# Contributing

This is the main information hub for drive-by contributors and regular developers alike. Everyone is encouraged to partake in documenting our project in here. Feeling insecure about an edit? Ask for affirmation in our Matrix channel :)

# Running Lix main

Thank you for choosing to help us dogfood in our daily development! If you run into any issues, you can reach out for quick support in `#running-main:lix.systems` on Matrix.

There is a lot of work-in-progress documentation and a lot of it is work in progress or awaiting move to the wiki. Our apologies for this state, let us know if there is something you need.

If you run into any friction, please let us know. We would like to hear all your complaints.

## Getting yourself set up with an account (if desired)

Sign in with GitHub on https://identity.lix.systems.

Note that your email will be visible on Gerrit if you use it, so change it on https://identity.lix.systems if necessary.

## A brief tour of the Lix systems

See Information Organisation for where information is.

The Lix sources are developed on Gerrit and released on a Forgejo repo.

Contributor documentation for the project is maintained on this wiki. FIXME(jade): a lot of it is awaiting migration onto the wiki from the private pad system, see tracking issue.

### On NixOS/nix-darwin

Use the overlay: https://git.lix.systems/lix-project/nixos-module

Please file bugs if this explodes the build of tooling you use, we can fix it in the overlay.

#### Flakes

Add Lix to your system configuration like so:

```
{
  inputs = {
    lix = {
      url = "https://git.lix.systems/lix-project/lix/archive/main.tar.gz";
      flake = false;
    };
```

```
    lix-module = {
      url = "https://git.lix.systems/lix-project/nixos-module/archive/main.tar.gz";
      inputs.nixpkgs.follows = "nixpkgs";
      inputs.lix.follows = "lix";
    };
  };

  outputs = {nixpkgs, lix-module, lix, ...}: {
    # or equivalent for darwin
    nixosConfigurations.your-box = nixpkgs.lib.nixosSystem {
      system = "x86_64-linux";
      modules = [
        ./machines/your-box
        lix-module.nixosModules.default
      ];
    };
  };
}
```

You can then update it with `nix flake update lix; nix flake update lix-module`.

## Not flakes

Also supported.

Add inputs for `git+https://git.lix.systems/lix-project/lix` and `git+https://git.lix.systems/lix-project/nixos-module` to your preferred pinning tool.

Use in a NixOS module: e.g. `imports = [ (import "${your-pinning-thingy.lix-nixos-module}/module.nix" { lix = your-pinning-thingy.lix; }) ];`

Niv

Add the sources for the module and Lix itself, using `ssh://` after registering your keys with git.lix.systems:

```
$ niv add git -n lix-nixos-module --repo 'https://git.lix.systems/lix-project/nixos-module'
$ niv add git -n lix-lix --repo 'https://git.lix.systems/lix-project/lix'
```

Then, import the Lix NixOS module:

```
  imports = [
    (import "${sources.lix-nixos-module}/module.nix"
```

```
      (let lix = sources.lix-lix.outPath;
      in {
        inherit lix;
        versionSuffix =
          "pre${builtins.substring 0 8 lix.lastModifiedDate}-${lix.shortRev}";
      }))
  ];
```

## On other Linux or on macOS

Currently we are still working on the installer (see tracking project). It is possible to convert an existing Nix install to Lix.

### flakey-profile

This is **experimental**. Some people have successfully used it on macOS. We have tested it on an Arch Linux system installed a long time ago with the shell-based installer, and it works fine. This method works by replacing your system profile with one that is built by simple Nix code with flakey-profile.

You can rollback if it blows up by `/nix/var/nix/profiles/default-{SECOND-HIGHEST-NUMBER}/bin/nix-env --rollback --profile /nix/var/nix/profiles/default`.

Clone `https://git.lix.systems/lix-project/nixos-module.git`, then, inside it, run `sudo nix run --extra-experimental-features 'nix-command flakes' .#system-profile.switch`.

Finally, run `sudo systemctl daemon-reload && sudo systemctl restart nix-daemon`, or, for macOS:

```
sudo launchctl stop system/org.nixos.nix-daemon
sudo launchctl enable system/org.nixos.nix-daemon
sudo launchctl kickstart -k system/org.nixos.nix-daemon
```

#### Restoring a broken install after a macOS update

After updating macOS, you may get error messages like these:

```
~/.nix-profile: no such file or directory
/nix/var/nix/profile/default: no such file or directory
error: cannot connect to socket at '/nix/var/nix/daemon-socket/socket': Connection refused
```

You can fix this by opening "Disk Utility" and manually mounting the `Nix` Volume again. Then, run these commands to re-install the lix daemon:

```
sudo launchctl load /nix/var/nix/profiles/default/Library/LaunchDaemons/org.nixos.nix-daemon.plist
```

```
sudo launchctl kickstart -k system/org.nixos.nix-daemon
```

### Manually, with `nix profile`

We::Qyriad have used these steps **on macOS** has it has **seemed** to work, but we would recommend flakey-profile over it.

```
$ sudo -H --preserve-env=SSH_AUTH_SOCK nix --experimental-features 'nix-command flakes' profile install --profile /nix/var/nix/profiles/default git+ssh://git@git.lix.systems/lix-project/lix --priority 3
```

- `--preserve-env=SSH_AUTH_SOCK` assumes that your SSH agent is important to access the Lix repo
- `--priority 3` makes it symlink Lix over your existing Nix install

If you then run `sudo nix --experimental-features 'nix-command flakes' profile list --profile /nix/var/nix/profiles/default`, you should get output similar to this:

```
Index:              0
Store paths:        /nix/store/8ma7xas2nb0i3lq8mm7fpgalv94s8pzh-nss-cacert-3.92

Index:              1
Store paths:        /nix/store/53r8ay20mygy2sifn7j2p8wjqlx2kxik-nix-2.19.2

Index:              2
Flake attribute:    packages.aarch64-darwin.default
Original flake URL: git+ssh://git@git.lix.systems/lix-project/lix
Locked flake URL:   git+ssh://git@git.lix.systems/lix-project/lix?ref=refs/heads/main&rev=98b497a1a43a4ff39263ed5259f12c5d00b4d8c0
Store paths:        /nix/store/8040hxr4rr8bpb5yp4b48709x3qs4bwb-nix-2.90.0
```

You may then use `sudo nix --experimental-features 'nix-command flakes' profile remove --profile /nix/var/nix/profiles/default 1` to remove your original installation of Nix. This is (probably) optional.

## Verification

You should now get something like the following:

```
~ » nix --version
nix (Nix) 2.90.0-lixpre20240324-f86b965
```

# Information organisation

Lix has a lot of information as a project, and we want to make it accessible in a way that it can be found later if necessary.

There are various tools for keeping information in the project, and they have different purposes

## Chat

Chat is good for:

- Information that will be meaningless in hours
- Ephemeral discussions, in general

The chat is expected to move way too fast to follow. As such:

- Don't write things in chat that you expect to be found later
- If discussions of design happen, write them down, at least by copy pasting into a pad and adding the pad to the index
- If tips and tricks are discussed, please write them down
- Please do reviews on Gerrit so they are archived, rather than in chat
- Write things down in the log if they are expected to be found

## Wiki (https://wiki.lix.systems)

The wiki is good for:

- Development process information, like you would find on https://rustc-dev-guide.rust-lang.org/ for the Rust compiler, for instance.
- Design documents

The wiki is not good for:

- User facing documentation
- Documentation that deserves to be reviewed
- Actively writing a document in real time with others

## Markdown files in the Lix repo

Markdown files in the Lix repo are good for:

- Maintaining things that are tied directly to the code
- Documentation that needs to be reviewed
- User facing documentation

Markdown files in the Lix repo are bad for:

- Quickly iterating on things
- Design documents

# Forgejo issues (https://git.lix.systems)

Our primary issue tracker is Forgejo issues.

We are currently attempting to use the Forgejo project boards feature to communicate what people are working on; it may be replaced with better Kanban software in the future. When making project boards on Forgejo, make them on the `lix-project` organisation unless they are strictly contained within one project.

The issue tracker is good for:

- Actionable work
- Bugs

The issue tracker is not good for:

- Dreams or otherwise not actionable information that is a long term goal
- Private information
- Information that needs to be found later, design documentation

## Where to put an issue

- lix-project/lix, if it is contained within Lix (but is not more appropriate to put in the installer e.g.)
  - If it is an upstream bug, tag its equivalent `lix-import` on https://git.lix.systems/nixos/nix, and get someone with the bot token to run the issue import script in `maintainers/issue_import.py`. (FIXME: someone ought to put that on a cron job)
  - Please never file issues on our Nix mirror.
- lix-project/installer, if it is the installer
- lix-project/web-services, if it is infrastructure related
- lix-project/meta, if it does not fit anywhere obvious and you just need it to put it on a board
- lix-project/nixos-module, if it is a packaging bug in that specifically

# Gerrit (https://gerrit.lix.systems)

Gerrit is good for:

- Reviewing code

- Maintaining a record of code reviews

Gerrit is not good for:

- Persisting information in a discoverable way to anyone in the future
- Documentation

## Pad (https://pad.lix.systems [private])

We anticipate that the pad service will be semi-permanently private by default, since it doesn't support ACLs.

The pad is good for:

- Sketching out drafts of documents that aren't ready yet
- Planning private things
- Generally getting people on the same page about things in active design, making what might be meeting notes, or similar.

The pad is not good for:

- Information that should be available to users (unless it is planned to move)
- Information that is not actively changing

It is very important that pads remain temporary in nature, and are quickly moved to their final destination, e.g. the Wiki.

N.B. For users who aren't in the Lix core team, the service returns 500 when you attempt to login. This is a known issue that can't be fixed.

# Intro to Gerrit

Everything about Gerrit: how to get started, advanced tips and tricks, and more!

# Getting Started with Gerrit

Thanks for showing interest in contributing to Lix! Gerrit can seem daunting at first, but it is our hope that you'll learn to navigate it and use it confidently after finishing this tutorial.

Perhaps the first question you have is: "Why Gerrit"? Well, glad you asked! In fact, we have an entire page describing that. But in short: it's just very nice for working with code. Instead of "PR from a branch" model that Github uses, Gerrit assigns a Change-Id to a commit, which is preserved even when the commit itself changes. This is perhaps familiar to Jujutsu VCS users.

This model allows us to largely ignore branches, which have long been known to introduce many operational complexities. Instead, we can focus on getting the work done, with our tooling supporting us in this goal!

## Setting Up

Here are a few things you will need. Let's go over them one by one.

1. Go to our Gerrit instance and sign in. It will prompt you to sign in with Github SSO. If you don't want to use your Github account, that's fine; just hit us up on Matrix and we'll create you an account!
2. Get your SSH key ready! If you don't have one, run `ssh-keygen -t ed25519` to generate the SSH keypair with ed25519 cypher. It is the most secure SSH cypher currently available, and quite ergonomic to use! Passphrase isn't necessary for Gerrit, but if you want to use it, we recommend to also set up ssh-agent.
3. Go to your user settings in Gerrit, scroll down to "SSH keys", and add your public SSH key (from `~/.ssh/id_ed25519.pub`).

## Configuring Git Repo

Now, we need to make some setup for the Git repo. If you haven't yet cloned Lix repo, use this fancy one-liner to do it (just remember to change `USERNAME` to your actual username!):

```
git clone "ssh://USERNAME@gerrit.lix.systems:2022/lix" && (cd "lix" && mkdir -p `git rev-parse --git-dir`/hooks/ && curl -Lo `git rev-parse --git-dir`/hooks/commit-msg https://gerrit.lix.systems/tools/hooks/commit-msg && chmod +x `git rev-parse --git-dir`/hooks/commit-msg)
```

If you want to get a deeper understanding, or you already cloned the repo from Forgejo - read on!

This is a little awkward, but yes - we store a Git repo in both Forgejo and Gerrit. This is a technical limitation of our infra. The "actual" repo lives **in Gerrit** - and it is automatically mirrored to Forgejo. This means that we want to push changes directly to Gerrit.

1. If you cloned the repository from Forgejo, rewrite the origin to point to Gerrit instead: `git remote set-url origin ssh://USERNAME@gerrit.lix.systems:2022/lix`
2. Gerrit wants a `Change-Id` footer in your commit message to work (and track changes to your commit). Adding it by hand is very inconvenient; thankfully, there's a Git hook that adds this footer if it doesn't already exist, even on amends of existing commits. It is added **automatically** if you are in `nix develop` shell. Otherwise, you can add it manually:

```
mkdir -p .git/hooks
curl -Lo .git/hooks/commit-msg https://gerrit.lix.systems/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg
```

3. Now you can just `git commit` your change. No need to create a separate branch - your commit is the unit of review here!
4. Gerrit also expects you to push in an unusual way, too: `git push origin HEAD:refs/for/main`. Just run this command to automate it: `git config remote.origin.push HEAD:refs/for/main`. Now you'll be able to `git push` like normal!
5. Now, you can just `git push`, and you'll see a link to your CL (Change List)! Now pat yourself on the back and wait for review :)

# What Next?

Now, you should be getting a review in a few days - especially if your CL is quite small. Generally, Lix team goes through the open CLs quite often, and helps the new ones get to completion. Sometimes, people might be busy or just miss your CL - so don't take it to heart, and ask in Matrix for a review!

Once you got a review, you can address some issues - and fix others. When fixing issues, don't create new commits - use `git commit -a` instead to amend the first commit that you pushed to Gerrit initially. Remember, each commit is a separate review piece - so unless you want to create a new CL, just amend the existing commit!

When the review process is done and everybody is satisfied, you'll get approvals from maintainers, and your change will become "Ready to submit". Now is your time to shine! Unlike Github, Gitlab or Forgejo, maintainers aren't the ones who can press the "Submit" button. You can get a last look at the change - and when you decide you're ready, push the "Submit" button, and your change will be part of the codebase. Congratulations!

Now, perhaps this experience was too easy, and you want to learn more Gerrit. Fair enough! Here are some pointers for you:

- We have a [different page] on Gerrit, that lists a bunch of cool tricks and features.

- You might want to check Github to Gerrit user guide: it largely goes over the same things as this documentation page, but it can give you more information about specific topics
- You might want to explore more of our Gerrit documentation!
- There's official user guide
- This user guide is part of official guide! There is also
- Github to Gerrit user guide, which this guide is largely based on.

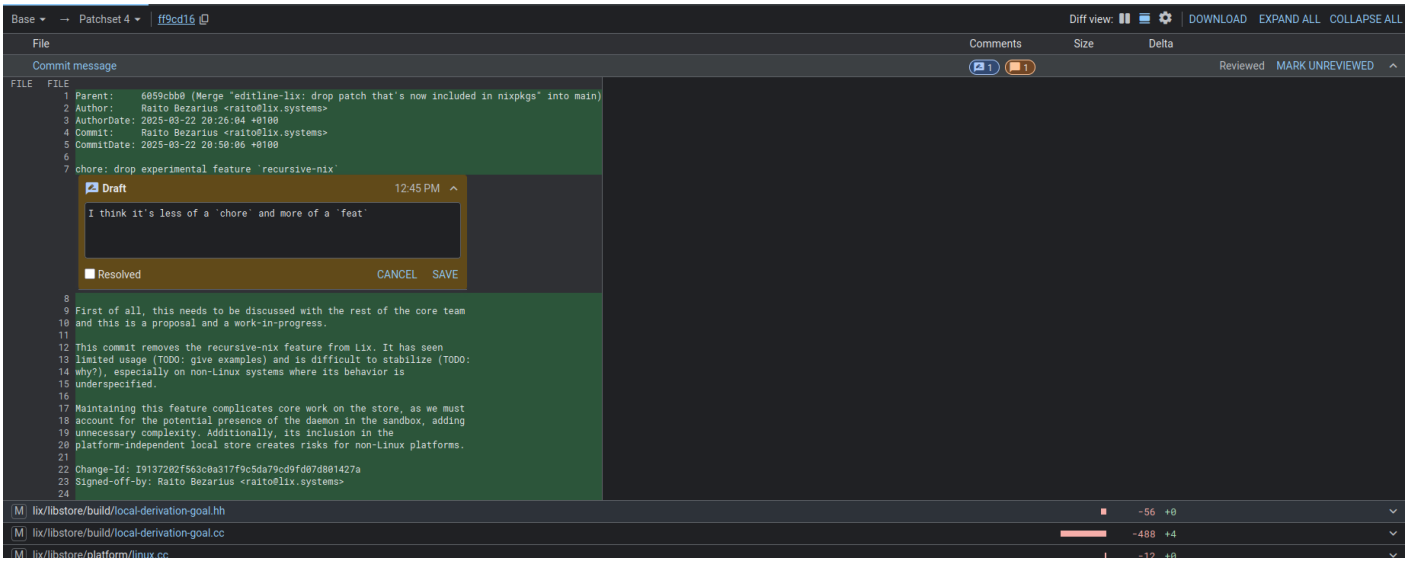
Good luck on your journey, and thanks for your contribution!

# Review Flow

The review flow is quite similar to how Github does it, but there are a few differences here too. The UI also hides a few pretty powerful features!

