

# Codex remote development

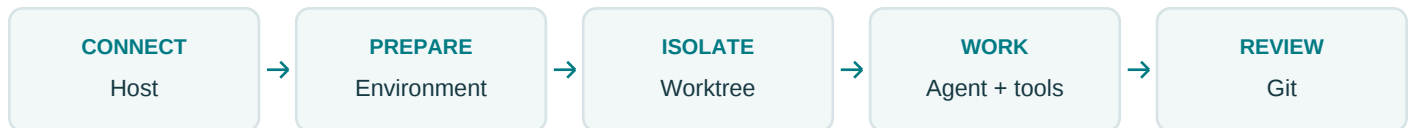
## AWS Helper field guide and cheat sheet

Use this guide to choose the right Codex feature, put information in the right file, and run a repeatable development workflow on your Mac or an SSH host.

**Prepared September 25, 2026.** AWS Helper 0.3.0; AWS integration preview. Documentation checked against current OpenAI sources; local CLI version 0.153.4. Settings labels follow your screenshot. Current OpenAI docs often call the desktop surface the ChatGPT desktop app.

## Five features, five jobs

| Feature      | The question it answers                          | Choice for this repository                                            |
|--------------|--------------------------------------------------|-----------------------------------------------------------------------|
| Connections  | Where do commands and file edits run?            | An existing SSH development host; your Mac remains the interface.     |
| Environments | How is this checkout prepared and operated?      | Shared setup plus Quick checks, Full checks, Build, and Demo actions. |
| Worktrees    | How do parallel tasks get separate source files? | A new worktree for each change; a branch when ready to publish.       |
| Git          | How do changes become reviewable and shareable?  | Review the diff, commit intended files, push a branch, open a PR.     |
| Hooks        | What should happen at a particular agent event?  | A short SessionStart hook with local branch and commit context.       |



## Keep these layers separate

**Instructions guide the agent.** `AGENTS.md` describes repository conventions. A skill describes a workflow to use when it is relevant.

**Configuration selects behavior.** `config.toml` supplies runtime settings; the environment file supplies setup commands and action buttons.

**Tools perform actions.** An MCP server exposes callable capabilities. A plugin packages capabilities for installation. An agent is the worker using those instructions and tools.

Start with the actual `codex-aws-helper` repository as your Codex project. The parent `codex-projects` folder is a different project scope.

## Reading map

Connections: page 2. Environments: page 3. Worktrees: page 4. Git: page 5. Hooks: page 6. Instruction hierarchy: page 7. Skills, plugins, and agents: page 8. First remote task: page 9. Troubleshooting and sources: page 10.

# Connections

## Choose the machine that owns the work.

A connection gives Codex access to a host's files and shell. For SSH projects, the desktop app starts a Codex app server on the remote host through SSH. Edits, builds, tests, and local MCP subprocesses run there.

| Execution choice | What it is good for                                                   | What you must prepare                                        |
|------------------|-----------------------------------------------------------------------|--------------------------------------------------------------|
| Your Mac         | Desktop behavior, local debugging, existing credentials.              | The local checkout and tools.                                |
| An SSH host      | Longer builds, remote dependencies, a persistent development machine. | SSH access, the repo, Codex, tools, and host authentication. |
| Codex cloud      | Isolated background code changes and pull requests.                   | GitHub access and a separate cloud environment.              |

## Connect an existing SSH host

1. Add a concrete alias to your Mac's `~/.ssh/config`. Replace the example values with your host, account, and existing key.

```
Host aws-helper-dev
  HostName devbox.example.com
  User developer
  IdentityFile ~/.ssh/id_ed25519
```

2. Confirm `ssh aws-helper-dev` works. Install and authenticate Codex on the host for your chosen provider. Its login shell must find `codex`.
3. Clone the private repository on that host and run the setup on page 3.
4. In **Settings > Connections**, add or enable the SSH host, then select the remote repository folder.
5. Save this same repository as a project on both hosts if you want Handoff. If the project is a subdirectory, select the same subdirectory on each.

