Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Federal Chancellery FCh
**Digital Transformation and ICT Steering DTI**

20.10.2026

# Setting up the development environment

Version e48c4bf

# Table of Contents

1. Prerequisites 2
2. Installing Nix 2
3. Configuring Nix 2
4. Installing devenv 3
5. Installing direnv 3
6. Entering the environment 3
7. Verifying the setup 3
8. The git hooks 4
9. Rendering the documents 4
10. The editor 4
11. Matching what CI does 5
12. Local overrides 5
13. Troubleshooting 5

Everything needed to lint and render the documents in this repository is declared in `devenv.nix` and pinned by `devenv.lock`. Nothing is installed globally and no version is left to the machine: the toolchain a contributor gets is the one CI uses, which is why a clean `lint` locally means a green lint job.

This page covers getting into that environment. What to do once inside — how a five-language document is structured, and how to render one while translating it — is TRANSLATING.adoc.

## 1. Prerequisites

- **Linux on x86-64**, or Windows with WSL2. macOS is not supported: the renderer ships as a `linux-x64` binary and is run through a Nix FHS environment, which exists only on Linux, so the environment fails to evaluate elsewhere. See the comment at the top of `devenv.nix`.
- **git**, per the official instructions.
- **Disk and bandwidth.** The first shell downloads a few GB into the Nix store — most of it VS Code — and the first render pulls a further 112 MiB for the renderer.
- An internet connection, for both of the above.

## 2. Installing Nix

On NixOS there is nothing to do here. Everywhere else:

```
sh <(curl -L https://nixos.org/nix/install) --daemon
```

Then open a new shell, so that the profile script the installer added is sourced and `nix` is on the path.

> **NOTE** On Fedora and other SELinux distributions, the daemon may refuse to start. Open `/nix/var/nix/profiles/default/etc/profile.d/nix-daemon.sh` and comment out the first two lines.

## 3. Configuring Nix

Two settings, both in `/etc/nix/nix.conf` (on NixOS, in `nix.settings` in your system configuration instead):

```
experimental-features = nix-command flakes
extra-substituters = https://devenv.cachix.org
extra-trusted-public-keys = devenv.cachix.org-
1:w1cLUi8dv3hnoSPGAuibQv+f9TZLr6cv/Hm9XgU50cw=
```

The first is required: devenv is a flake, and the stock installer leaves both features off. The second points Nix at devenv's binary cache, so the closure is downloaded prebuilt rather than compiled locally — the difference between minutes and hours on the first shell. CI does the same thing with `cachix/cachix-action`.

Restart the daemon so the new settings are read:

```
sudo systemctl restart nix-daemon.service
```

## 4. Installing devenv

```
nix profile install --accept-flake-config nixpkgs#devenv
```

`--accept-flake-config` is what lets devenv's own substituters apply without a prompt. If you would rather not enable flakes at all, the channel equivalent works too:

```
nix-env --install --attr devenv -f
https://github.com/NixOS/nixpkgs/tarball/nixos-26.05
```

## 5. Installing direnv

direnv is what makes the environment load on `cd` instead of on demand. It is optional — `devenv shell` gives the same environment — but everything below assumes it.

1. Install it per the official instructions.
2. Hook it into your shell. This step is easy to skip and nothing works without it.

## 6. Entering the environment

```
git clone {url-repo}.git
cd opensource-guidelines
direnv allow
```

`direnv allow` is required once per clone, and again whenever `.envrc` changes: direnv refuses to run a file it has not been told to trust. The first load takes a while and says little; subsequent ones are instant.

Without direnv, the equivalent is `devenv shell` from the repository root, repeated in every terminal.

## 7. Verifying the setup

Entering the shell prints the helper scripts it provides. `devenv-help` prints the same list again at any time:

| Script | What it does |
|---|---|
| `devenv-help` | Print the list of scripts |
| `lint` | Run every git hook over every file |
| `render-docs` | Render every document in every language into `build/` |
| `open-govpress` | Render named documents — the renderer itself |

A working setup runs both of these clean:

```
lint                  # every hook, every file
render-docs en        # the English site into build/en/
```

`render-docs` finishes by naming the file to open — `build/index.html`, the language chooser, with `build/en/index.html` behind it. That is the same site CI publishes to GitHub Pages, built by the same scripts, which is the point of being able to build it locally at all.