## Reviewer Flow

You can scroll down to the changes and start a review:

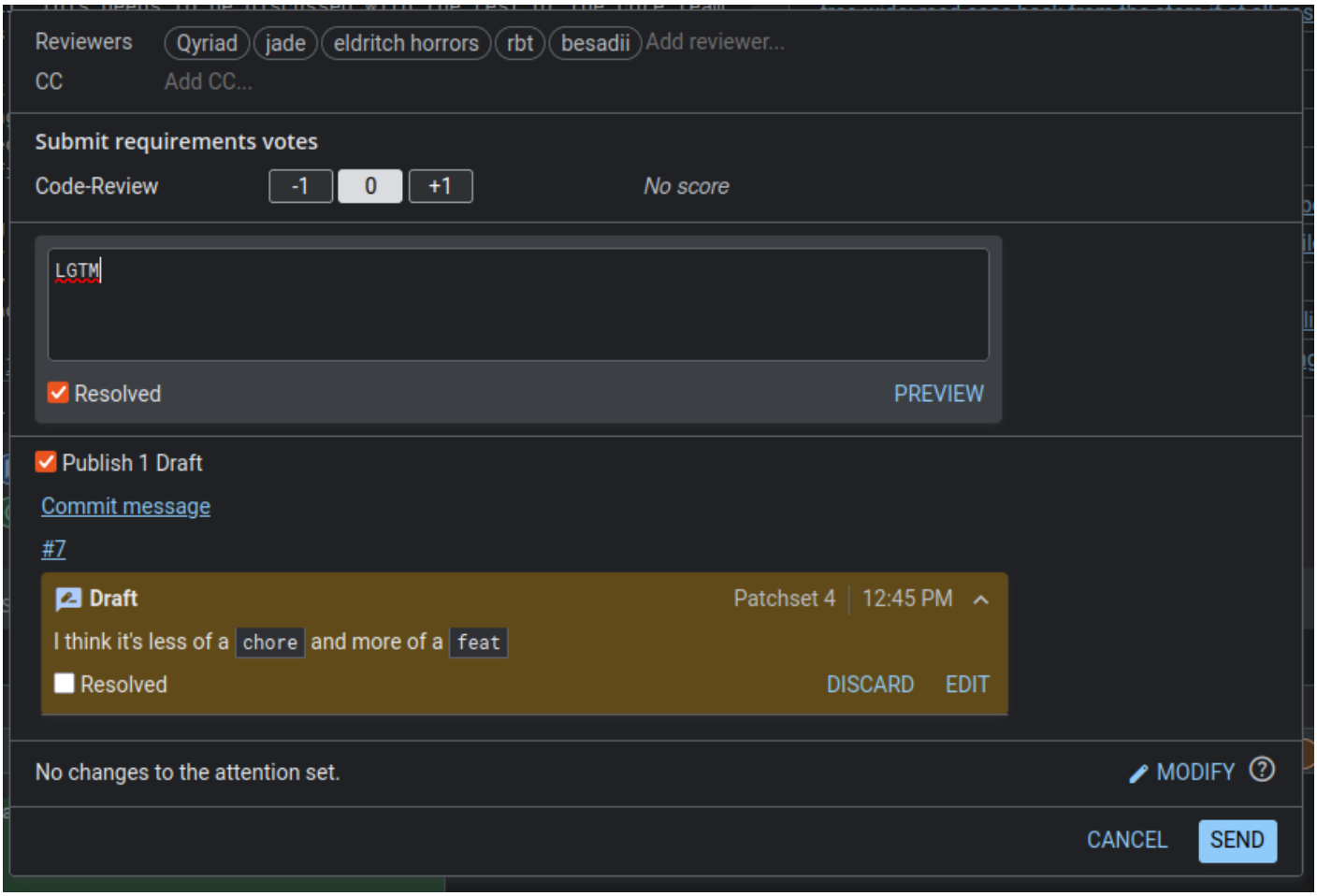



You can click on a line number to add a comment. Like on Github, the comments will be saved in a "draft" state, and will be submitted all at once when you finish the review.

When you expand the change to a file, the file will be marked as "reviewed" automatically - for your convenience. If you press "Expand All" button, you'll have to mark files as "reviewed" manually.

Finally, you can also change the "revisions" of comparison: you can choose to review the diff between Patchset [version] 3 and Patchset [version] 4! FIXME: this might be done automatically on re-review?

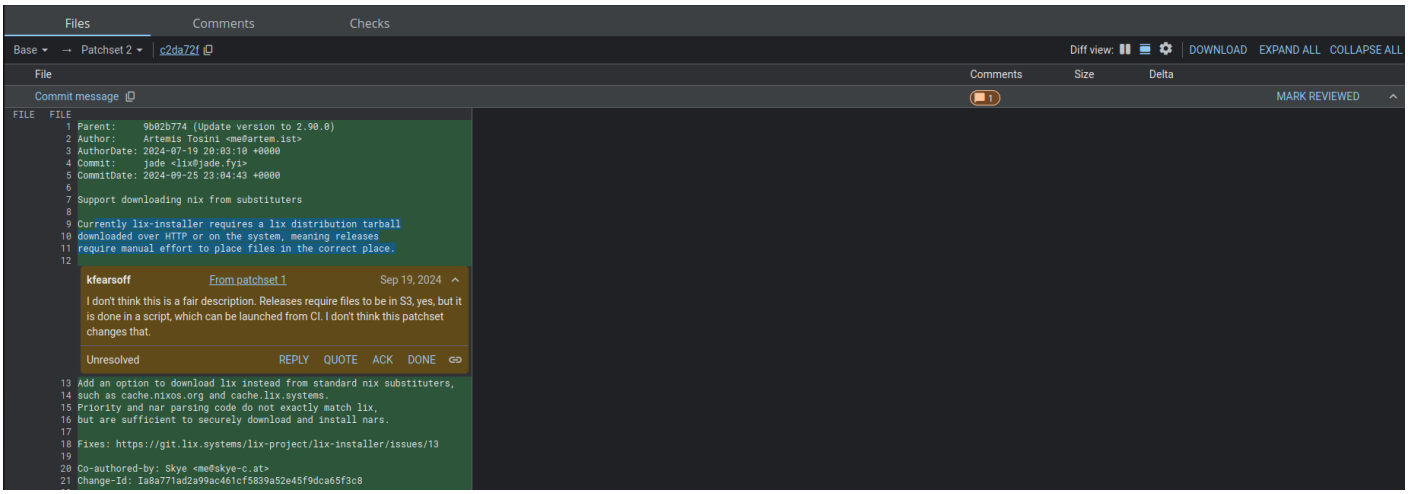
Finally, when the review is done, you scroll up and press the "Reply" button to submit a review, where you can add a final comment:




There, it's possible to mark some comments as resolved (maybe they're a non-important nitpick!), and also add a Code-Review vote. This is similar to Github's "approve/comment/request changes" on a PR, but a little more flexible. For a change to be ready for merge, it needs +2 on Code Review. Lix maintainers can +2 your change by themselves, so in normal circumstances, your change will require maintainer approval or 2 reviews from non-maintainers.

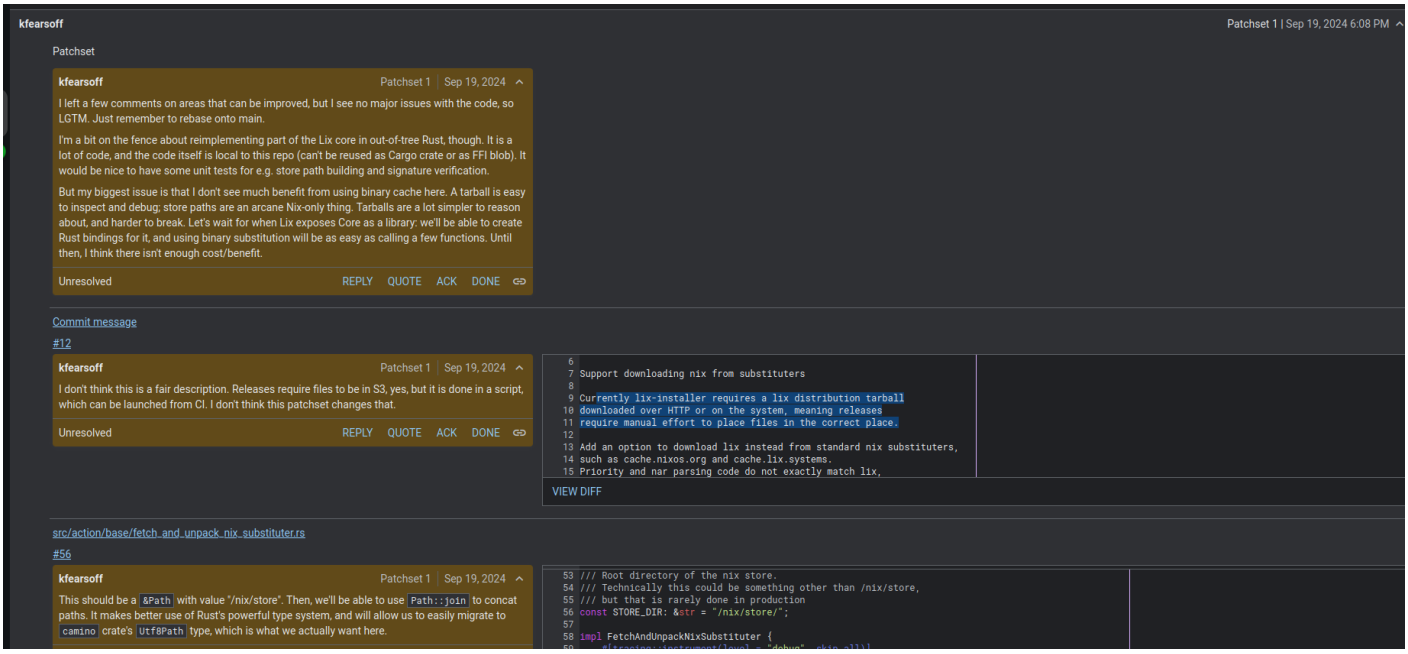
# Reviewee Flow

There are two ways to view the review. One is to look at it directly in code:



This will show you the whole comment thread.

Another way is to scroll down to the change log and expand a review:

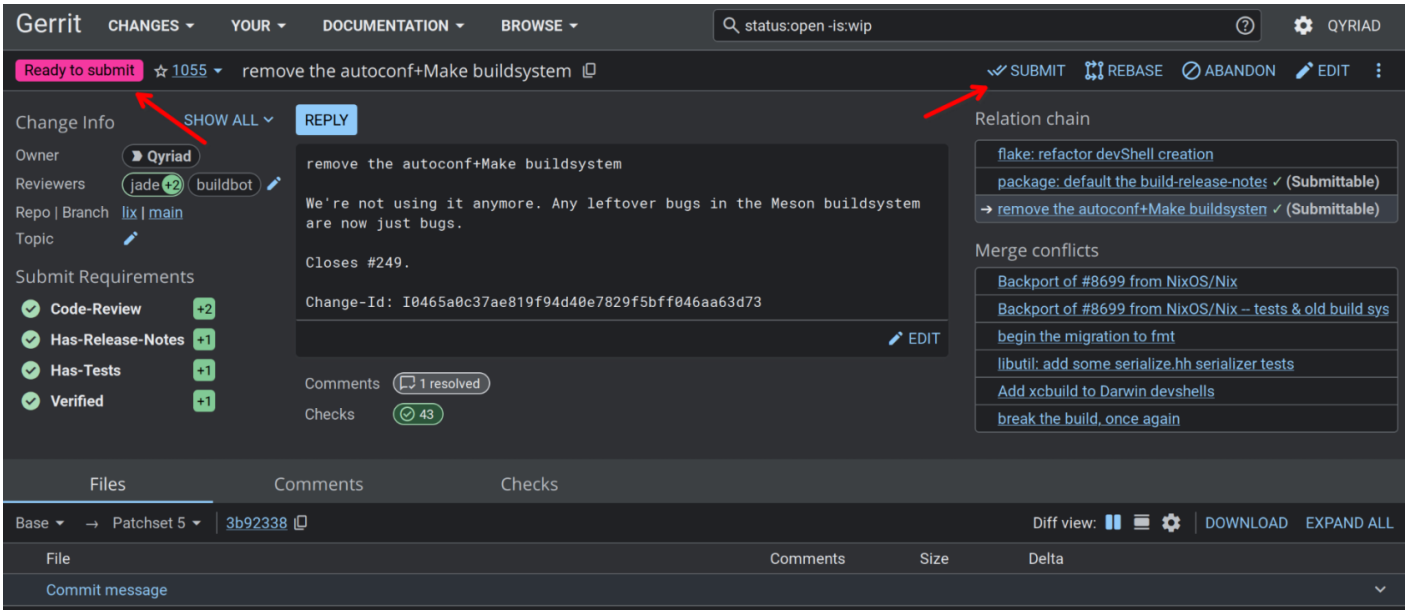



This will show you the entire review at once, with threads collapsed.

Choose whichever you like best!

# Merge Flow

Once approved, the change can be merged. Maintainers will sometimes do that, but we generally prefer to leave the decision to the person who submitted the change! Take a final look at your change, verify that everything looks good, and press the "Submit" button!



# Why Gerrit?

Gerrit produces better code:

- Gerrit enforces good commit messages. "PR message" and "commit message" are the same thing in Gerrit; there's no duplication, and information about a change can be seen in regular commit history.
- Gerrit enforces good commit hygiene, since adding another commit is really just splitting a commit with `git revise -c` or other tools; since there are no PR dependencies or branches to worry about, splitting commits is no longer a big ask.
- Relatedly, this directly makes reviews smaller since the overhead of doing another change is low.

Gerrit makes reviewers' lives easier and reduces review round trips:

- As a reviewer, you can look at what changed since you last reviewed, even in the presence of rebases, by looking at the patchset history of a CL. This avoids pointless rereview; you can actually diff versions of changes properly.
- The change author generally merges the change after approval, without them needing commit access. This means that they can do a final once-over of the change and make sure that they are ok with its state before merging it. This reduces miscommunication causing merging of unfinished code.
- As a reviewer, you can edit someone's change and/or commit message to fix a typo (in the web interface) and then stamp it, while giving them the final say on merging the edited change.
- You can give feedback like the following: "I would merge this as-is but you can consider this feedback if you would like" and then let the change author decide to merge it.
- Since the permission-requiring step in Gerrit is approving the change, not merging it, every change author can have final say in when the change gets merged.
- Review suggestions get applied as a batch without cluttering commit history in a confusing manner.
- You can download someone's change to look at it locally in one command that you can copy paste from the Gerrit interface (keyboard shortcut: `d`).

Gerrit makes your life easier as a contributor:

- Submitting a new change is just a matter of committing it and pushing it. You don't need to think about branches or the web interface or extra commands. Want to do more changes building on it? Just commit them and push them.
- Branches are not required and you can easily build off of other peoples' changes by fetching them and rebasing against them; change dependencies are simply commit parents. They can then be merged in whichever manner they will be merged.

- If you are doing a larger change, it is natural to merge it piece by piece, adding little improvements as you go, and putting the highest risk parts of it at the tip, making the obviously good parts of the change land and keeping your diffs and rebases against main smaller.
- Gerrit makes it clear which comments still need action in a clean way, compared to GitHub where resolved comments get regularly broken or disappear altogether.
- Gerrit guesses (with reasonable accuracy) who a change is blocked on and shows it on the dashboard with a little arrow next to their name, allowing you to see at a glance which changes are your responsibility at a given time.
- There is a rebase button that just works. Trivial non-conflicting rebases do not require a rereview.

That being said, there are some downsides:

- Gerrit is very mean to you if you don't have your commit history in a clean presentable state, which takes some getting used to and Git does not make editing history easy, so it does involve a little more fighting of Git. However, this also means that the reviews can be of cleaner and smaller pieces of code with fewer unrelated changes.
  - **jujutsu fixes this**: if you are finding an amend based workflow frustrating, we highly recommend using Jujutsu in place of Git. See Jujutsu usage with Gerrit for more details.
  - This makes pushing work in progress code with questionable commit history harder; see below for solutions to this.
- Gerrit requires a little bit of local setup in the form of adding your SSH key or setting up the HTTP password. It also requires a Git commit-msg hook, but nix develop automatically does that for you.

# Gerrit with Jujutsu

Jujutsu has a very similar model to Gerrit and natively supports sending Gerrit changes!

You'll need to configure a Gerrit remote. If it's not `origin`, you can instruct Jujutsu to use it by specifying it in `.jj/repo/config.toml`:

```
[gerrit]
default-remote = "gerrit"       # name of the Git remote to push to
default-remote-branch = "main"  # target branch in Gerrit
```

After that, `jj gerrit upload -r <revision>` will automatically add `Change-Id` footers and send your changes to Gerrit, printing a link to a resulting patchset to the terminal.

See Jujutsu docs for more information.

# Tips and Tricks

## SSH Tuning

Add these lines to your `~/.ssh/config`:

```
Host gerrit.lix.systems
    User YOUR_GERRIT_USERNAME
    Port 2022
    # Keep sessions open for a bit in the background to make connections faster:
    ControlMaster auto
    ControlPath /tmp/ssh-%r@%h:%p
    ControlPersist 120
```

Now you can use `ssh://gerrit.lix.systems/lix` instead of `ssh://USERNAME@gerrit.lix.systems:2022/lix` URL, and have faster iterations on a change!

## Splitting Commits

Sometimes a commit that was supposed to be a single feature gets out of hand. You have options. `git rebase -i` is a "default" suggestion, but you might want to look at `git-revise` or even at Jujutsu, which has a pretty cool squash workflow to do this!

# Advanced Features

If you feel confident in your Gerrit-fu, this page is for you. Perhaps you've already noticed how Gerrit brings a few good improvements to the workflow just based on the commit-centric design. Now we are getting to the really cool stuff: Gerrit-specific features that further enhance your experience!

## WIP Changes

Want to mark a change as WIP? You can do it in the Web UI. Or you can do it immediately on push:

```
git push origin HEAD:refs/for/main%wip
```

## Submitting Chains of Changes

In traditional Git forges, it is hard to submit multiple changes to one repo at once. Perhaps you want to avoid unnecessary rebuilds - or broken intermediate states. You might have to coordinate with people, plan downtime, or create meta-PRs to do that.

Thankfully, Gerrit has topics for that! Assign a topic to necessary CLs, or set it on push:

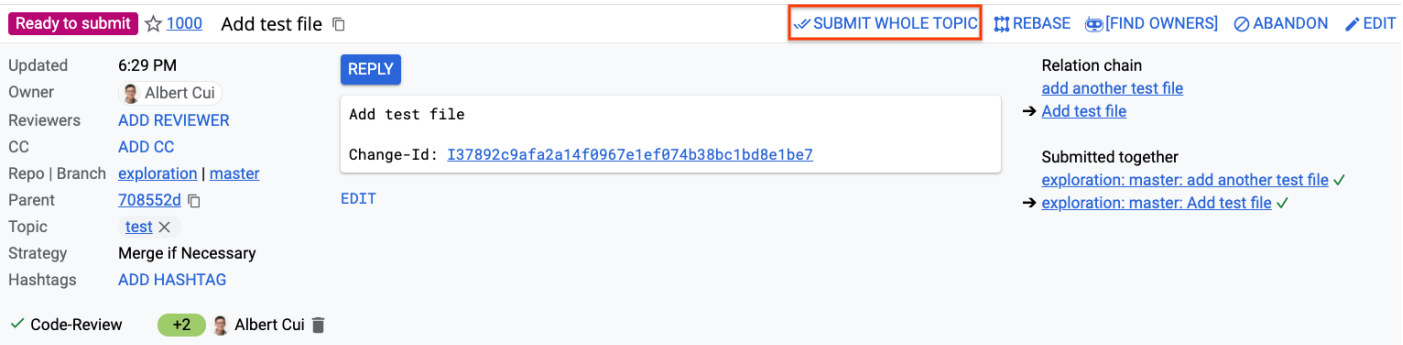
```
git push origin HEAD:refs/for/master%topic=<username>-<ISO8601 date>-<topic name>
```

For example:

```
git push origin HEAD:refs/for/master%topic=kfearsoff-2025-03-29-rebranding
```

Why the convoluted naming scheme? Unfortunately, Gerrit doesn't namespace topics, so we do a funny naming scheme to prevent accidental collisions. We are truly sorry for this.

