParameterChoice	no	0	Map Projection
out.proj.user.map.utm.zone	number	no	31.0	Zone number
out.proj.user.map.utm.northhem	boolean	no	False	Northern Hemisphere
out.proj.user.map.epsg.code	crs	no	ProjectCrs	EPSG Code
elev.dem	file	no	—	DEM directory
elev.geoid	file	no	—	Geoid File
elev.default	number	no	0.0	Default elevation

Outputs: out.vd (outputFile)

Example usage

```
run otb:VectorDataReprojection in.vd=input.dat out.proj.image.in=dem.tif in.kwl=dem.tif
 ↪  out.proj=0 out.proj.user.map=0 out.proj.user.map.utm.zone=31
 ↪  out.proj.user.map.utm.northhem=false out.vd=out.vd.txt
```

This calls **VectorDataReprojection**: in.vd=input.dat — Input vector data; out.proj.image.in=dem.tif — Image used to get projection map; in.kwl=dem.tif — Use image keywords list; out.proj=0 — Output Projection choice; out.proj.user.map=0 — Map Projection; out.proj.user.map.utm.zone=31 — Zone number; out.proj.user.map.utm.northhem=false — Northern Hemisphere; out.vd=out.vd.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:VectorDataSetField — VectorDataSetField

group: Vector Data Manipulation

Parameter	Type	Required	Default	Description
in	vector	yes	—	Input
out	vectorDestination	yes	—	Output
fn	string	yes	—	Field
fv	string	yes	—	Value

Outputs: out (outputVector)

Example usage

```
run otb:VectorDataSetField in=input.gpkg fn=value fv=value out=out.gpkg
```

This calls **VectorDataSetField**: in=input.gpkg — Input; fn=value — Field; fv=value — Value; out=out.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:VectorDataTransform — VectorDataTransform

group: Vector Data Manipulation

Parameter	Type	Required	Default	Description
vd	vector	yes	—	Input Vector data
out	vectorDestination	yes	—	Output Vector data
in	raster	yes	—	Support image
transform.tx	number	no	0.0	X Translation
transform.ty	number	no	0.0	Y Translation
transform.ro	number	no	0.0	Rotation Angle
transform.centerx	number	no	0.0	Center X
transform.centery	number	no	0.0	Center Y
transform.scale	number	no	1.0	Scale

Outputs: out (outputVector)

Example usage

```
run otb:VectorDataTransform vd=input.gpkg in=dem.tif transform.tx=0 transform.ty=0
↪  transform.ro=0 transform.centerx=0 transform.centery=0 out=out.gpkg
```

This calls **VectorDataTransform**: vd=input.gpkg — Input Vector data; in=dem.tif — Support image; transform.tx=0 — X Translation; transform.ty=0 — Y Translation; transform.ro=0 — Rotation Angle; transform.centerx=0 — Center X; transform.centery=0 — Center Y; out=out.gpkg is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:VectorDimensionalityReduction — VectorDimensionalityReduction

group: Learning

Parameter	Type	Required	Default	Description
in	vector	yes	—	Name of the input vector data
instat	file	no	—	Statistics file
model	file	yes	—	Model file
out	fileDestination	no	—	Output vector data file containing the reduced vector
feat	field	no	—	Input features to use for reduction
featout	OTBParameterChoice	no	0	Output feature
featout.prefix.name	string	no	—	Feature name prefix
featout.list.names	string	no	—	Feature name list
pcadim	number	no	0.0	Principal component dimension

Parameter	Type	Required	Default	Description
mode	OTBParameterChoice	no	0	Writing mode

Outputs: out (outputFile)

Example usage

```
run otb:VectorDimensionalityReduction in=input.gpkg model=input.dat instat=input.dat
↪  feat=field1 featout=0 featout.prefix.name=value featout.list.names=value out=out.txt
```