## What stays with the host

The host supplies its filesystem, tool installations, Codex configuration, plugins, and credentials. An AWS Helper stdio server started there reads that host's selected AWS files. A working SSH connection does not establish AWS access or copy the Mac's authenticated sessions.

"Remote" can also mean controlling your desktop from a phone or another device. That pairs the devices; work still runs on the connected computer or its SSH environment. The connected desktop must remain available.

## Handoff

Handoff transfers a task and its Git state to a matching saved project on another connected host. It interrupts a running response before moving it. Handoff to Codex cloud is not supported. Use the cloud task's diff or PR to bring those changes back.

**For this setup:** host selection, SSH credentials, and live connectivity remain activation steps. No remote machine has been provisioned by the repo.

# Environments

## Turn a fresh checkout into a usable workspace.

Desktop local environments define a worktree setup script and named actions. Codex stores the shared file inside the project's `.codex` directory. Setup runs when the app creates a new worktree for a task and the environment is selected. Actions run in the integrated terminal.

## The shared environment

File: `.codex/environments/environment.toml`

```
version = 1
name = "AWS Helper development"

[setup]
script = "bash scripts/codex-dev.sh setup"

[[actions]]
name = "Quick checks"
icon = "run"
command = "bash scripts/codex-dev.sh quickcheck"
```

The checked-in file includes all five actions below. The shell wrapper is also usable directly over SSH and resolves its own checkout directory.

| Action / final command word | What it does                                                    | When to use it                                |
|-----------------------------|-----------------------------------------------------------------|-----------------------------------------------|
| Prepare workspace / setup   | Checks tools; fetches locked crates; compiles test binaries.    | A fresh worktree or changed dependencies.     |
| Quick checks / quickcheck   | Checks metadata, docs links, setup/hook tests, Rust formatting. | Docs and development-configuration edits.     |
| Full checks / check         | Quick checks, Clippy, and serial Rust tests.                    | Rust behavior changes and release validation. |
| Release build / build       | Builds the locked release executable.                           | Testing the native executable or packaging.   |
| Synthetic demo / demo       | Runs the interactive sample with Cargo offline.                 | Trying the experience without AWS access.     |

## Host prerequisites

Git, Bash, Make, a C compiler, Rust 1.88+ with `rustfmt` and `Clippy`, and Python 3.11+ for development checks. Python 3.9+ support refers to installation of the built package, not the development check script.

```
bash scripts/codex-dev.sh setup
bash scripts/codex-dev.sh quickcheck
# With dependencies already cached:
bash scripts/codex-dev.sh setup --offline
```

Setup can be repeated. It reuses Cargo caches and compiles with `--no-run`; successful setup is not a passing test suite. It does not install system packages, log in, or update personal AWS/Codex settings.

The desktop environment file does not create a Codex cloud environment. Cloud has separate setup/maintenance scripts and settings. Setup-shell exports do not automatically persist into its agent phase.

# Worktrees

## Give each change its own working directory.

A Git worktree is another checkout of the same repository. Each checkout has its own working files, while commits, branches, and Git objects are shared. Codex uses worktrees so parallel tasks can edit independently.

## The practical workflow

1. Start a task in the repository; choose the host, **Worktree**, starting branch, and **AWS Helper development** environment.
2. Let the setup script prepare the checkout. If a tool or manual Git command created the worktree without setup, run **Prepare workspace** yourself.
3. Make and verify one coherent change in that worktree.
4. Choose **Create branch here** before committing and pushing the result. A name such as `codex/config-preview-wording` identifies the work.
5. Review or hand off the task, then archive it when the work is preserved.

## Detached HEAD is normal

Codex-managed worktrees usually begin at a commit without a branch checked out. This is called detached HEAD. You can edit and test there; create a branch when you want a durable name for commits and a PR.