## 8. The git hooks

Entering the shell installs the hooks; there is no separate install step and no `pre-commit install` to remember. `.pre-commit-config.yaml` at the root is generated from `devenv.nix` and gitignored — edit the `git-hooks.hooks` block in `devenv.nix`, never the generated file.

The hooks run against staged files at commit time. `lint` runs them against everything, which is what CI does, and takes arguments through to prek:

```
lint                                    # every hook, every file
lint check-asciidoc --all-files         # one hook, every file
```

Several hooks fix the file rather than only complaining — `end-of-file-fixer`, `dos2unix`, `trim-trailing-whitespace`, `nixfmt`. A commit they touch fails; stage the fix and commit again.

`git commit --no-verify` skips them. CI runs the same hooks on push, so this defers the failure rather than avoiding it.

## 9. Rendering the documents

`render-docs` builds the whole site; `open-govpress` renders named documents, which is the faster loop while editing one:

```
render-docs                                          # all five languages
render-docs de rm                                    # just those two
open-govpress render --lang de -o build/de/ docs/em002-5.adoc
```

The trailing slash on `-o build/de/` is not decoration: with a single input file, that path is treated as a directory only if it ends in a separator or already exists. Without it, the first render into a fresh clone writes a PDF into a file named `build/de`. TRANSLATING.adoc covers this alongside the rest of the per-document loop.

The first invocation of either downloads the renderer — a ~112 MiB release asset, checksum-verified — into `open-govpress/.cache/` and unpacks it into `open-govpress/.extracted/`. Both are gitignored and both are reused afterwards, so only the first render pays for this. It happens on first use rather than on shell entry so that entering the shell stays fast for anyone who is not rendering.

The renderer is Electron and has no headless mode, so it is always run under `xvfb-run`. This is handled by the `open-govpress` script and needs no display of your own.

## 10. The editor

Interactive shells include VS Code with the extensions that match the hooks — AsciiDoc, direnv, EditorConfig, Nix, ShellCheck, YAML and GitHub Actions — so that what the hooks reject is visible while typing:

```
code .
```

Two things the extensions cannot infer, both worth setting once: point `nix.formatterPath` at `nixfmt`, so the Nix extension formats the way the `nixfmt` hook expects; and note that the ShellCheck hook runs stricter than the extension does (`-x -o all`), so a clean editor does not guarantee a clean commit.

## 11. Matching what CI does

CI never installs the editor. Interactive shells get it, CI does not, and the switch is an environment flag:

```
devenv --option env.cicd:bool true shell -- prek run --all-files --show-diff-on-failure
```

That is the lint job from `.github/workflows/lint.yml`, verbatim. Reach for it when a hook passes locally and fails in CI, or to lint without pulling the VS Code closure at all.

## 12. Local overrides

devenv reads two files that let a machine differ from the repository: `devenv.local.nix` for Nix-level overrides, and `.env` for environment variables, which this environment has enabled.

```
# devenv.local.nix
{ ... }:
{
  env.SOME_TOKEN = "...";
}
```

> **WARNING** Neither file is in `.gitignore` here. Keep them out of commits by hand, or add them before you write one. See the devenv documentation for what else they can set.

Run `direnv reload` after changing either.

## 13. Troubleshooting

**`direnv: error .envrc is blocked`**
Expected on a fresh clone and after any change to `.envrc`. Run `direnv allow`.

**`devenv: command not found` right after installing it**
The profile script is sourced by login shells only. Open a new terminal.

**Nothing happens on `cd` into the repository**
direnv is installed but not hooked into the shell. See the hook instructions.

**The first shell is compiling rather than downloading**
The binary cache is not in effect. Re-check `extra-substituters` and `extra-trusted-public-keys` in `/etc/nix/nix.conf`, and that the daemon was restarted afterwards.

**`open-govpress` exits with code 5**
Electron could not find a display. Use the `open-govpress` script rather than the extracted binary — the script supplies `xvfb-run`.

**A checksum mismatch after bumping the renderer**
`govpressVersion`, `govpressUrl` and `govpressSha256` in `devenv.nix` and the three `GOVPRESS_` variables in `.github/workflows/render-docs.yml` are one version pinned in two

places, and have to move together.

**Renders still use the old version after a bump**

The extracted copy is keyed by version and should not go stale, but `rm -rf open-govpress/.extracted` forces a clean unpack.