

Updating the engine

ar021 · 11 July 2026

Updating demolab is a dependency bump, not a migration. The engine is the `demolab-cli` package: build code, Typst templates, runbooks, guides. Your lab is content. The two never mix, so pulling a new engine can't disturb anything that's yours.

The engine/content split

An update touches exactly one thing: the installed package. Everything the update leaves alone falls into two zones.

- **Yours, never touched:** `writings/`, `tools/`, `experiments/`, `artifacts/`, `demolab.yaml`, `HOUSESTYLE.local.md`, `pyproject.toml`, and the root stubs (`AGENTS.md`, `README.md`, `CI`). Branding (the wordmark and PDF titles) lives in `demolab.yaml`, outside the package, so an update can't touch your lab's identity.
- **Machine-managed:** the gitignored `.demolab/` staging directory at the lab root. The CLI refreshes it automatically; you never hand-edit it (see below).

Ask your agent

Say **UPDATE** (or “update demolab”) to your coding agent and it follows the `UPDATE` runbook: check the installed version against the latest release, show you the changelog entries in between, bump the dependency, rebuild, and commit the lockfile change. One hard gate: if the **major** version differs, the agent stops and asks before bumping, because a major release may need edits to your own content.

By hand

The whole update is one line, then a rebuild:

```
uv lock --upgrade-package demolab-cli && uv sync
demolab build && demolab test
```

`demolab version` prints the installed engine; the newest release is on PyPI.

What changed

The engine ships its own changelog. Read it before or after the bump:

```
demolab docs CHANGELOG --print
```

(Before updating, the same entries are behind the Changelog link on the PyPI page.) Each entry says what the release added or fixed, and a major entry says exactly which of your content it may require you to edit.

How versioning works

demolab uses SemVer: a **patch** is fixes, a **minor** is a new backward-compatible capability, and a **major** is a break that may need edits to your content (the tool ↔ experiment contract, the meta schema, or import paths). Two mechanisms keep the version honest:

- `uv.lock` pins the exact engine version, so a fresh clone of your lab rebuilds with the same engine until you bump it. Reproducibility is the lockfile's job.
- On the first `demolab build` after a version change, the CLI notices, refreshes the staged `.demolab/` directory automatically, and prints a one-line notice pointing at the changelog. Nothing for you to reconcile.

If a build breaks after an update

First read the changelog entry for the version you just pulled: if the release was a major, its entry lists the content edits it expects, and that is almost always the fix. Then say **DOCTOR** to your agent, which audits the repo against the conventions and points at anything out of contract. And because the update was one committed change to `pyproject.toml` and `uv.lock`, reverting that commit and running `uv sync` puts you back on the previous engine while you sort it out.