Once all changes in a topic are reviewed, they can be gracefully sent all at once!



# Syncing Multiple Repos

In traditional Git forges, it is always a huge pain when you need to submit a batch of changes at once to multiple repos.

Topics to the rescue, again! Just follow the instructions for chains of changes, but in multiple repos. Isn't that awesome?

# Troubleshooting

## "Remote Unpack Failed" on push

Run a `git fetch` and try again.

## Re-run a CI

Do an empty commit amend: `git commit -a`. This will change the commit ID (because the date has changed!), so it will now be a new CL revision: you can `git push` it and pray for good CI run!

# Working in the Lix codebase

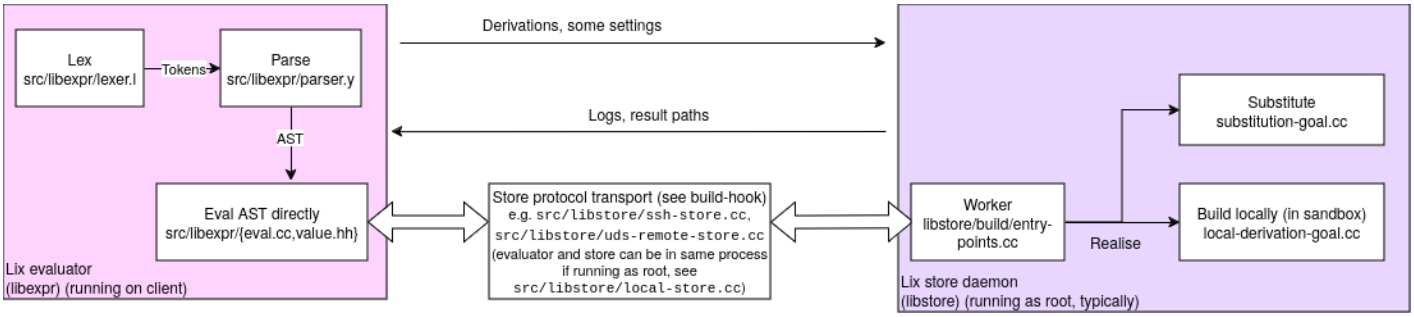
See also: https://git.lix.systems/lix-project/lix/src/branch/main/doc/manual/src/contributing/hacking.md

# Codebase overview

The Lix system is constituted of two broad parts, the evaluator and the store daemon. The two pieces may run on the same machine or on different machines.

For example, in a remote build setup like https://hydra.nixos.org, one node is running several evaluators in parallel, and builds are dispatched to several builder nodes.

(fyi to anyone editing this, double click the image in the preview to edit it)




# Evaluator

The evaluator is an AST tree-walking evaluator with lazy semantics.

Notable files:

- libexpr/value.hh, which defines the interface for evaluated values' interactions.
- libexpr/eval.cc, where most of the evaluator is.
- libexpr/nixexpr.cc, where the most of the `nix::Expr` class hierarchy is implemented, which are the AST types for the evaluator.
- libexpr/primops.cc, defining builtins.
- libexpr/parser.y, the (current) yacc generated parser.
- libexpr/lexer.l, the bison-generated lexer.

## Known design flaws

- GC issues (FIXME add details)
  - General tendencies to leak memory. Hydra restarts the evaluator every so often if it runs out of memory.
- AST based evaluator design limits perf
- Stack tracing has issues that make the traces confusing (FIXME add details)
- Funniness with attr ordering and equality that nixpkgs depends on, which is fragile

    - Currently no real tools to diagnose this and stop nixpkgs from depending on it.

      https://github.com/NixOS/nix/pull/8711 exists but regresses perf a lot and is not mergeable.
- Evaluation-time build dependencies (often called IFD) block the evaluator rather than allowing other evaluation to proceed
    - This has significant downstream effects such as typical derivations building hand-written large pieces rather than generated smaller pieces with IFD, since IFD is bad.
- The eval cache both has false hits and false misses, and needs redesign.

### Lix team plans

- Rewrite parser (done, being ported for 2.91 by horrors)
- Rewrite evaluator to be amenable to moving to bytecode (horrors) (long term)
- Do something about GC (long term)

## Store protocol

The store protocol is a hand rolled binary protocol with monotonically increasing versioning. It runs over a few different transports such as ssh (`src/libstore/ssh-store.cc`) or Unix sockets (`src/libstore/uds-remote-store.cc`).

### Known design flaws

- We cannot extend the store protocol (not that it is Good) because of the monotonic version numbers: we must always be stuck at *some* released CppNix version. This significantly moves up the need to replace it.
    - The code is significantly tangled with the current protocol design.

### Lix team plans

- Replace protocol with capnproto, transport with websockets?
    - Would likely be in addition to existing protocol; existing protocol likely would be run through a translator.

## Store daemon

The store daemon takes derivations (≈ execve args and dependencies) and realises (builds or substitutes) them. It also implements store path garbage collection.

Lix's local store implementation currently uses a SQLite database as the source of truth for mapping derivation outputs to derivations as well as maintaining derivation metadata.

Notable files:

- libstore/build/local-derivation-goal.cc, which implements the local machine's builder including the sandbox

- libstore/build/entry-points.cc, the server side entry points of the store protocol
- libstore/daemon.cc
- libstore/uds-remote-store.cc, the client implementation of Unix socket stores

## Known design flaws

- Sandbox is of dubious security especially on Linux (where it is actually expected to be somewhat secure)
  - Overall tangled code around the sandbox, particularly in platform specific parts
- Poor self-awareness: daemon doesn't know what it is building
  - Due to this plus the protocol being frozen, it would be very hard to implement e.g. dropping into a shell on failed builds
- Substitutions are inherently a kind of build so they can't happen out of dependency order or with better parallelism
- SQLite database has a habit of getting corrupted (probably due to Lix-side misuse)

## Lix team plans

- Replace sandbox with other software, perhaps bwrap
- Fix daemon self-awareness, add protocol level features to make this better
- Rearchitect substitution to enqueue weakly ordered jobs that happen in parallel and can resume downloads
- Switch to xattrs as the source of truth of store path metadata such that the SQLite DB can be completely rebuilt

# Freezes and recommended contributions

# Suggested contributions

Consider taking an issue marked E-help wanted: assign it to yourself and have a go. Feel free to ask for help in the development channel. The Lix team wants these issues fixed, but they are not high on our agenda to fix ourselves.

When in doubt, please ask the Lix team before beginning work, to make sure it is in line with our current priorities.

## I don't wanna write C++. How can I help?

This is totally reasonable, C++ is not super fun, but this isn't to say there isn't help if you do choose to. Nonetheless, there are still many issues on our tracker that do not require writing any C++ that would help out immensely.

Here's some possible places to look:

- docs issues on lix-project/lix. These generally involve writing documentation prose.
- testing issues on lix-project/lix. A large number of these are just writing shell scripts, pytest scripts (`tests/functional2` in lix), or fixing NixOS tests
- external project issues. This is a veritable grab bag of issues in external projects that we would love to have fixed. There's some in Gerrit (Java), some in Forgejo (Go), some in meson (python), some in nixpkgs. It's unlikely these will get done without some help so we really appreciate help!
- website issues our homepage still need some love in various areas. It uses Hugo as a static site generator, so contributing requires little more than some basic HTML and CSS skills.

## New builtins

One thing that is a little bit tricky and can get contributions canned late in the process is new builtins. We should write up a more full document on this, but the gist is that the API of any builtins needs to stand the test of time as they have pretty strong compatibility and usefulness expectations.

As such, it would be greatly appreciated if work on builtins starts with an issue on lix-project/lix tagged RFD discussing the use case for the builtin, then, once there is rough agreement on the use case, write a simple document giving examples and design of the proposed API. Then, we can

gather feedback before too much time is put into implementations that might not see the light of day.

# Freezes

We expect to have `main` always be in a state to run on machines you care about, unconditionally. Nightly builds should not be a problem to run in production in any freeze state.

For the time being, Flakes are frozen in their current feature set and will only receive maintenance bug fixes.

# Bug tracker organisation

We have a repo of directly imported nix bugs at https://git.lix.systems/nixos/nix. Please don't file bugs in there, we want the IDs to match. When we import a bug, we might put notes on there as we triage it, and potentially close it.

## Bug labels on NixOS/nix

- lix-import - Should be imported, we think it is still a bug
- lix-ignore - We don't care about this bug, it probably doesn't affect us
- lix-stability - Fixing this would improve the stability and reliability of Lix.

Dispositions:

- lix-norepro - Tried repro on upstream 2.18.1 and did not repro
- lix-retest-after-backports - Request that this be tested again once backports are done
- lix-reproduces-2.18 - Confirmed to repro in 2.18.
- lix-unclear-repro - Unsure how to repro but believe it affects lix
- lix-closed-libgit2 - Caused by libgit2
- lix-closed-lazy-trees - Caused by lazy trees

## Closed, marginal

- post-build-hook doesn't print a warning if not trusted-user https://git.lix.systems/NixOS/nix/issues/9790#issuecomment-273
- complaints about builtins.derivation https://git.lix.systems/NixOS/nix/issues/9774
- rejecting flake config still asks for confirm again https://git.lix.systems/NixOS/nix/issues/9788
- complaints of "substituter disabled", but is their bin cache just broken? https://git.lix.systems/NixOS/nix/issues/9749
- warn on eol https://git.lix.systems/NixOS/nix/issues/9556

# Gerrit

# FIXME: this page is duplicating a lot of content from the Gerrit section and should be considered deprecated

See the Gerrit section in this book for newer pages.

TODO: "Workflow tips" section seems to have got mangled in that transition and needs migration.

## What is Gerrit and why do people like it?

Gerrit is a code review system from Google in a similar style to Google's internal Critique tool, but based on Git, and publicly available as open source. It hosts a Git repo with the ability to submit changes for review and offers mirroring to other repos (like https://git.lix.systems/lix-project/lix). It has an entirely different review model to GitHub (and Forgejo, GitLab, etc that copy GitHub's review model), where, instead of pull requests, you have changelists (CLs): reviews on individual commits, with each revision of a commit being a different "patchset", rather than reviewing an entire branch at a time. CLs may be merged one by one or in a batch.

Although this has some learning curve, we expect that you will find it pleasant to work with after figuring it out. It has some rough edges and strong opinions that take some getting used to, but it has served us well and saved us an inordinate amount of time both as reviewers and change authors. The rest of this document gives some pointers on the workflows we use with Gerrit.

People like Gerrit because it makes the following things trivial or easy, all of which are somewhere between annoying and impossible on GitHub modeled systems:

Gerrit produces better code:

- Gerrit enforces good commit messages, since there is no second "pr message" so peoples' commit messages get actually looked at with some care
- Gerrit enforces good commit *hygiene*, since adding another commit is really just splitting a commit with `git revise -c` or other tools; since there are no PR dependencies or branches to worry about, splitting commits is no longer a big ask.
  - Relatedly, this directly makes reviews smaller since the overhead of doing another change is low.

Gerrit makes reviewers' lives easier and reduces review round trips:

- As a reviewer, you can look at what changed since you last reviewed, even in the presence of rebases, by looking at the patchset history of a CL. This avoids pointless rereview; you can actually diff versions of changes properly.
- The *change author* generally merges the change after approval, without them needing commit access. This means that they can do a final once-over of the change and make sure that they are ok with its state before merging it. This reduces miscommunication causing merging of unfinished code.
- As a reviewer, you can edit someone's change and/or commit message to fix a typo (*in the web interface*) and then stamp it, while giving them the final say on merging the edited change.
- You can give feedback like the following: "I would merge this as-is but you can consider this feedback if you would like" and then let the change author decide to merge it.
  - Since the permission-requiring step in Gerrit is *approving* the change, not merging it, every change author can have final say in when the change gets merged.
- Review suggestions get applied as a batch without cluttering commit history in a confusing manner.
- You can download someone's change to look at it locally in one command that you can copy paste from the Gerrit interface.

Gerrit makes your life easier as a contributor:

- Submitting a new change is just a matter of committing it and pushing it. You don't need to think about branches or the web interface or extra commands. Want to do more changes building on it? Just commit them and push them.
- Branches are not required and you can easily build off of other peoples' changes by fetching them and rebasing against them; change dependencies are simply commit parents. They can then be merged in whichever manner they will be merged.
- If you are doing a larger change, it is natural to merge it piece by piece, adding little improvements as you go, and putting the highest risk parts of it at the tip, making the obviously good parts of the change land and keeping your diffs and rebases against `main` smaller.
- Gerrit makes it clear which comments still need action in a clean way, compared to GitHub where resolved comments get regularly broken or disappear altogether.
- Gerrit guesses (with reasonable accuracy) who a change is blocked on and shows it on the dashboard with a little arrow next to their name, allowing you to see at a glance which changes are your responsibility at a given time.
- There is a rebase button that just works. Trivial non-conflicting rebases do not require a rereview.

That being said, there are some downsides:

- Gerrit is very mean to you if you don't have your commit history in a clean presentable state, which takes some getting used to and Git does not make editing history easy, so it does involve a little more fighting of Git. However, this also means that the reviews can be of cleaner and smaller pieces of code with fewer unrelated changes.

    - This makes pushing work in progress code with questionable commit history harder; see below for solutions to this.
    - **jujutsu fixes this**: if you are finding an amend based workflow frustrating, we highly recommend using Jujutsu in place of Git. See Jujutsu usage with Gerrit for more details.
- Gerrit requires a little bit of local setup in the form of adding your SSH key or setting up the HTTP password. It also requires a Git commit-msg hook, but `nix develop` automatically does that for you.

# Learning materials

- https://gerrit-review.googlesource.com/Documentation/intro-user.html
- https://docs.google.com/presentation/d/1C73UgQdzZDw0gzpaEqIC6SPujZJhqamyqO1XOHjH-uk/view

# Our installation

Gerrit is at https://gerrit.lix.systems

The Gerrit SSH server is running on port 2022. The repo URLs are:

- `ssh://{username}@gerrit.lix.systems:2022/lix`
- `https://gerrit.lix.systems/lix` if using HTTP auth; see Gerrit settings for setting an HTTP password if desired

Hit the d key on any change to download it, which will give you the right URLs.

## SSH config

You might like to add the following configuration to your `~/.ssh/config`:

```
Host gerrit.lix.systems
    User YOUR_GERRIT_USERNAME
    Port 2022
    # Keep sessions open for a bit in the background to make connections faster:
    ControlMaster auto
    ControlPath /tmp/ssh-%r@%h:%p
    ControlPersist 120
```

# Basic workflow for a change

The unit of code review is a "change", which yields a single commit when "submitted" (merged). The commit message is taken from the change description in Gerrit; in our experience this tends to

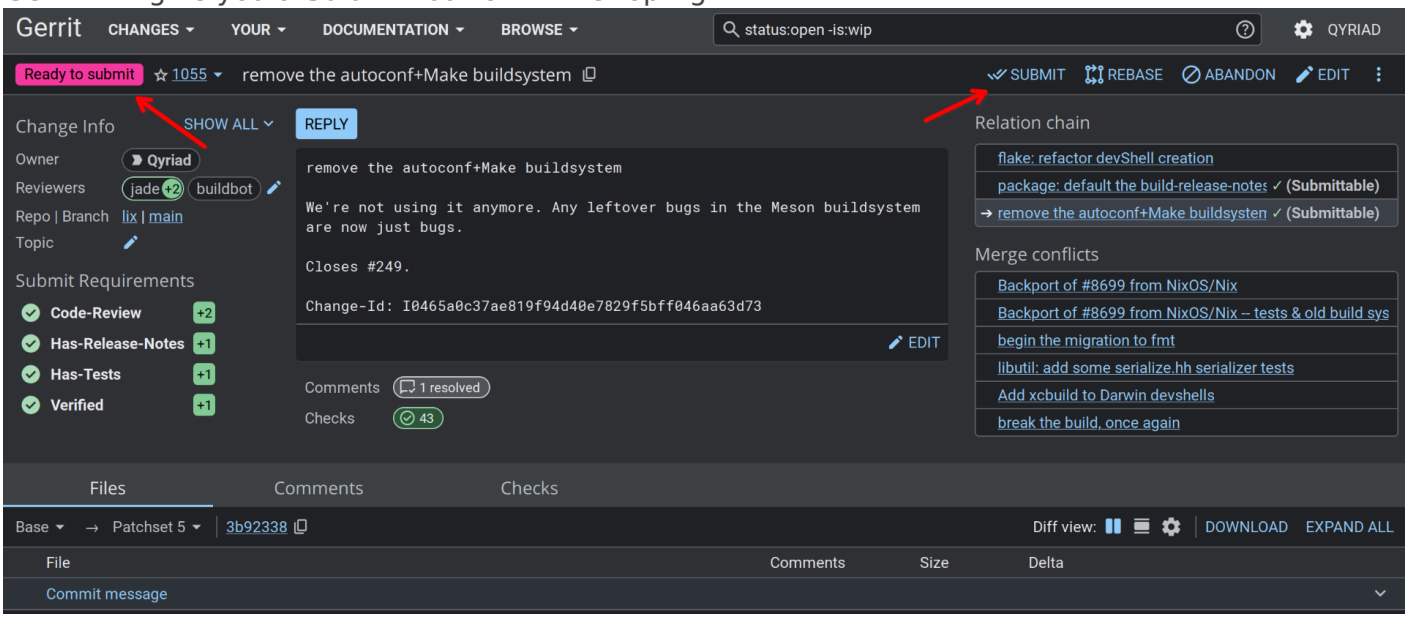
lead to more comprehensive commit messages.

For a change to be merged, it must have the following four "votes", in Gerrit's terminology:

- Set by reviewers:
  - +2 Code-Review: the committer that reviewed this thinks it can be submitted as-is (all users can vote +1/-1, expressing a weaker view on code acceptability)
  - +1 Has-Release-Notes: means the reviewer thinks your commit added relevant release notes for that commit, or that it does not need any. This serves primarily as a reminder.
  - +1 Has-Tests: means the reviewer thinks your commit added all the tests that commit needs, or that it does not need additional tests. Like Has-Release-Notes, this serves primarily as a reminder.
- Set automatically by CI:
  - +1 Verified: means CI successfully built for all our platforms and passed all tests

If you're newly part of the core team you will need to add yourself to the Gerrit `lix` group, otherwise you can't set the `Has-Release-Notes` or `Has-Tests` labels. If you're not, this doesn't affect you.