This calls **VectorDimensionalityReduction**: in=input.gpkg — Name of the input vector data; model=input.dat — Model file; instat=input.dat — Statistics file; feat=field1 — Input features to use for reduction; featout=0 — Output feature; featout.prefix.name=value — Feature name prefix; featout.list.names=value — Feature name list; out=out.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:VectorRegression — VectorRegression

group: Learning

Parameter	Type	Required	Default	Description
in	vector	yes	—	Name of the input vector data
instat	file	no	—	Statistics file
model	file	yes	—	Model file
cfield	string	no	—	Output field
feat	field	no	—	Field names to be calculated
out	fileDestination	no	—	Output vector data file

Outputs: out (outputFile)

Example usage

```
run otb:VectorRegression in=input.gpkg model=input.dat instat=input.dat cfield=value
↪  feat=field1 out=out.txt
```

This calls **VectorRegression**: in=input.gpkg — Name of the input vector data; model=input.dat — Model file; instat=input.dat — Statistics file; cfield=value — Output field; feat=field1 — Field names to be calculated; out=out.txt is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

otb:VertexComponentAnalysis — VertexComponentAnalysis

group: Hyperspectral

Parameter	Type	Required	Default	Description
in	raster	yes	—	Input Image
outendm	rasterDestination	yes	—	Output Endmembers
ne	number	no	1.0	Number of endmembers
rand	number	no	0.0	Random seed
outputpixeltype *(adv)*	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: outendm (outputRaster)

Example usage

```
run otb:VertexComponentAnalysis in=dem.tif ne=1 rand=0 outendm=outendm.tif
```

This calls **VertexComponentAnalysis**: `in=dem.tif` — Input Image; `ne=1` — Number of endmembers; `rand=0` — Random seed; `outendm=outendm.tif` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.

`otb:ZonalStatistics` — ZonalStatistics

group: Image Manipulation

Parameter	Type	Required	Default	Description
`in`	raster	yes	—	Input Image
`inbv`	number	no	0.0	Background value to ignore in statistics computation
`inzone`	OTBParameterChoice	no	0	Type of input for the zone definitions
`inzone.vector.in`	vector	yes	—	Input vector data
`inzone.vector.reproject`	boolean	no	False	Reproject the input vector
`inzone.labelimage.in`	raster	yes	—	Input label image
`inzone.labelimage.nodata`	number	no	0.0	No-data value for the input label image
`out`	OTBParameterChoice	no	0	Format of the output stats
`out.vector.filename`	vectorDestination	yes	—	Filename for the output vector data
`out.xml.filename`	string	yes	—	Filename for the output xml file
`out.raster.filename`	rasterDestination	yes	—	File name for the raster image
`out.raster.bv`	number	no	0.0	Background value for the output raster
`outputpixeltype *(adv)*`	enum	no	5	Output pixel type — options: 0=uint8, 1=uint16, 2=int16, 3=uint32, 4=int32, 5=float, 6=double, 7=cint16, 8=cint32, 9=cfloat, 10=cdouble

Outputs: `out.vector.filename` (outputVector), `out.raster.filename` (outputRaster)

Example usage

```
run otb:ZonalStatistics in=dem.tif inzone.vector.in=input.gpkg
 ↪  inzone.labelimage.in=dem.tif out.xml.filename=value inbv=0 inzone=0
 ↪  inzone.vector.reproject=false out.vector.filename=out.vector.filename.gpkg
```

This calls **ZonalStatistics**: `in=dem.tif` — Input Image; `inzone.vector.in=input.gpkg` — Input vector data; `inzone.labelimage.in=dem.tif` — Input label image; `out.xml.filename=value` — Filename for the output xml file; `inbv=0` — Background value to ignore in statistics computation; `inzone=0` — Type of input for the zone definitions; `inzone.vector.reproject=false` — Reproject the input vector; `out.vector.filename=out.vector.filename.gpkg` is where the result is written. Enum values are integer indices; the paths and values here are illustrative — substitute your own.