Git permits a branch to be checked out in only one worktree at a time. If Git says a branch is already used elsewhere, use **Handoff** or another branch. Handoff moves the task and code between Local and Worktree.

| Separate in a worktree            | Potentially shared on the host                         |
|-----------------------------------|--------------------------------------------------------|
| Tracked source files and edits.   | Git history, objects, and branch references.           |
| Default target/ build output.     | Cargo's downloaded dependency cache.                   |
| Test fixtures in temporary roots. | Personal config, AWS credentials, ports, and services. |

## Setup files and credentials

The repository supplies everything needed for its synthetic tests. This setup does not copy AWS credentials, `.env` files, or Codex history.

Codex supports `.worktreeinclude` for explicitly copying ignored files into **local managed** worktrees. That copying does not apply to SSH worktrees or plain `git worktree` commands. Tracked files are already in the checkout.

## Lifetime and cleanup

Current docs give a default retention limit of 15 managed worktrees. **Settings > Worktrees** controls the limit and worktree root. Pinned tasks, running tasks, and permanent worktrees are protected from automatic cleanup.

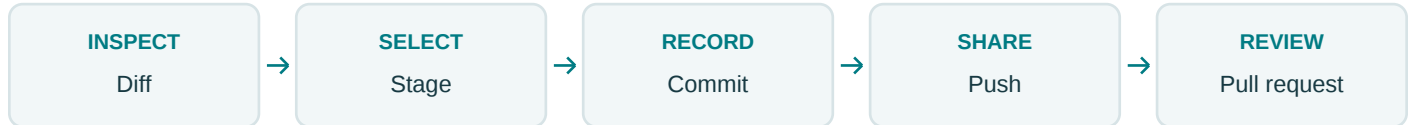
Archiving a task can delete its managed worktree. Codex saves a restorable snapshot before managed cleanup, but keep important work in commits and PRs. Use a permanent worktree when you want a long-lived development project.

Source isolation is useful; it is not a separate machine, credential boundary, or independent copy of every service your tests can reach.

# Git

## Make the change clear before sharing it.

Codex's Git controls operate on the current repository and worktree. The diff pane supports inline review comments, staging or reverting hunks and files, commits, pushes, and pull requests. Use the terminal for other Git operations.



## Suggested app preferences

Open **Settings > Git**. These are recommendations to apply in the app, not global settings already changed by the repository.

| Setting               | Suggested value or prompt                                                                                                           |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| Branch naming         | Prefix feature branches with <code>codex/</code> .                                                                                  |
| Force pushes          | Prefer ordinary pushes; resolve branch divergence deliberately.                                                                     |
| Review delivery       | Inline for quick iteration; detached when you want a separate review task.                                                          |
| Commit-message prompt | "Describe the concrete behavior change in a short imperative subject. Add a body only when context is needed."                      |
| PR-description prompt | "Explain the problem, resulting behavior, checks actually run, and material gaps. Keep credentials and personal configuration out." |

## Before you push

Review `git diff` and the staged diff. Stage the intended files so another task's changes do not accidentally enter your commit. Run checks appropriate to the change, then push its branch and open a PR.

The repository includes `.github/pull_request_template.md` with just the change and validation sections. A useful description explains what a user will experience and gives evidence a reviewer can assess.

```
git status --short
git diff
git diff --cached
```

These are inspection commands. Committing records local history; pushing uploads commits; a PR asks reviewers to consider a branch; merging updates the target branch. They are separate actions.

## Private repository and Code Defender

The repository stays private while testing continues. A Git remote such as `origin` is the synchronization destination; it is different from an SSH execution connection in Codex.

Git hooks execute during Git operations. Code Defender may run at this layer where installed. Codex lifecycle hooks run during agent events. Trusting the project `SessionStart` hook does not approve a repository for publication or replace the normal pre-push checks.

Push from the machine that owns the checkout, using that host's Git authentication and normal hooks. Keep publication separate from a successful build, a passing test, or a completed PR.