When all of those labels are set, a change becomes **Ready to submit**, in Gerrit's termology, and Gerrit will give you a **Submit** button in the top right:




By convention, ***the change author*** *has the final say on clicking the Submit button* (note: this is the opposite of the Github convention), and there is no special permission to merge a change once it has been fully reviewed (the permissions are in the reviewer +2'ing it). This gives you a last chance to have a look at your change before merging it.

# Workflow tips

## Local branches and commits

Gerrit is very mean to you if you don't have your local commit history in a linear presentable state, which takes getting used to but it is very low overhead once you get used to it. In short, amended commits become "patchsets", new commits become changes, and multiple commits help link your changes together as a "relation chain".

Note: if you're coming from Chromium, this is different to how they use Gerrit, where multiple commits become patchsets, and only the first commit on a local branch creates a new change.

Gerrit's commit-msg hook generates a new Change-Id for each commit you make, which in turn creates a new change that gets reviewed separately. To update an existing change after review feedback, amend or squash your changes into your old commit, keeping its Change-Id unchanged, then push.

Consider not pushing for review before it is clean, or split commits up with `git-revise` (good) or `jj` (better) after the fact, amending as you work. If you want a backup of your changes, you can fork it on Forgejo and push to that fork.

## Basic Pushing

If you cloned the repo from Forgejo, be sure to change your remote URL to point to Gerrit before continuing. Assuming your remote is called `origin` (which is the default):

```
git remote set-url origin ssh://{username}@gerrit.lix.systems:2022/lix
```

Then you can push to Gerrit with:

```
git push origin HEAD:refs/for/main
```

If you get tired of doing this every time, you can automate it by setting the `.git/config` as follows:

```
git config remote.origin.push HEAD:refs/for/main
```

You will have to do that in each fresh check-out. Once it's done, `git push` will work without additional options.

If you get a "remote unpack failed" error while pushing, run `git fetch` then try again.

If you wish to push a change and immediately mark it as WIP, you can push with `-o wip`, or make that the default behavior by checking `Set new changes to "work in progress" by default` in Gerrit's user settings, under "Preferences".

### Topics & Push Arguments

A Gerrit topic may be set on push with:

```
git push origin HEAD:refs/for/main%topic=foo
```

Which will create all pushed changes with the topic "foo". Topics are helpful for grouping long series of related changes.

A change may also be marked as "work in progress" on push:

```
git push origin HEAD:refs/for/main%wip
```

Gerrit has documentation on other push arguments you can use here, but it also takes a `help` argument whose output is more canonical and might be easier to understand, which you can view with:

```
git push origin HEAD:refs/for/main%help
```

At the time of this writing (2024/06/26), that output looks like this:

```
$ git push origin @:refs/for/main%help
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Processing changes: refs: 1, done
remote:
remote: Help for refs/for/branch:
remote:
remote:  --base BASE                              : merge base of changes
remote:  --cc CC                                  : add CC to changes
remote:  --create-cod-token                       : create a token for consistency-on-dema
remote:                                             nd (default: false)
remote:  --deadline NAME                          : deadline after which the push should
remote:                                             be aborted
remote:  --edit (-e)                              : upload as change edit (default: false)
remote:  --hashtag (-t) HASHTAG                   : add hashtag to changes
remote:  --help (-h)                              : display this help text (default: true)
remote:  --ignore-automatic-attention-set-rules : do not change the attention set on
remote:  (-ias, -ignore-attention-set)              this push (default: false)
remote:  --label (-l) LABEL+VALUE                 : label(s) to assign (defaults to +1 if
remote:                                             no value provided)
remote:  --merged                                 : create single change for a merged
remote:                                             commit (default: false)
remote:  --message (-m) MESSAGE                   : Comment message to apply to the review
remote:  --no-publish-comments (--np)             : do not publish draft comments
remote:                                             (default: false)
remote:  --notify [NONE | OWNER |                 : Notify handling that defines to whom
remote:  OWNER_REVIEWERS | ALL]                     email notifications should be sent.
```

```
remote:                                          Allowed values are NONE, OWNER,
remote:                                          OWNER_REVIEWERS, ALL. If not set, the
remote:                                          default is ALL.
remote:  --notify-bcc USER                      : user that should be BCC'd one time by
remote:                                          email
remote:  --notify-cc USER                       : user that should be CC'd one time by
remote:                                          email
remote:  --notify-to USER                       : user that should be notified one time
remote:                                          by email
remote:  --private                              : mark new/updated change as private
remote:                                          (default: false)
remote:  --publish-comments                     : publish all draft comments on updated
remote:                                          changes (default: false)
remote:  --ready                                : mark change as ready (default: false)
remote:  --remove-private                       : remove privacy flag from updated
remote:                                          change (default: false)
remote:  --reviewer (-r) REVIEWER               : add reviewer to changes
remote:  --skip-validation                      : skips commit validation (default:
remote:                                          false)
remote:  --submit                               : immediately submit the change
remote:                                          (default: false)
remote:  --topic NAME                           : attach topic to changes
remote:  --trace NAME                           : enable tracing
remote:  --wip (-work-in-progress)              : mark change as work in progress
remote:                                          (default: false)
remote:
To ssh://gerrit.lix.systems:2022/lix
 ! [remote rejected]     HEAD -> refs/for/main%help (see help)
error: failed to push some refs to 'ssh://gerrit.lix.systems:2022/lix'
```

## Pulling

Pulling from Gerrit will work normally. It's worth keeping in mind that sometimes a CL you're working on has been edited in the web UI or by another contributor, so the commit in your repo isn't the latest. Rebasing will usually make the duplicate go away; this is part of the normal rebase semantics, not Gerrit magic. You might consider making rebase-on-pull your default.

## Sandbox branches

> This feature has some notable ways to shoot yourself in the foot. We still support it, since it allows for running CI builds on things before they become proper CLs. If you don't need that and don't want to worry about the footguns, consider using a branch on a Forgejo fork for sharing WIP code.
>
> In particular, if a commit is in *any* branch already including a `sb/` branch, it will be rejected with the error "no new changes" if it is later pushed to `refs/for/main`. This can be worked around by amending all the commits so they are distinct, or by `git push origin HEAD:refs/for/main%base=$(git rev-parse origin/main)`, which forces the merge-base

Use `refs/heads/sb/USERNAME/*`.

### CI rerun

Push the CL again with a no-changes commit amendment if you want to force CI to rerun.

## Finding CLs to review

Consider bookmarking: https://gerrit.lix.systems/q/status:open+-is:wip+-author:me+label:Code-Review%3C2

# Improving build times

# Setup

Use a clang stdenv:

`nix develop .#native-clangStdenvPackages`

Then delete `build/` if you were using gcc before. Enable build-time profiling with:

`just setup; meson configure build -Dprofile-build=enabled`

Then run the build: `just compile`.

Enabling build-time profiling itself costs about 10% of compile time but has no other disadvantage.

# Build time reports

Use `maintainers/buildtime_report.sh build/` to generate a build time report. This will tell you where all our build time went by looking at the trace files and producing a badness summary.

## Sample report

**Build-time report sample**

```
lix/lix2 » ClangBuildAnalyzer --analyze buildtimeold.bin
Analyzing build trace from 'buildtimeold.bin'...
**** Time summary:
Compilation (551 times):
  Parsing (frontend):         1465.3 s
  Codegen & opts (backend):   1110.9 s

**** Files that took longest to parse (compiler frontend):
 10478 ms: build/src/libstore/liblixstore.so.p/build_local-derivation-goal.cc.o
 10319 ms: build/src/libexpr/liblixexpr.so.p/primops.cc.o
  9947 ms: build/src/nix/nix.p/flake.cc.o
  9850 ms: build/src/libexpr/liblixexpr.so.p/eval.cc.o
  9751 ms: build/src/nix/nix.p/profile.cc.o
```

```
  9643 ms: build/src/nix/nix.p/develop.cc.o
  9296 ms: build/src/libcmd/liblixcmd.so.p/installable-attr-path.cc.o
  9286 ms: build/src/libstore/liblixstore.so.p/build_derivation-goal.cc.o
  9208 ms: build/src/libcmd/liblixcmd.so.p/installables.cc.o
  9007 ms: build/src/nix/nix.p/..._nix-env_nix-env.cc.o

**** Files that took longest to codegen (compiler backend):
 24226 ms: build/src/libexpr/liblixexpr.so.p/primops_fromTOML.cc.o
 24019 ms: build/src/libexpr/liblixexpr.so.p/primops.cc.o
 21102 ms: build/src/libstore/liblixstore.so.p/build_local-derivation-goal.cc.o
 16246 ms: build/src/libstore/liblixstore.so.p/store-api.cc.o
 14586 ms: build/src/nix/nix.p/..._nix-build_nix-build.cc.o
 13746 ms: build/src/libexpr/liblixexpr.so.p/eval.cc.o
 13287 ms: build/src/libstore/liblixstore.so.p/binary-cache-store.cc.o
 13263 ms: build/src/nix/nix.p/profile.cc.o
 12970 ms: build/src/nix/nix.p/develop.cc.o
 12621 ms: build/src/libfetchers/liblixfetchers.so.p/github.cc.o

**** Templates that took longest to instantiate:
 42922 ms: nlohmann::basic_json<>::parse<const char *> (69 times, avg 622 ms)
 32180 ms: nlohmann::detail::parser<nlohmann::basic_json<>, nlohmann::detail::i... (69
times, avg 466 ms)
 27337 ms: nix::HintFmt::HintFmt<nix::Uncolored<std::basic_string<char>>> (246 times, avg
111 ms)
 25338 ms: nlohmann::basic_json<>::basic_json (293 times, avg 86 ms)
 23641 ms: nlohmann::detail::parser<nlohmann::basic_json<>, nlohmann::detail::i... (69
times, avg 342 ms)
 20203 ms: boost::basic_format<char>::basic_format (492 times, avg 41 ms)
 17174 ms: nlohmann::basic_json<>::json_value::json_value (368 times, avg 46 ms)
 15603 ms: boost::basic_format<char>::parse (246 times, avg 63 ms)
 13268 ms: std::basic_regex<char>::_M_compile (28 times, avg 473 ms)
 12757 ms: std::__detail::_Compiler<std::regex_traits<char>>::_Compiler (28 times, avg 455
ms)
 10813 ms: std::__detail::_Compiler<std::regex_traits<char>>::_M_disjunction (28 times,
avg 386 ms)
 10719 ms: std::__detail::_Compiler<std::regex_traits<char>>::_M_alternative (28 times,
avg 382 ms)
 10508 ms: std::__detail::_Compiler<std::regex_traits<char>>::_M_term (28 times, avg 375
ms)
```

```
  9516 ms: nlohmann::detail::json_sax_dom_callback_parser<nlohmann::basic_json<... (69
times, avg 137 ms)
  9112 ms: std::__detail::_Compiler<std::regex_traits<char>>::_M_atom (28 times, avg 325
ms)
  8683 ms: std::basic_regex<char>::basic_regex (18 times, avg 482 ms)
  8241 ms: std::operator<=> (438 times, avg 18 ms)
  7561 ms: std::vector<boost::io::detail::format_item<char, std::char_traits<ch... (246
times, avg 30 ms)
  7475 ms: std::vector<boost::io::detail::format_item<char, std::char_traits<ch... (246
times, avg 30 ms)
  7309 ms: std::reverse_iterator<std::_Bit_iterator> (268 times, avg 27 ms)
  7131 ms: boost::stacktrace::basic_stacktrace<>::basic_stacktrace (246 times, avg 28 ms)
  6868 ms: boost::stacktrace::basic_stacktrace<>::init (246 times, avg 27 ms)
  6518 ms: std::reverse_iterator<std::_Bit_const_iterator> (268 times, avg 24 ms)
  5716 ms: std::__detail::_Synth3way::operator()<std::variant<nix::OutputsSpec:... (182
times, avg 31 ms)
  5303 ms: nix::make_ref<nix::SingleDerivedPath, nix::DerivedPathOpaque> (178 times, avg
29 ms)
  5244 ms: std::__uninitialized_move_a<boost::io::detail::format_item<char, std... (246
times, avg 21 ms)
  4857 ms: std::make_shared<nix::SingleDerivedPath, nix::DerivedPathOpaque> (178 times,
avg 27 ms)
  4813 ms: std::__detail::_Synth3way::operator()<std::variant<nix::TextIngestio... (158
times, avg 30 ms)
  4648 ms: nlohmann::detail::json_sax_dom_callback_parser<nlohmann::basic_json<... (69
times, avg 67 ms)
  4597 ms: std::basic_regex<char>::basic_regex<std::char_traits<char>, std::all... (10
times, avg 459 ms)

**** Template sets that took longest to instantiate:
 55715 ms: std::__do_visit<$> (3603 times, avg 15 ms)
 47739 ms: std::__detail::__variant::__gen_vtable_impl<$>::__visit_invoke (11132 times,
avg 4 ms)
 43338 ms: nlohmann::basic_json<$>::parse<$> (85 times, avg 509 ms)
 43097 ms: std::__detail::__variant::__raw_idx_visit<$> (2435 times, avg 17 ms)
 32390 ms: nlohmann::detail::parser<$>::parse (83 times, avg 390 ms)
 30986 ms: nix::HintFmt::HintFmt<$> (1261 times, avg 24 ms)
 30255 ms: std::__and_<$> (25661 times, avg 1 ms)
 29762 ms: std::unique_ptr<$> (2116 times, avg 14 ms)
```

```
 28609 ms: std::__tuple_compare<$>::__eq (2978 times, avg 9 ms)
 27560 ms: nlohmann::detail::parser<$>::sax_parse_internal<$> (167 times, avg 165 ms)
 27239 ms: std::variant<$> (1959 times, avg 13 ms)
 26837 ms: std::__invoke_result<$> (10782 times, avg 2 ms)
 25972 ms: std::tuple<$> (5714 times, avg 4 ms)
 24247 ms: std::__uniq_ptr_data<$> (2116 times, avg 11 ms)
 24061 ms: std::__result_of_impl<$> (9029 times, avg 2 ms)
 23949 ms: std::__uniq_ptr_impl<$> (2116 times, avg 11 ms)
 21185 ms: std::optional<$> (2502 times, avg 8 ms)
 21044 ms: std::pair<$> (4989 times, avg 4 ms)
 20852 ms: std::__or_<$> (24005 times, avg 0 ms)
 20203 ms: boost::basic_format<$>::basic_format (492 times, avg 41 ms)
 20184 ms: std::tie<$> (2895 times, avg 6 ms)
 19938 ms: nlohmann::basic_json<$>::create<$> (668 times, avg 29 ms)
 19798 ms: std::allocator_traits<$>::construct<$> (5720 times, avg 3 ms)
 19182 ms: std::__detail::__variant::_Variant_base<$> (1959 times, avg 9 ms)
 19151 ms: std::_Rb_tree<$>::_M_erase (2320 times, avg 8 ms)
 19094 ms: std::_Rb_tree<$>::~_Rb_tree (2022 times, avg 9 ms)
 18735 ms: nlohmann::basic_json<$>::basic_json (243 times, avg 77 ms)
 18546 ms: std::__detail::_Synth3way::_S_noexcept<$> (2542 times, avg 7 ms)
 17174 ms: nlohmann::basic_json<$>::json_value::json_value (368 times, avg 46 ms)
 17111 ms: nlohmann::detail::conjunction<$> (907 times, avg 18 ms)

**** Functions that took longest to compile:
  2091 ms: _GLOBAL__sub_I_primops.cc (../src/libexpr/primops.cc)
  1799 ms: nix::fetchers::GitInputScheme::fetch(nix::ref<nix::Store>, nix::fetc...
(../src/libfetchers/git.cc)
  1388 ms: nix::Settings::Settings() (../src/libstore/globals.cc)
  1244 ms: main_nix_build(int, char**) (../src/nix-build/nix-build.cc)
  1021 ms: nix::LocalDerivationGoal::startBuilder() (../src/libstore/build/local-
derivation-goal.cc)
   918 ms: nix::LocalStore::LocalStore(std::map<std::__cxx11::basic_string<char...
(../src/libstore/local-store.cc)
   835 ms: opQuery(Globals&, std::__cxx11::list<std::__cxx11::basic_string<char...
(../src/nix-env/nix-env.cc)
   733 ms: nix::daemon::performOp(nix::daemon::TunnelLogger*, nix::ref<nix::Sto...
(../src/libstore/daemon.cc)
   589 ms: _GLOBAL__sub_I_tests.cc (../tests/unit/libutil/tests.cc)
   578 ms: main_build_remote(int, char**) (../src/build-remote/build-remote.cc)
```

```
   522 ms: nix::fetchers::MercurialInputScheme::fetch(nix::ref<nix::Store>, nix...
(../src/libfetchers/mercurial.cc)
   521 ms: nix::LocalDerivationGoal::registerOutputs[abi:cxx11]()
(../src/libstore/build/local-derivation-goal.cc)
   461 ms: nix::getNameFromURL_getNameFromURL_Test::TestBody() (../tests/unit/libutil/url-
name.cc)
   440 ms: nix::Installable::build2(nix::ref<nix::Store>, nix::ref<nix::Store>,...
(../src/libcmd/installables.cc)
   392 ms: nix::prim_fetchClosure(nix::EvalState&, nix::PosIdx, nix::Value**, n...
(../src/libexpr/primops/fetchClosure.cc)
   390 ms: nix::NixArgs::NixArgs() (../src/nix/main.cc)
   388 ms: update(std::set<std::__cxx11::basic_string<char, std::char_traits<ch...
(../src/nix-channel/nix-channel.cc)
   340 ms: _GLOBAL__sub_I_primops.cc (../tests/unit/libexpr/primops.cc)
   332 ms: nix::flake::lockFlake(nix::EvalState&, nix::FlakeRef const&, nix::fl...
(../src/libexpr/flake/flake.cc)
   305 ms: _GLOBAL__sub_I_lockfile.cc (../src/libexpr/flake/lockfile.cc)
   300 ms: nix_store::opQuery(std::__cxx11::list<std::__cxx11::basic_string<cha...
(../src/nix-store/nix-store.cc)
   296 ms: nix::parseFlakeRefWithFragment(std::__cxx11::basic_string<char, std:...
(../src/libexpr/flake/flakeref.cc)
   289 ms: _GLOBAL__sub_I_error_traces.cc (../tests/unit/libexpr/error_traces.cc)
   278 ms: nix::ErrorTraceTest_genericClosure_Test::TestBody()
(../tests/unit/libexpr/error_traces.cc)
   274 ms: CmdDevelop::run(nix::ref<nix::Store>, nix::ref<nix::Installable>)
(../src/nix/develop.cc)
   269 ms: nix::flake::lockFlake(nix::EvalState&, nix::FlakeRef const&, nix::fl...
(../src/libexpr/flake/flake.cc)
   257 ms: nix::NixRepl::processLine(std::__cxx11::basic_string<char, std::char...
(../src/libcmd/repl.cc)
   251 ms: nix::derivationStrictInternal(nix::EvalState&, std::__cxx11::basic_s...
(../src/libexpr/primops.cc)
   249 ms: toml::result<toml::basic_value<toml::discard_comments, std::unordere...
(../src/libexpr/primops/fromTOML.cc)
   238 ms: nix::LocalDerivationGoal::runChild() (../src/libstore/build/local-derivation-
goal.cc)

**** Function sets that took longest to compile / optimize:
 10243 ms: std::vector<$>::_M_fill_insert(__gnu_cxx::__normal_iterator<$>, unsi... (190
```

```
times, avg 53 ms)
  9752 ms: bool boost::io::detail::parse_printf_directive<$>(__gnu_cxx::__norma... (190
times, avg 51 ms)
  8377 ms: void boost::io::detail::put<$>(boost::io::detail::put_holder<$> cons... (191
times, avg 43 ms)
  5863 ms: boost::basic_format<$>::parse(std::__cxx11::basic_string<$> const&) (190 times,
avg 30 ms)
  5660 ms: std::vector<$>::_M_fill_insert(std::_Bit_iterator, unsigned long, bo... (190
times, avg 29 ms)
  4264 ms: non-virtual thunk to boost::wrapexcept<$>::~wrapexcept() (549 times, avg 7 ms)
  4023 ms: std::_Rb_tree<$>::_M_erase(std::_Rb_tree_node<$>*) (1238 times, avg 3 ms)
  3715 ms: boost::stacktrace::detail::to_string_impl_base<boost::stacktrace::de... (166
times, avg 22 ms)
  3705 ms: std::vector<$>::_M_fill_assign(unsigned long, boost::io::detail::for... (190
times, avg 19 ms)
  3326 ms: boost::basic_format<$>::str[abi:cxx11]() const (144 times, avg 23 ms)
  3070 ms: void boost::io::detail::mk_str<$>(std::__cxx11::basic_string<$>&, ch... (191
times, avg 16 ms)
  2839 ms: boost::basic_format<$>::make_or_reuse_data(unsigned long) (190 times, avg 14
ms)
  2321 ms: std::__cxx11::basic_string<$>::_M_replace(unsigned long, unsigned lo... (239
times, avg 9 ms)
  2213 ms: std::_Rb_tree<$>::_M_get_insert_hint_unique_pos(std::_Rb_tree_const_... (203
times, avg 10 ms)
  2200 ms: boost::wrapexcept<$>::~wrapexcept() (549 times, avg 4 ms)
  2093 ms: std::vector<$>::~vector() (574 times, avg 3 ms)
  1894 ms: bool std::__detail::_Compiler<$>::_M_expression_term<$>(std::__detai... (112
times, avg 16 ms)
  1871 ms: int boost::io::detail::upper_bound_from_fstring<$>(std::__cxx11::bas... (190
times, avg 9 ms)
  1867 ms: boost::wrapexcept<$>::clone() const (549 times, avg 3 ms)
  1824 ms: std::_Rb_tree_iterator<$> std::_Rb_tree<$>::_M_emplace_hint_unique<$... (244
times, avg 7 ms)
  1821 ms: toml::result<$> toml::detail::sequence<$>::invoke<$>(toml::detail::l... (93
times, avg 19 ms)
  1814 ms: nlohmann::json_abi_v3_11_2::detail::serializer<$>::dump(nlohmann::js... (39
times, avg 46 ms)
  1799 ms: nix::fetchers::GitInputScheme::fetch(nix::ref<$>, nix::fetchers::Inp... (1
times, avg 1799 ms)
```

```
  1771 ms: boost::io::detail::format_item<char, std::char_traits<char>, std::al... (190
times, avg 9 ms)
  1762 ms: std::__detail::_BracketMatcher<$>::_BracketMatcher(std::__detail::_B... (112
times, avg 15 ms)
  1760 ms: std::_Function_handler<$>::_M_manager(std::_Any_data&, std::_Any_dat... (981
times, avg 1 ms)
  1733 ms: std::__detail::_Compiler<$>::_M_quantifier() (28 times, avg 61 ms)
  1694 ms: std::__cxx11::basic_string<$>::_M_mutate(unsigned long, unsigned lon... (251
times, avg 6 ms)
  1650 ms: std::vector<$>::vector(std::vector<$> const&) (210 times, avg 7 ms)
  1650 ms: boost::io::basic_altstringbuf<$>::overflow(int) (190 times, avg 8 ms)

**** Expensive headers:
178153 ms: ../src/libcmd/installable-value.hh (included 52 times, avg 3426 ms), included
via:
  40x: command.hh
  5x: command-installable-value.hh
  3x: installable-flake.hh
  2x: <direct include>
  2x: installable-attr-path.hh

176217 ms: ../src/libutil/error.hh (included 246 times, avg 716 ms), included via:
  36x: command.hh installable-value.hh installables.hh derived-path.hh config.hh
experimental-features.hh
  12x: globals.hh config.hh experimental-features.hh
  11x: file-system.hh file-descriptor.hh
  6x: serialise.hh strings.hh
  6x: <direct include>
  6x: archive.hh serialise.hh strings.hh
  ...

173243 ms: ../src/libstore/store-api.hh (included 152 times, avg 1139 ms), included via:
  55x: <direct include>
  39x: command.hh installable-value.hh installables.hh
  7x: libexpr.hh
  4x: local-store.hh
  4x: command-installable-value.hh installable-value.hh installables.hh
  3x: binary-cache-store.hh
  ...
```

```
170482 ms: ../src/libutil/serialise.hh (included 201 times, avg 848 ms), included via:
  37x: command.hh installable-value.hh installables.hh built-path.hh realisation.hh
hash.hh
  14x: store-api.hh nar-info.hh hash.hh
  11x: <direct include>
  7x: primops.hh eval.hh attr-set.hh nixexpr.hh value.hh source-path.hh archive.hh
  7x: libexpr.hh value.hh source-path.hh archive.hh
  6x: fetchers.hh hash.hh
  ...

169397 ms: ../src/libcmd/installables.hh (included 53 times, avg 3196 ms), included via:
  40x: command.hh installable-value.hh
  5x: command-installable-value.hh installable-value.hh
  3x: installable-flake.hh installable-value.hh
  2x: <direct include>
  1x: installable-derived-path.hh
  1x: installable-value.hh
  ...

159740 ms: ../src/libutil/strings.hh (included 221 times, avg 722 ms), included via:
  37x: command.hh installable-value.hh installables.hh built-path.hh realisation.hh
hash.hh serialise.hh
  19x: <direct include>
  14x: store-api.hh nar-info.hh hash.hh serialise.hh
  11x: serialise.hh
  7x: primops.hh eval.hh attr-set.hh nixexpr.hh value.hh source-path.hh archive.hh
serialise.hh
  7x: libexpr.hh value.hh source-path.hh archive.hh serialise.hh
  ...

156796 ms: ../src/libcmd/command.hh (included 51 times, avg 3074 ms), included via:
  42x: <direct include>
  7x: command-installable-value.hh
  2x: installable-attr-path.hh

150392 ms: ../src/libutil/types.hh (included 251 times, avg 599 ms), included via:
  36x: command.hh installable-value.hh installables.hh path.hh
  11x: file-system.hh
```

