Lix users

Various information for lix users beyond the main documentation, as well as instructions how to use Lix from the `main` branch. Make sure to also visit docs.lix.systems for the main Lix manual.

- Related projects
- Nix Resources
- Debugging a stuck Lix invocation
- Common issues
    - [archived] macOS Sequoia breaks existing Lix installations
- Debugging weird DNS resolutions inside sandbox

Related projects

This page lists community projects that are built with, optimized for, or make specific use of Lix features. While these projects are not officially affiliated with the Lix project, we maintain this curated list to highlight tools, systems, and initiatives that align well with Lix's design goals and ecosystem.

The projects featured here often explore advanced or experimental use cases, provide integrations, or offer enhancements that take advantage of Lix-specific capabilities. We aim to keep this page up to date as the ecosystem evolves.

If you maintain or know of a project that fits this description, feel free to suggest it for inclusion.

CI automation

- lix-gha-installer-action
- nix-build-action – defaults to the Lix GHA installer
- lix-quick-install-action – based on https://github.com/nixbuild/nix-quick-install-action

Lix sub-commands

This requires the `lix-custom-sub-commands` experimental feature as of May 2025.

- lix-diff similar to https://khumba.net/projects/nvd/

Misc

- Xil, an experimental alternate CLI for Nix implementations by Qyriad, which will potentially slowly merge with the Lix CLI.

Nix Resources

Lix doesn't exist in a vacuum—but instead builds atop the incredible work of dozens of NixOS community members. This page collects resources that have been recommended by members of the community.

We're also currently in the process of writing brand new documentation, which will be listed here once available.

A note of caution

Nix has a long history. Some things that seemed like good ideas once are no longer common practice. In particular:

- It's probably a good idea to ignore `nix-env` and mutable environments in favor of declarative configuration
- If you don't already use channels, it's probably a good idea to learn about flakes or some other pinning mechanism instead

If you're learning about those topics because you want to understand the ideas that are currently in favor in the historical contexts they are responses to, or because you think you see a benefit to them that others don't, that's totally fine; this warning is only here to help beginners have a fruitful experience.

Understanding Nix

- The official Nix ecosystem documentation, maintained by the **NixOS Foundation**.
- The NixOS and Flakes Book, by Ryan Yin.
- The Nix pills, a community classic with a guided tour of some of the Nix language basics.

Using Nix

- Customizing packages in Nix, by bobvanderlinden.
- Finding functions in Nixpkgs, by our own **jade**.
- Getting started with Home Manager for Nix, by Mattia Gheda.

Understanding Flakes

- Flakes aren't real and can't hurt you, by our own **jade**.

- Several of Xe Iaso's blog posts:
  - Nix Flakes: An Introduction
  - Nix Flakes: Packages and How To Use Them
  - Nix Flakes: Exposing and using NixOS Modules
  - Building Go Packages with Nix Flakes

Debugging a stuck Lix invocation

If you're experiencing an issue where a `lix` command appears to hang or make no visible progress, this page outlines multiple methods to investigate what’s going on.

Step 1: Increase Verbosity

Start by rerunning the Lix command with high verbosity to get a better sense of whether it's doing useful work.

```
nix build -vvv <your-command>
```

Check the output for any signs of activity such as:

- Downloads or substitutions
- Evaluations
- Store interactions

If logs show it's idle or stalled, move on to the next steps.

Step 2: Check CPU and memory usage

Examine whether the Lix process is still consuming CPU time or increasing its memory usage. This can be done using `top` (press `Shift+P` to sort by CPU time, `Shift+M` to sort by memory) or your favourite graphical system monitor.

If the client process has CPU usage close to 100% or its memory usage grows, the culprit is likely in the Nix code. Unfortunately no good debugging tools are available there, so one of the better ways is trying to narrow down the issue by trying to reproduce it with smaller packages or a system configuration with less services.

If the CPU usage is low and memory usage stays constant, proceed to the next step.

Step 3: `strace` the client

Attach `strace` to the stuck `lix` process to see where it is spending its time:

```
strace -yy -f nix build <your-command>
```

Watch for long periods spent on blocking syscalls like:

- `poll`

- `read`
- `recvmsg`
- `futex`

This means that it's stuck on communicating with the daemon, move on to the next steps.

Step 4: `strace` the subdaemon

Lix launches subdaemons to handle client requests. You can identify the correct subdaemon by inspecting processes:

```
ps ax | grep nix-daemon
```

Look for a line like:

```
356981 ?        Ssl    0:00 nix-daemon 67890
```

Here, `67890` is the PID of the client Lix process. The daemon serving it is `356981`.

Attach `strace` to the subdaemon:

```
strace -f -p 356981 -yy
```

This lets you inspect whether the daemon is doing work, waiting on a lock, or itself stalled.

Step 5: Investigate lock contention

If any of the traced processes are stuck waiting for file or directory locks (common with `.lock` files in the Nix store), you can identify who is holding the lock:

```
lsof <path to the lockfile>
```

Alternatively, you can ask all locks held by a process: `lsof -p <PID of the target process>`.

Another handy tool is `lslocks` for this.

Check whether the holder is another `nix-daemon` (or `nix-<something>`) process. If so, `strace` that one as well.

If that second process is:

- Doing useful work: this may be expected behavior.
- Also stuck on a lock or polling something indefinitely: you may be looking at a Lix bug.

Try to reduce the chain of waiting processes to the minimal reproducible set and report the issue with detailed diagnostics.

Other ideas

- Try `--pasta-path ""` to your invocation.

Common issues

A simple knowledge base of common issues and their resolutions. They may be migrated to other documentation as required.

[archived] macOS Sequoia breaks existing Lix installations

macOS 15 "Sequoia" changed the user ID ranges used by the system, such that some of the IDs for existing Nix build users got used by some new system features instead.

The symptom of this problem is that Nix builds will fail out with the following error, since the previous user ID used by `_nixbld1` got clobbered by Apple's updater:

```
error: the user '_nixbld1' in the group 'nixbld' does not exist
```

To fix this on your system, run the latest Lix installer (any version of the Lix installer ≥ `2.91.0-1`) with the `repair sequoia` option:

```
curl -sSf -L https://install.lix.systems/lix | sh -s -- repair sequoia
```

You may run this before or after updating to Sequoia (to run it *before* the upgrade, add `--move-existing-users` to the end of the command); it just migrates the user IDs used by Lix to a range recommended by Apple.

Relevant links:

- https://git.lix.systems/lix-project/lix-installer/issues/24
- https://github.com/NixOS/nix/pull/11075
- https://github.com/DeterminateSystems/nix-installer/issues/1115

Debugging weird DNS resolutions inside sandbox

Run this with pasta:

```
$ nix-build -E 'let pkgs = (import <nixpkgs> {}); in pkgs.runCommand "resolvconf" {
outputHashAlgo = "sha256"; outputHashMode = "flat"; outputHash = pkgs.lib.fakeHash; } "cat
/etc/resolv.conf"'
```

Report the result to a Lix developer.