# Hooks

## Run a small, deterministic action at a named event.

Hooks are lifecycle callbacks. A command hook receives JSON on standard input and returns event-specific output. They are useful for brief session context or a focused validation. A hook runs because an event matched, not because the model remembered an instruction.

| Event        | Typical purpose                               | Important behavior                                       |
|--------------|-----------------------------------------------|----------------------------------------------------------|
| SessionStart | Add current checkout context.                 | Matches startup, resume, clear, or compact.              |
| PreToolUse   | Inspect a supported tool call before it runs. | Can block or rewrite supported calls.                    |
| PostToolUse  | Inspect a tool's result.                      | Cannot undo side effects that already occurred.          |
| Stop         | Check whether a turn needs more work.         | Can continue the turn; avoid endless continuation loops. |
| SessionEnd   | Brief final cleanup or bookkeeping.           | Advisory; does not keep the task running.                |

## What this repository defines

File: `.codex/hooks.json`

```
{
  "hooks": {
    "SessionStart": [{
      "matcher": "^(startup|resume|compact)$",
      "hooks": [{
        "type": "command",
        "command": "python3 \"$(git rev-parse --show-toplevel)/scripts/codex-session-start.py\"",
        "timeout": 5,
        "additionalContextLimit": 1000
      }]
    }]
  }
}
```

The script emits bounded branch/commit context and points to the development commands. It does not read transcripts, resolve AWS credentials, call AWS, or write configuration. It handles calls from repository subdirectories.

## Loading and trust

Hooks can live in user or project `hooks.json`, inline `[hooks]` tables in `config.toml`, or installed plugin bundles. Matching hooks from multiple sources all run; a project hook does not replace a user hook. Matching command hooks can run concurrently, so do not rely on their order.

Project hooks require a trusted project. Non-managed hooks also require review of the exact hook definition through **Settings > Hooks** or CLI `/hooks`. A changed definition needs review again. Managed hooks follow the organization's policy.

A trusted hook is executable code. Review the command and script together. Use the normal hook browser; installing a plugin or committing `hooks.json` does not by itself trust its hooks.

## Why keep this hook small?

Full tests belong in an action and CI. Running them after every tool call would repeat expensive work. The hook supplies timely context; `AGENTS.md` keeps durable conventions; the explicit check action supplies test evidence.

# Instructions and configuration

## Put guidance where its scope belongs.

An agent is a running AI worker. `AGENTS.md` is a guidance file the worker reads; it does not create an agent. The `.agents` name is a directory convention, not a general-purpose instructions file.

| File or location                                  | Primary audience                         | Put this here                                             |
|---------------------------------------------------|------------------------------------------|-----------------------------------------------------------|
| <code>README.md</code>                            | People trying the project.               | What it does, installation, examples, practical limits.   |
| <code>CONTRIBUTING.md</code> , <code>docs/</code> | Contributors and readers needing detail. | Development, architecture, troubleshooting, reference.    |
| <code>~/ .codex/AGENTS.md</code>                  | Codex across your projects.              | Personal coding preferences and general working guidance. |
| Repository <code>AGENTS.md</code>                 | Codex working on this repo.              | Conventions, boundaries, relevant checks and commands.    |
| Nested <code>AGENTS.md</code>                     | Codex in that subtree.                   | More specific local conventions.                          |
| <code>SKILL.md</code>                             | Codex using a particular workflow.       | When to use it, steps, references, and tool behavior.     |
| <code>.codex/agents/*.toml</code>                 | Codex spawning a named role.             | Role description, instructions, and allowed settings.     |
| <code>.codex/config.toml</code>                   | Codex's configuration loader.            | Shared settings for a trusted project.                    |
| <code>~/ .codex/config.toml</code>                | Codex's configuration loader.            | Personal runtime defaults, providers, and MCP setup.      |

## Guidance discovery

Codex reads personal guidance, then project guidance from the repository root down to the working directory. More specific project guidance appears later. At each scope, `AGENTS.override.md` takes priority over `AGENTS.md`; Codex uses one selected guidance file per directory.

This is a file-discovery order, not permission to override platform rules or managed policy. User task instructions take precedence over ordinary repository or skill guidance. Keep the guidance compact and relevant.

## Config precedence is a different chain

From highest to lowest: **CLI overrides; trusted project config; selected profile file; user config; cloud-managed defaults; system config; built-ins**. Inside the project chain, the closest directory wins for conflicting values. Untrusted project configuration is skipped.