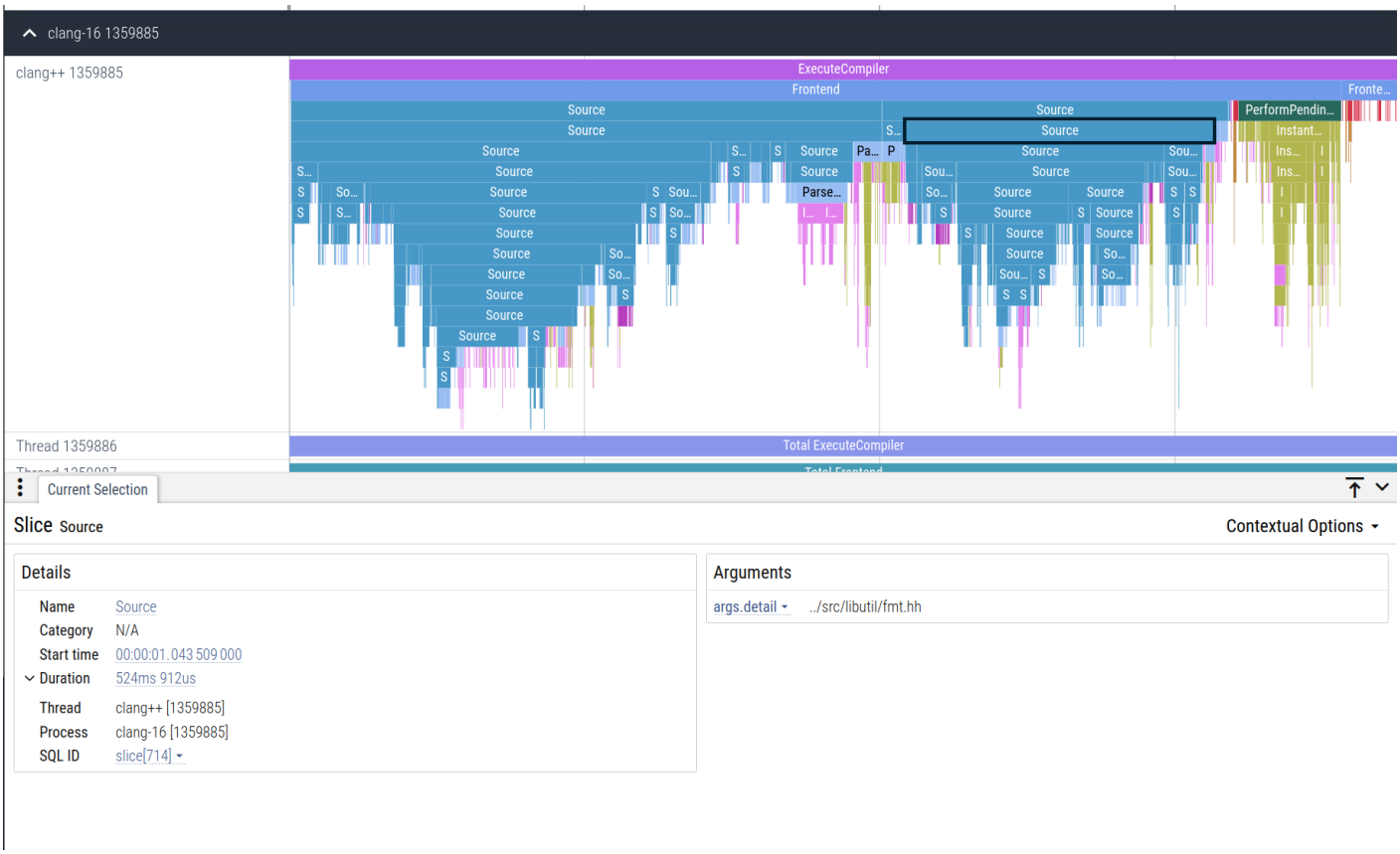
```
  10x: globals.hh
  6x: fetchers.hh
  6x: serialise.hh strings.hh error.hh
  5x: archive.hh
  ...

133101 ms: /nix/store/644b90j1vms44nr18yw3520pzkrg4dd1-boost-1.81.0-
dev/include/boost/lexical_cast.hpp (included 226 times, avg 588 ms), included via
:
  37x: command.hh installable-value.hh installables.hh built-path.hh realisation.hh
hash.hh serialise.hh strings.hh
  19x: file-system.hh
  11x: store-api.hh nar-info.hh hash.hh serialise.hh strings.hh
  7x: primops.hh eval.hh attr-set.hh nixexpr.hh value.hh source-path.hh archive.hh
serialise.hh strings.hh
  7x: libexpr.hh value.hh source-path.hh archive.hh serialise.hh strings.hh
  6x: eval.hh attr-set.hh nixexpr.hh value.hh source-path.hh archive.hh serialise.hh
strings.hh
  ...

132887 ms: /nix/store/h2abv2l8irqj942i5rq9wbrj42kbsh5y-gcc-
12.3.0/include/c++/12.3.0/memory (included 262 times, avg 507 ms), included via:
  36x: command.hh installable-value.hh installables.hh path.hh types.hh ref.hh
  16x: gtest.h
  11x: file-system.hh types.hh ref.hh
  10x: globals.hh types.hh ref.hh
  10x: json.hpp
  6x: serialise.hh
  ...

  done in 0.6s.
```

# Manually looking at traces

Note that the summary in the report can miss details like *why* one particular header is bad; to find that out, use a trace viewer to inspect the JSON trace file; we suggest `rg -t json -uu error\.hh build/ | less` to find some `.cc` trace that the bad header (in this example, `error.hh`) appears in.

You can look at individual file traces by opening some file like `build/src/libcmd/liblixcmd.so.p/command.cc.json` in https://ui.perfetto.dev or another Chrome-trace-json compatible trace viewer like Speedscope.

This will produce a flamegraph of the trace (screenshot shows Perfetto):




The most general spans of compile time are at the top, and the constituent spans are shown as you go down.

# Successful build time reduction CLs

- cl/588: 15%
- cl/1351: 10%

See the build-time-optimisation Gerrit topic for more related things.

# Commit porting (cherry-picking) guide

This guide is primarily about CppNix->Lix commit ports and was written when we were doing a *lot* of those before 2.90. It also applies to Lix->Lix ports, though, those are much easier, often just taking clicking "cherry pick" in the gerrit UI.

Don't forget, using Gerrit is a bit different than other systems.

## Lix -> Lix ports

We generally only backport fixes from `main` if there are severe bugs that users will hit in production, or if we had to patch the derivation in nixpkgs. The only real thing to remember about this is to use `git cherry-pick -x`. Gerrit will match up the Change-Ids and link the CLs on release branches to the other CLs on release branches or main. Then, push as if any other change: `git push origin HEAD:refs/for/release-2.92` (e.g.).

You can also use the cherry-pick button in the Gerrit UI.

## CppNix -> Lix ports

Commit style:

- Remember to use `git cherry-pick -x` to generate the `(cherry-picked from abc)` line.
- It's strongly preferred to include a link to the PR as `Upstream-PR: https://github.com/nixos/nix/pulls/123` in the trailer of the commit message.

GitHub encourages messy branch history, so the cherry-picking procedure can be rather annoying in a codebase like Lix which mandates that every commit passes CI. However, this document describes how we have generally done it.

Also note that we have renamed `src/` to `lix/`, which may or may not confuse git sometimes.

# Procedure (focused mostly on CppNix -> Lix)

## single commits

try `git cherry-pick -x` first. if this works, excellent. if not, apply the usual cherry-picking procedures:

- track down apply failures to intermediate changes. maybe cherry-pick those first if they're not too awful, but experience shows that they usually are too awful

- anything that touches the store or contains the word `Accessor` in the vicinity probably needs to be picked about and rewritten from scratch. that stuff is evidence of the CppNix version of lazy trees which is not in Lix.
- *always* test ofc, even if something applies cleanly it will sometimes just fail to build (or worse, run)
- nix prs tend to have broken inner commits. it is often necessary to pick parts from later commits in a pr to fix ci, in that case note this down as `fixes taken from <commits...>`
- sometimes a commit may be so broken that it can't reasonably be fixed except by squasing it with some other commit, in that case just squash them and not it down somehow (either with multiple `cherry picked from` commit hashes or multiple commit message+cherry-pick-hash blocks, depending on whether the fix messages were any useful)

# full prs

single-commit prs were mostly picked using `cherry-pick -x -m1` to keep the association with the upstream pr number for clarity. this implicitly squashes the pr into a single commit so it's only useful for single-commit prs. (some prs that have broken intermediate commits also benefit from this, but see above for that)

when pushing these to gerrit please set a topic like `backport-<pr-number>` using push options (`-o topic=backport-<pr-number>` in `git push`) to delineate one picked pr from a pr that depends on it

# Misc tips

## pyright is not dealing with import paths correctly for `functional2`

I don't know why it's doing this but it seems to be assuming that the functional2 test suite is relative to the workspace root.

You can jam the following into `:CocLocalConfig` for vim, and probably the workspace config for VSCode and it should fix it.

```
{
    "python.analysis.include": ["tests"],
    "python.analysis.extraPaths": ["tests"],
}
```

## buildbot user style to make the pulsing pills bearable

```
@keyframes pulse_animation {
  0% { transform:scale(.9) }
  50% { transform:scale(1) }
  to { transform:scale(.9) }
}

.pulse {
    animation-duration: 10s !important;
}
```

FIXME: someone should PR this, now that we have the ability to patch buildbot

## run all lix vm tests locally

```
tests=$(
    nix eval --json --impure \
        --apply '
            let f = n: t:
                if __isAttrs t
```

```
                then (if t.type or "" == "derivation"
                    then (if t.system == __currentSystem
                            then [ n ]
                            else [])
                    else __concatMap (m: f "${n}.${m}" t.${m}) (__attrNames t))
                else [];
            in f ".#hydraJobs.tests"
        ' \
        .#hydraJobs.tests \
        | jq -r '.[]'
)

nix build --no-link -L ${tests[@]}
```

# check out current patchset of a cl by git alias

put this in a gitconfig that can configure aliases:

```
[alias]
	cocl = !\
  ps=$(\
    ssh $(git config remote.origin.gerriturl) \
      gerrit query --format=json --current-patch-set $1 \
        | jq -sr .[0].currentPatchSet.ref \
  ) && git fetch origin $ps && git checkout FETCH_HEAD && true
```

then run as `git cocl <cl-number>`. needs a `git config remote.origin.gerriturl gerrit.lix.systems`, or some other url that ssh can connect to. (could've extracted it from the remote url but we didn't want to do that much shell)

# git stuff

## git-revise

git-revise is a cool tool for splitting and shuffling commits in-memory without breaking your working tree. it's great.

It also has some broken stuff with respect to gerrit commit-msg hooks. However, this can be fixed (this is opt-in because some commit-msg hooks make unsound assumptions but the gerrit one should be fine):

```
# allow gerrit git hooks to run on git-revise
[revise "run-hooks"]
        commit-msg = true
```

## Making `git clean` clean the stuff that isn't removed by `make clean`

If you don't want to use `git clean -x` to remove all git-ignored stuff, but want to remove things that are generated in Lix's build process but aren't removed by `make clean`, apply this patch: no-ignore-not-cleaned.patch

# Building Locally

See hacking.md in the Lix repo for the main documentation. Extra tips can go here.