Current named profiles can use `~/ .codex/<name>.config.toml`. Provider and authentication configuration belongs at user/machine scope; project layers cannot override protected provider keys. `requirements.toml` supplies enforced constraints, not just another set of defaults.

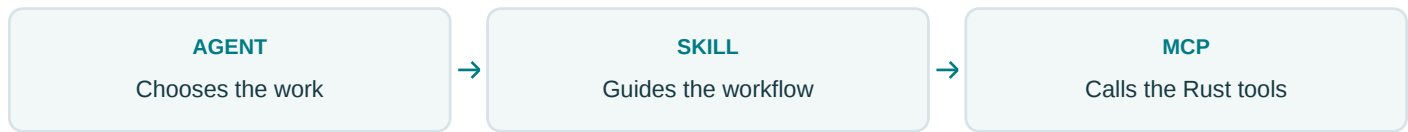
The environment's setup script prepares a checkout. `config.toml` configures Codex. Neither file is the place to paste AWS secret keys into this repo.

## Inspect instead of guessing

In the CLI, `/status` shows the active session and `/debug-config` shows configuration layers and managed requirements. `--strict-config` catches unknown keys. Saved files alone do not prove what an existing task is using.

# Skills, plugins, MCP, and agents

Package workflows and tools; give workers clear roles.



## Skill: a workflow on demand

A skill's name and description help Codex decide when it is relevant. Codex loads its `SKILL.md` body when selected, then reads supporting references or uses scripts as needed. You can request a skill explicitly or let its description match a task.

Standalone skills commonly live in `~/.agents/skills/<name>/SKILL.md` or a repository's `.agents/skills/`. Installed plugins carry their own skills. The AWS Helper plugin's `aws-setup` skill guides someone using the tool; the repository's `AGENTS.md` guides someone developing it.

## Plugin: an installable package

This repository keeps its existing, supported Codex compatibility layout:

|                                               |                      |
|-----------------------------------------------|----------------------|
| <code>.agents/plugins/marketplace.json</code> | discovery catalog    |
| <code>plugins/codex-aws-helper/</code>        |                      |
| <code>.codex-plugin/plugin.json</code>        | plugin manifest      |
| <code>.mcp.json</code>                        | studio server launch |
| <code>skills/aws-setup/SKILL.md</code>        | usage workflow       |

The marketplace points to the plugin directory. The manifest describes its capabilities. The MCP file starts `codex-aws-helper mcp`. Codex installs a copy in its plugin cache; edit the source and update the install, rather than editing that cache.

New portable plugins may use root `plugin.json` and `mcp.json` files. Their schema differs: do not migrate this bundle by merely renaming files.

## MCP: the executable capability

MCP is the protocol that exposes tools to Codex. This local server uses standard input/output and the same Rust configuration services as the CLI. It is not a second implementation or an extra background AWS agent.

For missing nonsecret settings, the server can request a native form. If the host lacks form support, the skill asks in the conversation. Sign-in stays in AWS CLI or the browser. The sequence is **choose values, preview the redacted change, apply the authorized plan, restore if needed**.

## Custom agent: a focused worker

`.codex/agents/aws_helper_reviewer.toml` defines an optional read-only reviewer for configuration loss, credential exposure, offline behavior, and CLI/MCP consistency. It is used when spawned; it does not run continuously.

Example: "Ask `aws_helper_reviewer` to review this diff and identify concrete regressions." A role inherits the session's permissions and may restrict them; naming a role does not grant new access.

# Your first remote task

Prepare, verify, review, then share.

## 1. Prepare the chosen host

Configure SSH and install the prerequisites on pages 2-3. Authenticate GitHub on the host with access to the private repository.

```
git clone https://github.com/mccartnick/codex-aws-helper.git
cd codex-aws-helper
bash scripts/codex-dev.sh setup
bash scripts/codex-dev.sh quickcheck
```

PR #1 is merged; start from main.

Add the host in Connections, select this repository's environment, and review the hook. The repository includes the setup; these host/app activation steps must still be completed for your chosen machine.