# RISC-V support

Goal: install lix on a riscv64-linux system

The target is a DevTerm R-01, so it's an AllWinner D1 RISC-V processor @ 1GHz, with 1GB of memory and 32GB of microSD.

We can't run the Lix installer without building it, because there's no canned build for it. So let's try building it natively:

```
$ rustup
$ git clone https://git.lix.systems/lix-project/lix-installer
$ cd lix-installer
$ RUSTFLAGS="--cfg tokio_unstable" cargo install --path .
```

This doesn't work because there's some conditional complication^Wcompilation that doesn't cover riscv64. So we need to open `self_test.rs` and add an entry:

```
    #[cfg(all(target_os = "linux", target_arch = "riscv64"))]
    const SYSTEM: &str = "riscv64-linux";
```

At this point, it will, in principle, build. In practice, however, 1GB is just not enough RAM. If you add some swap it'll make it to the last step, but then it wants 1.5GB+ for that. I wouldn't try it on a system with less than 2GB, and ideally more.

Ok, native build is a bust unless I want to let it thrash all night. So let's cross-compile it on `ancilla`, which is, conveniently, already running nixos.

The nix-installer flake doesn't come with riscv64 cross support, and rather than try to figure it out I just winged it with nix-shell. I am skipping over a lot of false starts and blind alleys here as I ran into things like dependency crates needing a cross-compiling gcc, or rust not having a stdlib on riscv64-musl.

```
$ git clone https://git.lix.systems/lix-project/lix-installer
$ cd lix-installer
$ $EDITOR shell.nix
with import <nixpkgs> {
  crossSystem.config = "riscv64-unknown-linux-gnu";
};
mkShell {
```

```
  nativeBuildInputs = with import <unstable> {}; [ cargo rustup ];
}

$ nix-shell
[long wait for gcc to compile]

$ export RUSTUP_HOME=$PWD/.rustup-home
$ export CARGO_HOME=$PWD/.cargo-home
$ rustup default stable
$ rustup target add riscv64gc-unknown-linux-gnu
$ edit src/self_test.rs
[apply that same patch to SYSTEM]
```

The build invocation is a bit more complicated here, because we need to tell it where to find the linker:

```
$ RUSTFLAGS="--cfg tokio_unstable" cargo build \
    --target riscv64gc-unknown-linux-gnu \
    --config target.riscv64gc-unknown-linux-gnu.linker='"riscv64-unknown-linux-gnu-gcc"'
[another long wait]

$ file target/riscv64gc-unknown-linux-gnu/debug/lix-installer
target/riscv64gc-unknown-linux-gnu/debug/lix-installer:
  ELF 64-bit LSB pie executable, UCB RISC-V, RVC, double-float ABI,
  version 1 (SYSV), dynamically linked,
  interpreter /nix/store/g4xam7gr35sziib1zc033xvn1vy9gg8m-glibc-riscv64-unknown-linux-gnu-
2.38-44/lib/ld-linux-riscv64-lp64d.so.1,
  for GNU/Linux 4.15.0, with debug_info, not stripped
```

Since we couldn't do a static musl build it needs the nix ld.so, but we can get around that!

```
$ scp target/riscv64gc-unknown-linux-gnu/debug/lix-installer root@riscv:.
$ ssh root@riscv
# ./lix-installer
-bash: ./lix-installer: no such file or directory

# ldd ./lix-installer
/nix/store/.../ld-linux-riscv64-lp64d.so.1 => /lib/ld-linux-riscv64-lp64d.so.1
[other output elided]
```

```
# /lib/ld-linux-riscv64-lp64d.so.1 ./lix-installer
The Determinate Nix installer (lix variant)
[...]
```

Sadly we can't actually use it to install, because `nix_package_url` needs a default value, and on RISC-V, it doesn't have one! It's `self_test.rs` all over again except it doesn't manifest until runtime.

So, off to `src/settings.rs` we go. It doesn't need to be a valid URL, just something URL-shaped.

```
/// Default [`nix_package_url`](CommonSettings::nix_package_url) for unknown platforms
pub const NIX_UNKNOWN_PLATFORM_URL: &str =
    "https://releases.lix.systems/unknown-platform";
```

```
    #[cfg_attr(
        all(target_os = "linux", target_arch = "riscv64", feature = "cli"),
        clap(
            default_value = NIX_UNKNOWN_PLATFORM_URL,
        )
    )]
```

Rebuild, re-push, re-run:

```
# /lib/ld-linux-riscv64-lp64d.so.1 /opt/lix-installer install linux
Error:
   0: Planner error
   1: `nix-installer` does not support the `riscv64gc-unknown-linux-gnu` architecture right
now
```

Ok, missed a few places in settings.rs, let's put a quick and dirty hack in there:

```
            #[cfg(target_os = "linux")]
            (_, OperatingSystem::Linux) => {
                url = NIX_UNKNOWN_PLATFORM_URL;
                nix_build_user_prefix = "nixbld";
                nix_build_user_id_base = 30000;
                nix_build_user_count = 32;
            },
```

```
            #[cfg(target_os = "linux")]
            (_, OperatingSystem::Linux) => {
                (InitSystem::Systemd, linux_detect_systemd_started().await)
            },
```

It also needs a tarball to install; jade_ kindly updated the flake for it to support riscv64, so we just check it out (or, well, check out review branch 1444) and then `nix build -L .#nix-riscv64-linux.binaryTarball` and away we go.

This, it turns out, also doesn't work, because the installer is hardcoded to expect the directory the tarball contains to start with `nix-*`. You can either unpack and repack the tarball to meet that requirement, or find all the places in lix-installer that assume that and edit them -- they're in `src/action/base/move_unpacked_nix.rs` and `src/action/base/setup_default_profile.rs`.

Finally, this particular kernel lacks `seccomp` support -- in order to get it working, I had to edit the lix (not lix-installer) `package.nix` and add `(lib.mesonEnable "seccomp-sandboxing" false)` to the meson flags.

And with that done, it works!

```
root@devterm-R01:~# uname -a && nix --version
Linux devterm-R01 5.4.61 #12 PREEMPT Wed Mar 30 14:44:22 CST 2022 riscv64 riscv64 riscv64
GNU/Linux
nix (Lix, like Nix) 2.90.0pre20240613_dirty
```

# Branches

The Lix repository contains multiple releases in parallel. The branches work as follows:

- `main`. This contains *major* tags (except for 2.90 because of an early branch-off. We might fix that manually?), and is for the *next* major version of the software. This is where new development typically happens.
- `release-*`. These contain tags for `*.0` and further minor releases on a major release. We generally try to not backport things, since we would much rather get another major release out. (subject to revision; we would really like to not have LTS releases, but distro may make us do it?). These branches are *development branches* for a given release after it is released.
- (suggestion?) `stable-*` - Branch which is always pointed at the latest tag in that given major version.

# Version types

- Full release, e.g. 2.90.0. This is a snapshot of HEAD that we believe is stable for release and that we have fully performed out-of-tree validation on.
- Beta release, e.g. 2.90.0-beta.1. This is something that we would consider running in more or less any environment, given that we all run HEAD ourselves. This is an arbitrarily selected snapshot of HEAD that we are deciding to produce installers for, and is not special.
- Release candidate, e.g. 2.90.0-rc1. This is something that we would just release but it needs a bit more out-of-tree validation.

# Git tags

Git tags are created with the format `2.90.0`.

# Docker tags

- `latest` - The latest minor version of the latest major version
- `2.90` and similar - The latest minor version of the `2.90` major release.
- `2.90.0` - Exactly `2.90.0`.

# Working Groups

# What are they?

Lix doesn't have a very formal development structure - ice cream isn't a stranger to being fluid, after all! But all of the contributors have a different shape - so they are naturally drawn to different parts of the project. Documenting that is useful - for making communication more streamlined, and for a high-level overview on what activities are there!

People working on some part of the project are called "working group". But that's not all! There might be other people that are looking to help that area - they would be *interested to join!* Some cool projects depend on others, and so people who monitor the prerequisite project would be *tracking* it. And of course, some projects can be understaffed, in other words: *help wanted*.

## C++ Code

### IPC Rewrite

horrors is doing it, @raitobezarius and @jade_ on review

### Evaluator Changes

@piegames

## Rust Code

### Lix Installer

@KFears

Help wanted!

### NixLang parser rewrite

@piegames wants to do it, waiting for Rustification of the codebase

### Lix Logging

@jade_ wants to do it, blocked on Rustification

@KFears is interested to join

## Documentation

See this: https://wiki.lix.systems/books/lix-contributors/page/docs-rewrite-plans

There also exist issues in Lix bug tracker: https://git.lix.systems/lix-project/lix/issues?labels=151

# Integration Work

## Lix Rustification

@jade_ drives it

@KFears and @piegames are tracking the progress

## Infra

@raitobezarius

@KFears is interested to join (a bit :P)

## Release Engineering

@jade_ knows a lot, but the project hasn't seen much activity lately due to burnout

@KFears is interested to join

## Testing

Lix integration tests are a bit of a mess: some are in pytest, some are in Bash scripts

https://git.lix.systems/lix-project/lix/issues?labels=121

### tests/functional2

@Commentator2.0 and @helle mainly, with @piegames, @pennae and others handling reviews and helping out

## External Issues

There are a bunch of different ones! Some are integration work, some are Go, some are Java, some are Python, some are NixLang. Take a look!

https://git.lix.systems/lix-project/external-bugs

# Dealing with CI

## Logging in to CI for restarts

Go via https://buildkite.com/sso/lix-project and use your Lix account.

# Style Guide

Not just about code, a style guide is a list of decisions we've made, that we want to be consistent about going forwards. It does not need to be comprehensive of all possible issues, nor does it need to confine itself to trivial topics such as formatting. It's a tool for ourselves so we don't forget where we've been, and can avoid solving the same problems again.

Don't be shy about adding to it. Things written here do impose some burden, but the hope is that they lessen other burdens in the long run. Use your judgement about what's worth it.

Please fix style issues in existing code as you encounter them. Style is aspirational, a journey not a destination. :)

# Language and terminology

## Language

Most existing Lix documentation is written in British English. We intend to continue with that.

## Terminology

(FIXME: unsure if this should be in the style guide but ... it kinda should be -jade)

- **Nix language** - Use this to refer to the language which haunts us all.
- **Nix** - Nix refers to the *technology*. Used when referring to the Nix store, for example. Or, to a Nix derivation. Lix is a *Nix* implementation.
- **CppNix** - This is the preferred term for when it is necessary to refer to Nix, the software that Lix is forked from, rather than the technology.
- **Lix** - Use *Lix* when referring to the implementation. For example, "Install Lix".
- `nix`, `nix-build`, etc - Use lowercase `nix` when referring to the `nix` command, which is still supported by Lix.

# Code

# Code changes

## Tests

If at all practicable, all new code should be tested to some extent. If writing a test is hard, we need to prioritize making it easier, and potentially block features if that is the case.

## Documentation

Reference documentation should be added, in addition to release notes (`doc/manual/rl-next-dev`), for user visible changes.

For notable dev facing changes, consider adding release notes in `doc/manual/rl-next-dev`. This is not critical for all changes; in some cases it may make more sense to write it up in dev documentation instead, and indeed it may be ok to defer writing that dev documentation (it's helpful to create an issue to not forget).

## Benchmarking

Changes that touch the core of the evaluator or other performance critical code in Lix should be benchmarked.

See bench/README.md for instructions.

## Changelist size

If a CL is too long to review, it should be split up into smaller pieces with tests. The exact length varies but passing the 1000 line mark should give significant thought to splitting.

- When a CL is split, each commit should still be a valid state (tests passing, etc). If you must, you can gate in-progress changes with a flag or similar until the final commit. (Qyriad)

## Commit messages

Include at least a sentence or two as to why you are making a commit. For example, it can be nice to have the reproduction of a bug in the commit message. The commit message *is* the message for your review.

There's no particular format or specific style for commit messages; just make sure they're descriptive and informative.

# C++

While we hope to migrate the Lix interpreter from C++ to Rust eventually, C++ is a language that is likely to exist for a long time, and we may end up having to use it in other contexts.

Lix is a C++20 codebase. Features of C++20 that compile on all supported platforms can be used.

## NULL vs nullptr

`nullptr` where at all possible.

## Static vs anonymous namespace

Prefer anonymous namespace, both currently exist in the codebase (jade: any other opinions?).

## Type Aliases with `typedef` vs `using`

Prefer `using` declarations, as they can be used in more places, can be templatized, and have clearer syntax. Both currently exist in the codebase. (Qyriad)

## TODO/FIXME/XXX Comments

jade: this is not consistent with the conventions I use, needs further discussion imo (TODO: block in pre-commit hook, used in local tree but should never pass code review, FIXME(name||feature): its busted, someone should go fix it later, XXX: this is bad, we are writing down that it is ugly but leaving it as-is as we didn't figure out a better way)

Something along the lines of:

- **TODO**: acknowledgement that something is acceptably-for-now incomplete, especially if the scope of fixing it is high or unknown
- **FIXME**: this should be fixed before the feature or major change that it's a part of is considered "ready"
- **XXX**: this should not pass code review and should be considered a left-in mistake

## Header files

### Filenames

Headers should end with `.hh`. This reduces the likelihood anyone will try to include them from C files, which would require following the rules of both languages and is easy to get wrong.

The implementation of the functions declared in a `.hh` file should be in a `.cc` file of the same name, absent reasons to do otherwise.

### Order-independence

Headers should not care what order they're loaded in.

The exception, for now, is `config.h` in the `lix` repo. This must always come before all other headers. This observation should not be taken to imply it must always be that way, but at the moment it's helpful to be aware of.

### Idempotence

Use `#pragma once`, it helps. You can see this in most existing header files.

### `///@file` and header documentation

`///@file` should be at the top of all nix headers - Doxygen and other tools use it to decide whether a header should have documentation generated for definitions in it. See the relevant Doxygen documentation for more details.

Strongly consider adding a description of the purpose of a header file at the top of it in with `@brief A sentence saying what it is for`.

Examples:

```
/**
 * @file
 * @brief This header is for meow meow cat noises.
 */
```

```
/// @file
/// @brief meow meow meow
```

## Source files

### Filenames

Source files should end with `.cc`.

# Python

Python is a widely used tooling language in the Lix codebase. It's used in the test suite in `functional2`, a pytest based test suite intending to replace the Bash `functional` test suite. It's what the release engineering in `releng/` is written in.

In general, follow PEP-8 while writing Python code: identifier style should be the typical Python one per PEP-8, indents are 4 spaces, etc. In the near future, we will autoformat Python code with `ruff`.

## Type annotations

Type annotations should be used where they are helpful, and should be used for parameter and return types of functions. If it is too much of a bother to convince the type checker of something, it is acceptable to put an Any there.

Prefer inserting Any if the alternative is lying to the type checker via `# type: ignore` comments.

`dataclasses` is a useful library that produces nicer, more typed, code, and it is widely used. It's the correct choice for most classes which are just blobs of data.

## Doc strings

Please write them. Single-line ones are better than none at all.

We use the Sphinx docstring format for parameter documentation, though we currently do not generate Sphinx docs for our Python codebases. See this Sphinx tutorial for an example and the Sphinx domains reference for reference.

# Nix language

Unsurprisingly Nix contains Nix code. Some amount is tests and a lot is packaging.

We use the nixfmt formatter on files outside the test suite. It's run through `treefmt` with `pre-commit` hooks. Nix code outside the test suite is expected to be formatted.

Test suite files need not be formatted with the formatter at this time, but please consider doing so with new tests that don't rely on formatting.

## with

Prefer not to use `with` to bring things into scope as it obscures the source of variables and degrades language server diagnostics.

Use `let inherit (attrset) attrs` instead.

# Meson

Generally based on the style in Meson's docs made consistent and with a couple tweaks; notably multiline function calls are done in "block style" (think like rustfmt does it), rather than aligned,

e.g.:

```
executable('sdlprog', 'sdlprog.c',
  win_subsystem : 'windows',
  dependencies : sdl2_dep,
)
```

rather than:

```
executable('sdlprog', 'sdlprog.c',
           win_subsystem : 'windows',
           dependencies : sdl2_dep)
```

Meson's docs go back and forth on this, but we also put a space before and after the colon for keyword arguments (so `win_subsystem : 'windows'`, rather than `win_subsystem: 'windows'`).

# Operations

# Operational Conventions

## Code Review

### Self Stamping and Merging

On our Gerrit, core members have permissions to +2 any arbitrary CL, because sometimes we should be able to get something in quickly, and talk about it afterwards. In almost every case, the author of a change should not +2 their own CL, however Lix members may use their best judgement so long as they talk about it with the team when they can. Some cases where skipping synchronous review is a good idea:

- Reverting commits that accidentally broke main
- Fixing typos in other peoples' CLs that you would have +2'd, then +2'ing the edited CL
- Maybe typo fixes in `main`, though those can probably wait to be reviewed

Just make sure to talk about what you do :)

# Discussion notes

If you have a discussion or a meeting and are writing things down and want the meeting notes to persist, here is the correct place. Try to always move all relevant information off any scratch pads into appropriate more permanent places, like the wiki.

The template is optional and meant to help not forget any of the important aspects

# Discussion notes template

- Date:
- Participants:
- Topic:

## Conclusion / Action items

## Meeting notes / Discussion

Discussion notes

# 2024-04-29 – Lix Release Bootstrapping

## Agenda

Prohibited items:

- Nix foundation politics, for everyone's safety

Core Agenda:

- Figure out an initial governance model we can use to bootstrap governance later
- Figure out how to do a soft release
- Figure out anything that needs to be done before this release

Other suggested agenda items:

- What to use for our contribution policy in the interim.
  - Raito going to come up with contributor doc, asking others for help.
- What to use for ~~code-of-conduct~~ community standard in the interim.
  - Jade going to either come up with community standards or make it happen otherwise.
- Contributor License Agreement T_T_T_T
  - None (thank goodness)

## People

hexchen, Irenes, Irides, jade, ktemkin, puck, Osiria, Raito

## Decisions made

- Bootstrap governance is made via:
  - Consensus first
  - 3/4 of bodies is quorum for non-governance decisions, otherwise it's the full set of bodies
  - 2/3 of the quorum number of bodies for a vote to succeed
  - Anyone can call for a vote at any time, including after a decision was made, except if a decision has been binding
  - Decisions can be binding or not, if they are not binding, they can be relitigated
  - Binding decisions are governance decisions
  - Focused on social governance but technical matters are a special case of social governance in the current model
  - Consult the governance before speaking on project's behalf for controversial topics

    - Delegation and on-the-spot decisions should be used carefully and consensus should be obtained back as soon as possible after such a thing is done

put here for edit history reasons

https://gist.github.com/RaitoBezarius/e8509c1bdae2980031e9630de4b6d69e

- We should accept PRs, from day one of soft launch, regardless of tooling status
    - "We do accept them, we will make it work."
    - If it gets bad we'll yoink the Go tooling

# Action Items

- Write trivial community standards.
    - Write CoC that says "everyone is expected to follow community standards"
    - See governance pad thing with the general ideas. Want to write down principles of what we expect in the community.
    - jade owns this and causes it to happen
- Do something about continuous open beta
    - jade owns this and will make sure it doesn't get forgotten

# Notes

Meeting starts at roughly 20:10 UTC

## Governance Model

- irenes: how to avoid power structures being created by most jurisdictions forcing power structures for creating an entity. We would need an entity for money reasons due to infrastructure and needing to give people money
    - need to have an agreement on what we stand for overall
    - this is more of a consideration for the future
- kate suggestion: interim hypothesis: simple majority (of the core team list)
    - jade: constituency: people like Gabriella who are there for historical reasons but are not significant contributors
    - hexchen: 2/3 majority?
        - kate: ranked choice, or consensensus, approval voting also possible
    - puck: 50/50 split can happen, which can be a problem for simple majority
    - hexchen: for day to day decisions, 50% is good enough, higher level of agreement required to mess with governance
    - irides: consensus with majority vote to break ties?
    - kate: what is a quorum?
    - kate: if we can reach consensus on decisions, we can avoid doing things that people don't want/avoid using the voting system
    - kate: day to day vs governance decisions: what is a day to day decision? or we can treat every decision as governance decision

    - jade: what is a decision?
        - kate: some decisions are binding, some are made ad hoc. cannot rely on vibes to know when there might be disagreement
- hexchen: supports consensus for bootstrap but we do *need* a tiebreaker (2/3 reasonable)
- jade: should probably revert first in cases where things should need more discussion?
    - kate: stuff like potential security issues mean that revert-first doesn't work necessarily always
    - qyriad: revert-first favours the status quo
    - jade: this is kind of what we are doing
- hexchen: there is a difference between social and technical decisions, should focus to do social decisions in our governance model first. technical governance inflates the agenda for the meeting.
- raito: re revert first, consider a compromise: suspend people from involving themselves in a thing, for a period of time if things get heated?
- hexchen: re lazy consensus: make snap decisions, reëstablish consensus afterwards. we would prefer to not have to make such decisions, but if they are made, we should try to get consensus after they are made
- kate: use common sense, especially re speaking for the project. don't go around loudly speaking on the project's behalf without some consensus
- kate: what is the quorum? is the threshold *of the quorum*?
    - hexchen: possibility (remote for us) of excluding people maliciously and still forming quorum
    - kate: anyone outside the quorum can call revote?
        - jade: time bounds of revote?
    - puck: nixos foundation bylaws include some mechanisms to avoid quora being formed maliciously
        - some ways to deal with long term absentees
- kate: we want to make governance decisions in terms of actually making a real governance with the entire group of people. the entire group should be involved in the final governance making. everyone on the core team list should sign the final thing or so.
  two stage vote thing: vote initially and then everyone can change to be unanimous at the end as an explicit approval
- qyriad: concrete numbers: 2/3 is 7/10 bodies. 3/4 is 8/10.
    - kate: 3/4 for quorum, 2/3 for vote? everyone there -> binding decision, not relitigible
    - kate: something that is not relitigible is considered a governance decision, and thus requires a full quorum

summary by qyriad: generally try for consensus. when things have to be put to a vote, 3/4 is quorum, 2/3 of participants for vote. for governance quorum is everyone that can be reached. votes are done, for example, for decisions that either are expected to need agreement, or after the decision if agreement was necessary; also things that are project direction.

- irides: things that are votable but are not necessarily controversial.
- hexchen: how too moderation? generally want to reëstablish consensus after such decisions are made.
  - first put it on wiki then talk about it. kate: this is just a social expectation, does not necessarily need to be codified
- kate: need to write coc; general framework is using judgement based on safety of the community.
- raito: how does ownership of topics work? avoid cases of people doing diverging snap decisions on things outside their region
  - hexchen: delegating topics is something we need to do but not worry about *ownership* for bootstrap governance. delegate discussion space for certain things: e.g. ensure that discussions about things are in set places so that people see it.
  - irenes: consider how delegation is a hierarchical thing and avoid doing it immediately

## Moving on to the soft release

- hexchen: I would simply drop a link on social media and let the everything spread!
  - Kate: just telling people in social circles and social media (save for Irenes and Kate) is probably good for a soft release
- Kate: Oh right we need to do binary cache for substituting Lix
- hexchen: are we still going to be reaching out to the Nix/Tvix/Guix developers a few days beforehand?
  - Kate: Nix kind of already got their notice
  - Raito: Most of the Tvix members already know about Lix
  - Kate: the soft release also kind of *is* notice for the hard release
    - But we can also send the Guix devs an email or something
    - Irides: why are we doing this in the first place?
    - jade: for security reasons! Nix's security response stuff has historically been disasterious
    - Raito: we share componenets in our codebases with Guix, so it makes sense to communicate in general
- hexchen: should we be PRing Lix to Nixpkgs for soft release?
  - Kate: we probably shouldn't touch Nixpkgs until the Foundation stuff stops exploding
- hexchen: I would *really* like to avoid making Lixpkgs a thing
  - Osiria: gods we would really like this too
- we want to reach out to the guix project
  - raito: should we reach out to the guix lead
  - general agreement to just reach out to them politely to say that we exist
- jake hamilton: decided not to disclose to him
- puck: maybe we should just tell Guix at the same time as the soft release?
  - jade: doing it ahead of time is a show of good faith
- raito: re jake hamilton: their fork is a "we are trying to do this", don't think they have the relevant skills/insider stuff to really get far soon
  - jade: doesn't really see any advantage to disclosing early
  - kate: we're going to be out so soon anyway

    - kate: also doesn't really see a *downside* to disclosing to them ¯_(□ )_/¯
- hexchen: meow
    - maybe now is a good time to do a short couple-minute break
- reconvene at 21:25 UTC
- hexchen: how do we want to deal with expanding the core team?
    - should we continue the freeze?
    - Osiria: yes, we absolutely should continue the freeze
    - jade: the bootstrap model has loopholes that are exploitable. anyone to be onboarded would have to be deeply trusted
    - Kate: after we open up the "core team" distinction will likely break down as we do things more out in the open
    - jade: a lot of stuff lands in the core chat either because it's sensitive, or because Matrix is kind of bad, which actually really sucks
        - Having a less bad chat platform enables things more

## ~~Code of Conduct~~ Community Standard

- Kate: stuff on the wiki will probably be what we adopt long-term
    - In the meantime we should adopt something in the interm
    - Irides: maybe we should adopt something that just expresses intent instead of a traditional code of conduct
    - Kate: that's kind of a code of conduct with more steps — if someone has the time/spoons to write something up feel free
    - Raito: the team composition also kind of signals some degree of intent implicitly
    - Kate: code of conduct or not we're kicking out nazis and that expresses intent too
    - jade: coc explicitly a bad term
    - Irenes: if someone actually tries to call us out for no CoC yet we just point out that we actually have a functioning governance model and we're taking the time needed to figure out CoC stuff
    - hexchen: should we just do contributor covenant?
    - Irides & jade: that document would actually be a detriment
    - Kate: re: CoC vs CS: we just put a code of conduct that says "follow the community standards"
- Kate and jade both offered to do writing stuff
- How about jade takes ownership of the thing, and go ahead and probably write it, but if she doesn't/can't/etc then she can poke Kate who will write it
- hexchen: should we subscribe to NixOS's Matrix banlist?
    - Kate: pulling it in, sure, subscribing, probably not
    - jade: we can follow first and review after, or review first and follow after
    - Kate: we can also just follow their moderation decision repo
    - hexchen: draupnir can send a message when the banlist updates. or we can just have someone in the banlist room and action it

## Contributing Policy

- Kate: this should be liftable from somewhere?
    - jade + Irides: what is a contributing policy?

    - Kate: it's the set of expectations we have for contributors
- We already have some stuff on the wiki
    - jade: Should do some crosslinking between CONTRIBUTING.md and the wiki
    - jade: Maybe find a user to experiment on
- Irides: a lot of contributor documentation is not very good
    - Raito: nixpkgs mood
    - We should have good onboarding process documentation
    - puck: we may be conflating "how to use our tooling" and "what kind of contributions are we accepting"
        - External contributors should probably be aware that we're not generally making large changes right now
        - Irides: the wiki already has a document on that
        - jade: ideally before you write a single line of code you should know this information
        - Osiria & jade: a clone/devShell/pre-commit hook that informs the user of our freeze-state
        - Raito & Irides: this is largely a social problem
        - jade: can we put the contributing channel in the devshell hook or something?
            - osiria & irenes: explicitly say "we want to talk to you", link to the channel
            - puck: put CL push webhook into a room maybe? not necessarily now. generates sense of activity
            - kate: specific dedicated channel for "prospective contributors/feedback"
            - puck: concerned about discussion fragmentation with many channels
            - jade: maybe like... a threadbot? that makes temporary channels because Matrix threads suck?
            - puck: more projects we need =P
            - Kate: I would simply make Zulip have a good UI
    - jade: how do we MAKE people read the "freeze policy" or so
    - jade: should there be synchronization between contributing document and the wiki (probably with repo as source of truth)?
    - Osiria: noting: what should the wiki/forgejo/gerrit permissions be for new logged-in-with-github users?
- jade: if the contribution document just links to the wiki, you force people to go the wiki
    - if it has a bunch of links to the wiki, no one will read the links
    - maybe should just be a summary
    - kate: "ask first, we will help you, here is where to ask"
- jade: noting: do something about the github pr bot thing, or set up a manual process in the interim
- jade: what should we do about github PRs?
    - Thinks we should accept PRs, from day one of soft launch, regardless of tooling status
        - "We do accept them, we will make it work."
        - If it gets bad we'll yoink the Go tooling
- Kate: if we're going to do governance stuff on git, *there* we should probably sign our commits
    - jade: aggressively sighs and concurs

- Osiria: permissions for soft launch?
  - Kate: for soft launch, signin w/ Github is disabled for the public
    - jade: we really really want to have the majority of accounts especially from people we don't trust be from github due to being harder to make socks on it
  - Kate: or we could do local accounts
  - jade + Osiria: that breaks bans and is SSO-messy
  - Kate: deferred, put it on the soft release board
- Kate: CLAs suck and we're stuck with LGPL forever?
  - Irides: throw MIT license and Apache license in there, and future contributions are licensed under *all three* licenses, allowing us to eventually migrate
  - Kate: that doesn't really work since there's all the existing licensed code
  - Irides: MIT and Apache are LGPL-compatible
    - Really would not like to be confinded to LGPL forever
    - We already have in-tree mixed licensing
  - Kate: we can't do that for anything that touches LGPL code
  - Osiria: defer to asynchronous
- Decision: no CLA
- Raito: why a permissive license anyway?
  - Irides: deferred to asynchronous
- Irides: we cannot *possibly* do worse than Nix at this point.
  - jade: that fact that we're doing *any* of this is already a major improvement

---

- Raito: how do we transition from beta to soft release? what do we do with the beta rooms?
  - we just leave them
  - jade: these are people we generally trust, and who are politically all on the same page. we must continue to have this space exist, and also not grow it
  - Raito: we need to address the fact that people want to add people
  - jade: not opposed to growing them, just opposed to growing them to people who don't meet the same demographics as the ones who are in there
  - Kate: try not to stray too far into politicking for now
- jade: all the people in the beta are running main, and that's good actually, and if we could get more people to run main that would be awesome
  - great for feedback
  - maybe even have a channel for "I'm running main", and have a permanent open beta
  - Kate: sounds good! make an issue :)
- irides: how should we do linking to scratch?
  - kate: can put sso on it possibly
  - jade: the link can be shared to more people

---

Bootstrap meeting complete! Well done everyone ♡

Roughly 22:25 UTC, meeting end

Discussion notes

# 2025-06-27 Wiki work

- Date: 2025-06-27
- Present: jade, piegames
- Topic: Information organization in the wiki and the homepage

**Summary:** We have a mess of information in the pad system which is not well organized by quality, and it's hard to deal with it.

## Action items

- Create a new book for founding-days Lix pad in the Wiki, migrate stuff there
- Create a place for meeting notes in the wiki, with a nice template and everything
- Clarify whether people are intended to edit each section of the wiki (or like somehow blanket make it clear that it is okay absent further information)
- Maybe document a procedure for taking a matrix log (with LLM prompt for initial phase?) and turning it into more useful information
- piegames can take care of migrating the governance-related pads, in the sense of finalizing them and getting them approved and making them official (this will take a lot of time though)
- Rework https://wiki.lix.systems/link/9#bkmrk-freezes to reflect current processes
- Create a "Developers Announcements" channel for low-volume communication between all devs
- Migrate https://lix.systems/resources/ to the wiki

## Meeting notes

- Organization of the design documents is confusing
- Need to explain contributions better
  - The "how to not write C++" is ill-located
- Need governance meta related channel
- We should probably move the resources page
- We probably should have better organization of the "lix manifesto" related stuff
- Related projects is ill located
- Unclear where we want to put each thing, e.g. do community standards go on the website
- minor wiggles moment of the quicklinks page being added to the website but not being linked anywhere
- Stuff like cultural values and so need to escape the pad zone
- IFD discussion pads
- Discoverability problems
  - Wiki > Pads > Matrix

- Pads problems
  - We sometimes archive matrix discussions into pads
  - Throw them into a large language model lol
  - Eg. improving IFD pad
- Who does this work of archiving and librarian stuff
  - Could jade do this while tired at work?
- Information organization
  - Wiki: stuff that should not connote endorsement and does not allow people to mess with wiki
  - piegames: this is kind of a problem for the governance pages
  - The website has no way to have table of contents and nesting support
  - Failure mode of not caring too much about too wide edit permissions is that people don't think that they are allowed to edit it
- RFC process where it gets committed to a repo means that you know what actually is committed to
  - In general we cannot afford more process
- This will burn out piegames if they have to do all the coordination labour in the project
  - How do we make this sustainable?
- We can avoid having to do more of this by having a very minimal quality bar for the wiki and accept that it is going to become a mess
- Archived stuff
- Space for meeting notes
  - Write meeting notes with a template that requires that it says who was doing the meeting
- Issues with pads is that we don't separate trash and useful information
  - Pad index is a mixture of stuff that is partially dev log and partially dev result
  - We should organize the dev log by date/topic, rather than by topic
- piegames prefers a process and end result type note structure rather than these, which are a headache

Discussion notes

# 2025-06-26 Lix NixOS Module

- Date: 2025-06-26
- Participants: Niko, k900, piegames, Raito
- Topic: The current state of the Lix NixOS module (https://git.lix.systems/lix-project/nixos-module/)

## Conclusion / Action items

☑ piegames will reach out in *Lix on main* asking for volunteers

- Primary goal is to get rid of the module by integrating its overlay-style functionality into `pkgs.lixPackageSets`
- Stretch goal is to have a `lix` module in the NixOS name space for additional configuration

## Discussion

Selective transcript from a Matrix discussion

- niko: Is lix-module something that's planned to be gotten rid of? I'm asking because since Lix now requires cgroups for auto-allocate-uids, whereas cppnix doesn't (unless I'm wrong) then it'd be nice for lix-module to extend the nix module and add checks for stuff like whether cgroups is enabled if auto-allocate-uids is, and similarly to check whether nix-daemon service unit has cgroups configured, and other potential differences that come up that nixpkgs doesn't necessairly take into account
  - raito: answer is yes
- k900: I think we can upstream all of that to nixpkgs tbh
- piegames: Even the lix overlay?
- piegames: This functionality should still be available for non-NixOS though. The module does little more than setting the nix package option and applying an overlay that does all the nix lix substitutions in dependents
  - k900: But we can have things in lixPackageSets that are lix-overlay-shaped
  - raito: lixPackageSets does it too – lixPackageSets is the Nix-dependent universe reinstantiated with Lix
- raito: If someone wants to take the lead on Nixpkgs packaging, I can provide guidance, review time and help, but I cannot spearhead this until we landed the start of RPC in Lix I'd say
  - piegames: Do we want to publish a larger call-for-participation on this? Maybe on *Lix on main* for example. Raito would you be open to mentor such a thing?
    - raito: Agreed
- k900: What are we missing on the nixpkgs side?

    - raito: Double checking there's nothing missing between the module, implementing a deprecation notice on nixos-module side, I'd say. Instructions & docs
    - k900: Oh you mean not nixpkgs packaging of lix, but absorbing nixos-module to nixpkgs
    - raito: I meant the nixos-module replacement inside nixpkgs, yep
    - raito: Possibly, designing for `lix.enable = true;`, etc. We are increasingly diverging from CppNix, the `nix.*` namespace will feel constrained at some point
- raito: `import <nixpkgs> { config.nixImplementation = "lix"; }` something like this, which dumbs it down to an overlay or anything idc

Discussion notes

# 2026-02-11 Core team technical discussion

- Date: 2026-02-11
- Participants: Raito, horrors, rootile, piegames, jade, kate, Qyriad
- Topic: Core team discussion on technical vision and alignment alignment

## Conclusion / Action items

- @kate: suggestion, every topic we discussed above should be made into a Zulip thread ; please if you want to see a point you raised, please open a Zulip thread for it!

## Meeting notes / Discussion

Relevant documents:

- Technical vision document WIP from @piegames (behind Lix SSO)
- Dreams

### Agenda

- Context
- What do we want out of Lix on the short term?

### Context

Kate explains there has been changes in the team activity, a need to update expectations and communicate on this.

We all came in this project with a laundry list of what is wrong with CppNix. A lot of us had a dream about what a Nix implementation should be.

In terms of what happened in practice, in the context of the a fall of a large nation to fascism, the fall of the NixOS project, we all are kinda in a situation where that laundry list of problems we had and opportunities we saw has become a responsibility for a lot of things.

We have to overcome the list of what is wrong and align on an actual technical vision.

Something we have all in common is we want to have this software being able to use it on day-to-day and have it work. @piegames worked on an evolution of Nix into Nix2. horrors worked a lot on the correctness, the nastyness of the codebase and improve the state of the codebase. There might be also other people who want to do commercial-shaped things around. More implicit stakes are developed by people around the call such as @Raito's involvement with the public sector.