## 2. Start with a small development task

Choose **Worktree** and the environment. Try:

Improve one configuration-preview message. Follow AGENTS.md, use the shared planner, run the relevant checks, and show the diff.

Confirm the task is on the intended host and checkout. The trusted hook should supply branch/commit context. Use Quick checks for docs/setup work and Full checks for Rust changes. Ask the reviewer role for a second pass.

## 3. Install the plugin separately when you want to use it

Install the package and plugin on the host where Codex runs tools:

```
uv tool install codex-aws-helper
codex-aws-helper plugin install
```

The plugin records that host's installed executable path. Rerun `plugin install` after package upgrades. A permanent plugin should not depend on a binary inside a disposable worktree. Manual export remains available for custom bundles.

Start a new task and ask: "Show my configured AWS profile and region using offline information." For live discovery, specify the profile and region. For configuration, choose the exact model and review the proposed change.

## 4. Understand what each result proves

| Result                                 | Evidence it provides                                  |
|----------------------------------------|-------------------------------------------------------|
| Setup succeeded                        | Dependencies fetched and test binaries compiled.      |
| Synthetic suite passed                 | Tested code behavior with isolated fixtures.          |
| Model appeared in a catalog            | Listing visibility for that scope.                    |
| Explicit identity/model request passed | That particular AWS operation succeeded.              |
| A new Codex task worked                | Evidence for that task's tested client/configuration. |

Saved settings apply to new sessions. AWS profile metadata, a catalog row, or an STS result cannot establish the active task's effective credentials.

# Troubleshooting and references

Find the layer that owns the symptom.

| Symptom                          | First useful check                                                                                        |
|----------------------------------|-----------------------------------------------------------------------------------------------------------|
| SSH host is absent               | A concrete alias in <code>~/ .ssh/config</code> ; ordinary SSH works.                                     |
| Remote Codex cannot start        | codex is installed, authenticated, and on the remote login-shell PATH.                                    |
| No action buttons                | Open the actual repository project; select its environment. Use the script in the terminal as a fallback. |
| New worktree lacks tools         | Run Prepare workspace; confirm host prerequisites and Cargo network/cache access.                         |
| Hook does not run                | Project trust, hook enablement, and review of the current definition in <code>/hooks</code> .             |
| Branch is already checked out    | Use Handoff or a different branch.                                                                        |
| Plugin is missing or stale       | Check its enabled install, executable path, and host; start a new task.                                   |
| AWS profile differs from the Mac | Inspect the executing host's selected file roots and profile; do not assume credentials were copied.      |
| Full checks fail on Linux        | Inspect the failing fixture, prerequisites, and PTY access. Synthetic tests never need real credentials.  |

## Cloud setup, if you choose that route

The setup files are on the default branch, which cloud setup caches. Use Python 3.12, Rust 1.88+ with rustfmt/Clippy, and bash `scripts/codex-dev.sh` setup for setup and maintenance. The maintenance step refreshes dependencies after a cached container changes branches.

Keep AWS secrets unset for synthetic work. Cloud setup has network access; agent internet access is off by default. Run the checks with `CARGO_NET_OFFLINE=true` after setup. Environment variables persist through the task; cloud secrets are available to setup only. A Linux cloud container cannot validate the macOS desktop launcher.

## Keep this nearby

The companion `docs/codex-remote-setup.md` contains the full setup steps. The repository's `docs/validation.md` records product evidence. The PDF source is `docs/codex-cheatsheet.md`.

## Sources checked September 25, 2026

1. Remote connections - SSH, host resources, and Handoff.
2. Local environments - setup and actions.
3. Git worktrees - isolation, copying, cleanup.
4. Developer settings - Git and configuration inspection.
5. Hooks - events, trust, input/output, source merging.
6. `AGENTS.md` and `Config basics` - discovery and precedence.
7. Build skills, Subagents, and Package plugins.
8. Cloud environments - setup, cache, variables, secrets.
9. Codex environment example - verified TOML shape.

Feature availability and interface labels can vary by app version and rollout. This guide documents the prepared setup; it does not claim remote activation or live AWS validation.