## Lix on the short term?

- @Raito: Being able to use it day-to-day and not have it break and RPC: for remote building/CI/CD/AFNix usage
- @Qyriad: Being able to use it day-to-day and not have it break and RPC: for us, it's about enabling incremental evaluation usecases.
- @Jade: I pretty much used Lix exclusively at work these days, at work, we want operational visibility. Being able to write things in Rust instead of C++ would make it more easier for us to jump in and help. Having tracing of any sort, even coarse-grained tracing of "this pull request regresses eval tiem by 5 % — maybe you should not do that" kind of things. Being able to understand the impact of Lix performance is really important. The other thing related to that is I'd like to improve certain areas of Lix performance that are often a problem, especially store performance. Copying files into the store wastes over 30s/developer/day, probably more than that.
- @piegames: Language improvements iteratively and the tools. We are stuck with Nix and Nixpkgs, so might as well make it good. Good tools for distributed builds, good CLI, and evaluation APIs — remote evaluators. Bytecode interpreter, some JIT maybe, memoization, all that stuff. There's a lot of things to do. In the long term, component rewrite of everything we have in Rust, maybe something taken from Snix. With a language with a proper type system and more than an iteration of the Nix language. At this point, we are technologically stuck. What are options to do a next-generation Nix? Looking at it, there's no chance of getting such a thing out of the ground because it would die an immediate death of second system syndrome. You'd redo everything and not get anything done. I'd to build the platform where other people can modularize and do their own things: example with Flakes, I want to make it possible for people to build their own dependency management things. Same for the language, I'd like to standardize a semantic where people can build their own language and lower it to the bytecode and we can have piecewise evolution with an ecosystem that interacts with each other.
- @horrors: RPC. The store protocol we are using today is absolutely bad. It also binds us to CppNix decisions that were frozen in 2.18. Once we have RPC, many possibilities are opened. Evaluation RPC as Qyriad wanted is also much more feasible.
- @piegames: I'm not a strong user, I have deployed it on my personal system and so on. I'm not a power user of Lix. I'm not interested into getting caught into supporting a too-large userbase.
- @jade: One of the problems I had with Lix: we had this problem where querying substituters got regressed by a large percentage and it was impossible to run the 2.92 series; it was also really hard to measure the regression. I think there's a large opportunity if we are able to run more devbuilds of Lix at work than it will be possible assuming if we had some tracing of some kind. Then, we can turn that into signals of regression.
- @raito: Would like to bring the topic of opt-in voluntary telemetry for Lix in 6 months or more.
- @jade: 100% agreed and I'd like to suggest we look at OpenTelemetry traces. Especially because we don't need to specifically send them into a Lix blessed central place.
- @kate: 2 things are really important for the future of Lix: making the project sustainable — including the fact that the store is a problem for everyone — (background on the known

inefficiences for the store: cache.nixos.org issues, etc.) *and* we are a very small team, we have very little ability to provide user support. One thing we need to work on sustainability is: how do we get more people to be able to take care of some of these roles that are difficult for such a small team? How do we get people who wants to profit off Lix give back? The second item I want to take the Lix monolith into multiple less-coupled pieces as much as possible. Take NixOS as an end user or Floral as an end user, I don't think there should be anything special there's a bunch of modules provided for NixOS. I don't think we should not have `nixos-rebuild` or `darwin-rebuild` where I would like to have some composability scheme where I want to do `nix $os rebuild` — I would like a strong enough plugin architecture where I want to build an hypervisor + VMs infrastructure concentrically without too much coupling. The whole ecosystem can work such in a way that `nixos-rebuild` is not THAT special cased in the long run. I'd like to enable lang versioning to happen easily by having this Nix-minimal fragment that we can convert things to. I'd like also to break out even things like the Git fetcher to interact via RPC. I don't want to care about how you fetch things.

- @piegames: We should focus more on what are the points of contention. Various different ways of getting there. Especially but not limited to community stuff. Because those are the points where we are not sustainable.
- @kate: I'm glad to hear we all have the broad same technical vision here. How we are getting there? Approaches like @raito or @horrors (more @horrors than @raito let's be honest) have been pushing us towards robust RPC. We make sure we have agreement on what are the priorities, we also need to make sure we have our expectations set right on the different paces and speeds so that our individual efforts meshes well. In the past, we had misunderstandings not based on differences in technical vision, but not what everyone else in the community was doing towards that technical vision.
- @kate: I suggest we take the 20 minutes left to identify the points we need to align on inside the pad. Also the points that would allow us to make a plan forward.
- @piegames: In terms of points I see: team structures reflected by a state of a codebase and we inherited a codebase from CppNix which reflects their own team structures and way of doing things. We need to break up how we work to better distribute the load but it also can be done if the software architecture follow through. RPC is the top priority then Rust is the top second priority because I believe those are what would enable us to onboard new devs without requiring super heavy mentoring like we did before *and* split up the codebase in more modular pieces. The other thing I want to talk about is the community side: it's a bit in the open air on how our community is looking. There has been discussions like Matrix is frankly a liability as a protocol. This is where also a lot of the socialization happens, even if it's not a lot, it's not nothing. We need to have a discussion about we want to proceed with Matrix and moderation. If we close down the Matrix channels, can we have the socializing bits on Zulip, etc. ? These are the pipelines which will get us new contributors in the project. We should be investing some resources into that. The community management in my eyes has been mostly fairly hands-off and did not require a lot of efforts, apart from some very recent big conflict. I have spend more time dealing with Matrix issues and Draupnir being down and getting channels into Matrix *THAN* doing moderation stuff for the community. That's not sustainable. Something needs to change here.

- @kate: Let's keep this meeting about technical vision and let's have community vision calls as well.
- @horrors: The Blender community has exclusively moved to Zulip-like platforms, they do not have Matrix, they ditched IRC, they do not have Discord. It seems to be working really well for them. Zulip alone might not be a big of a problem. (NOTE(horrors): blender actually moved to matrix *after* we last looked, so. ugh. may have been rocketchat or whatever got eaten by matrix)
- @kate: Typical open source strategy would be to seek more volunteers but we are also having sustainability issues with volunteers. We wonder if we should have more boring people in our project who are able to work with the people who are willing to make computers work in a way that aligns with our technical agenda. We may be able to bring more hands that are going to work on this project because of a job / a money transaction rather than the architect who have profound opinions on how things work.
- @jade: I have been primarily interested into making things just work. I have had very limited emotional energy and this was one of the thing which made Lix hard to work on.
- @kate: Maybe we need a call to define what is the community and what it should focus on, what are the stakeholders, the community team.
- @raito: One problem as part of afnix as well:
  - Wants to make it possible for people to receive money to work on Lix
  - Make it possible for people to be paid to do things we want to have
  - Also including e.g. infrastructure or pay for infrastructure
  - e.g. working on Lixcon, would love to use this to community more of our agenda, especially to potential
  - Work on fundraising
- @qyriad: I'd like to have observability on what is getting built on my system, I want to remove a lot of barriers that makes Lix difficult work for new users. A lot of this is technically entangled which makes it difficult work on with these other big moving pieces like RPC.
- @rootile: our& shortterm vision: 1. kill functional1 with fire and spite; 2. get rust going bc we& won't touch cpp.
- @kate: Backward compatibility about CppNix and RPC, we need to decide whether we want to have backward compat daemon<->client. [@raito: lost notes because my brain crashed.]
- @jade: Most of the time, backward compat is needed by accidental situations, e.g. devshell.
- @piegames: graceful degradation / errors, my personal approach for langver is to let the old stuff rot and keep it for interop and at some point remove the old stuff

# Things we wanted to discuss but did not / Agenda for next time

- raito: getting closer to an actual roadmapping document we can communicate
- availability (work-time & response-time) and expectations of core and non-core members
- tracing (zulip thread)
- piegames: Urgently needed devx improvements like a merge queue, less flaky CI

- piegames: agree on and clarify code owner semantics, committers etc.
- piegames: Documentation, issue triaging, wiki etc.

Discussion notes

# 2026-03-11 DevX meeting

- Date: 2026-03-11
- Participants: piegames, Qyriad, horrors, rutile, raito (15 minutes late)
- Topic: Lix DevX

# Agenda

- On Doxygen
- Merge queue
- Other DevX issues folks are facing?

# Meeting notes / Discussion

## On Doxygen

Objective: document the RPC protocol APIs in Lix, esp. the async bits.

Takeaway from @piegames: Keep Doxygen for now, but don't rely on it for new documentation. Improving it somewhat would be possible but require someone to step up and own the effort.

**Actionable @Qyriad: Publish the API documentation on docs.lix.systems, reusing the nightly docs pipeline. [Send this information to AFNix]**

## Merge queue

State of things: autosubmit bot, only works for already rebased CLs. Wanted: merge queue, that automatically rebases applicable commits and merges them.

**Raito: can implement the most trivial change but only in 3rd week of April.**

## Usual annoyances that folks silently accepts that we should not

- Non-flakey CI @piegames
  - @pennae: functional tests should be unflaked by migrating them
  - @pennae: NixOS tests cannot be unflaked
    - Unless replaced entirely
    - All NixOS tests failures are related to the virtual Ethernet switch use and the setup sometimes fails and doesn't get detected or fails and reports success.
    - It seems that network operational readiness reporting is failing and we do not know why. networkd reports that an interface is operational but it does not say that a service is ready.
    - `wait_for_open_port` is the thing that blocks and never completes.

    - NixOS tests failures are highly correlated with CI loads.
    - The best solution is not to use multiple nodes.
    - Proposal 1: Turn every multi-nodes NixOS tests into a single VM NixOS test containing multiple containers (implementing the multiple nodes).
        - Implementation details: have systemd spawn the containers and write a pytest-based framework to run f2-style tests against the containers spawned inside the single VM.
- Justfile could get better @piegames
    - Justfile to build Hydra tasks
        - Wants to provide the Hydra test name
        - @raito: nix-build -A hydraJobs.tests
        - @piegames: I did not know this! This is why I want Justfile!
    - Justfile task to build the reference manual & other documentation
        - without doing nix-build -A lix.dev
    - @raito: I want the `-custom` split to disappear
        - @Qyriad: if it takes arguments, it cannot take multiple targets at once
        - `just setup && just build` vs `just setup build`
            - Qyriad: currently: `just setup && just build-custom manual` works
        - **@raito will own this and run a poll on this topic on the Zulip thread and implement the popular vote**
    - List all available build targets: `meson introspect build --targets | jq '[.[] | .name] | sort | unique | .[]' --raw-output`
- `just test` requires `just install`, even functional2 requires that!
    - Proposal: make `just` do a `just install` for `just test-functional2` *and* `just test`
        - Qyriad: we cannot make `meson test` automatically install, but we can make `just test` automatically install

## F2 needs

- @rutile: command wrapping
- @pennae: i need command wrapping as well, esp. for the PTY stuff
- @pennae: ability to asynchronously communicate with a running command for interactions

**Actionable @rutile (?): adding async(io) wrappers for command subprocesses (stdin/stdout/stderr)**

Discussion notes

# 2026-03-13 State of the Matrix

- Date: 2026-03-13
- Participants: piegames, Raito, hexchen, Qyriad,
- Topic: Community team discussion on Lix's Matrix server, space, and channels

## Conclusion / Action items

- ☑ Make a backup the current database
  - Owner: @afnix @raito
- ☐ Check the PL levels are what they should to allow removing Draupnir
  - Owner: @piegames
- ☐ Get rid of Draupnir
  - Owner: @piegames
- ☐ Nuke our server and deploy a Synapse
  - ☐ Writing a plan and a rollback plan incrementally
    - ☐ Owner: @hexchen @raito
  - ☐ Execute the plan in a distant future afternoon (post-LixCon)
    - ☐ Owner: @hexchen @raito
- ☐ Announce the consolidation plan on blog + Matrix announcements
  - Owner: @piegames
- ☐ Tombstone and redo the #space:lix.systems room to heal the split brain
  - ☐ Consolidate the rooms via tombstones
  - ☐ Upgrade all our rooms to v12 via tombstones
  - Owner: @hexchen @piegames
- ☐ Update our documentation
  - Owner: @piegames

## Meeting notes / Discussion

- State of the current Matrix issues
  - Draupnir is broken, homeserver is broken, also possible network split
- piegames: So, how much Matrix do we want
- Raito: architecture proposal
  - The core issue is the way we deploy Matrix
  - If we just do a boring synapse thing we should have far fewer issues
  - piegames:
    - issues with draupnir

        - room state issues
        - e.g. draupnir issues aren't necessarily homeserver problems
            - likewise with room state desyncs
        - Swapping homeserver is necessary but not sufficient
    - Matrix, unfortunately, is currently one of the main ways people get involved in the Lix project
- piegames:
    - Reduce number of rooms
    - Remove Draupnir and do manual moderation
        - hexchen: we lose importing bans from NixOS moderation
            - …But that ship kind of sailed anyway
        - Raito: are they even doing bans rn anyway
        - hexchen: don't think so
            - We don't have a lot of common ground with them at this point
    - Once we replace Draupnir, our server has zero state
        - The things that matter can be recovered via federation
    - Should we consolidate rooms?
        - Right now we have (besides moderation rooms):
            - Lix Dev
            - Lix Off Topic
            - Lix
            - Lix on main
            - #off-topic
            - Lix project infrastructure
            - Community
            - Announcements
            - functional2
            - Nix lang 2
        - Raito: kill Lix Dev, Lix on main, Community, Nix lang 2, Lix project infrastructure
        - piegames: Maybe keep Lix on main
            - It has better SNR than the Lix
            - Qyriad: I agree
            - raito: fine with it
    - hexchen: we need to make sure draupnir allow to elevate the other users to the right privilege level we want
    - hexchen: re: room v12
        - All members of the community team should be set as Founders

Discussion notes

# 2026-03-18 - Governance weekly

- Date: 2026-03-18
- Participants: raito, horrors, piegames, rootile, kate, Qyriad.
- Topic: Lix governance weekly

## Agenda

- Job channel
- Inactive core team members
- Retrospective on the 2.95 release situation
- Various status updates from @raito for AFNix
- LixCon 2026 status

## Conclusion / Action items

### Job channel

Lix will postpone exploring a job channel idea until past LixCon and will mesh this problem with a sponsorship policy.

### Inactive core team members

https://git.lix.systems/lix-project/lix-website/pulls/69 is approved and merged along

https://git.lix.systems/lix-project/lix-website/pulls/70.

- Lily Foster
- 9999years (Rebecca Turner)

are disboarded from the core team role in the Lix project. Thanks for all their help and support towards the Lix project.

@piegames will adjust the governance document to better match our current procedures.

The Lix core team will examine in a future meeting whether to maintain a "historical core team member" section.

### Retrospective release situation

Split into two topics:

- the current 2.95 crisis, @raito @kate @horrors are addressing it.

- how to avoid repeating this mistake? post-poned to future meetings as we would like to have Jade part of this.

## Status updates from @raito

- AFNix implemented the fundraising proposals in https://zulip.lix.systems/#narrow/channel/16-Governance/topic/Fundraising.20in.20the.20name.20of.20Lix.20for.20AFNix/with/8503 on three fronts: the Open Collective is set up by @delroth, Matrix announcement and social media post has been done by @raito. Next will be Open Collective link in manual and project README.
- Currently, the AFNix project representative of Lix is @raito, @raito would welcome someone else to decrease the CoI issues of being AFNix president as well.

## LixCon 2026

- @raito expects more core team talks for LixCon and will pester core team members in private.
- @raito needs graphical design for LixCon, @hexchen seems sick and someone needs to take over. @kate offered to work on this.
- DGNum proposed a meeting at M-1 for LixCon but Lix did not respond. @kate propose we set up a weekly (in 4 weeks) between DGNum and Lix for LixCon. @raito proposed a scheduling link to both sides.

## Future agenda

- Examining our posture on perfectionism vs incremental changes.
- Historical core team members section.

# Meeting notes / Discussion

Agenda:

- Job channel
  - pennae: We want to do it mostly for employees than employers.
  - pennae: We should maybe make it the other way around and let people post they're looking for jobs.
  - kate: it would prove difficult for companies to actually operate with this due to the whole bias.
  - kate: having employers post ads as long as they behave in our code of conduct.
  - let's shelve this topic as there might not be enough demands to justify the costs of doing it right, at least, after lixcon.
  - @piegames own the fact of starting a draft on sponsorship policy.
- Inactive core team members

    - Merge https://git.lix.systems/lix-project/lix-website/pulls/70 and https://git.lix.systems/lix-project/lix-website/pulls/69
    - Qyriad: what about the proposal for alumni/emeritus section?
    - Kate: thinks this would be good to be able to list historical core team members to permit core team members to use this for hiring decisions.
    - pennae: wouldn't archive.org satisfy this objective too?
    - Kate: a hiring manager would not bother with that.
    - No objection to merge these two PRs.
    - @piegames will adjust the governance document to better match our current procedures
- Retrospective release situation
    - QA is required for 2.95.1 & 2.94.1
    - releng for 2.95.1 is owned by @raito if no one else will do it
    - Feature / merge freeze policy for releases owned by @raito but after LixCon
    - Testing commonly used 3rd party programs (`direnv` / `devenv`) with new Lix releases
    - Static builds should go into Hydra for the next releases, not sure we should put them back into CI
        - expensive
        - we do not need them very often
    - AFNix: convert releng to Hydra and automatic things
- [Status update, Raito] LixCon talks from the core team
- [Status update, Raito] Fundraising for Lix project [AFNix]
- [Status update, Raito] Sponsorship of Mercury in LixCon
- [Status update, Raito] AFNix project representative [AFNix]
    - https://docs.afnix.fr/policies/afnix-membership.html
- [Status update, Raito] LixCon 2026
    - We need graphic design for LixCon
    - DGNum team has proposed a meeting
    - Scheduling a weekly between DGNum and Lix for LixCon

Discussion notes

# 2026-03-27 - LixCon 2026 weekly

- Date: 2026-03-27
- Participants: raito (Lix/DGNum/AFNix), sinavir (DGNum), soyouzpanda (DGNum)
- Topic: LixCon 2026 weekly

# Call for Proposals (CfP)

- Proposals received: 8-9
- Core team talks and transactional conference talks (intro/outro, etc.) not included in the Pretalx
- Some invited talks are pending, awaiting final approval
- No issues anticipated regarding the current lineup.

## Proposal review plan

- 1 proposal already accepted for logistical reasons (travel support)
- Review schedule:
  - Tonight: @raito reviews proposals
  - Tomorrow (or so): @kate reviews proposals
  - This weekend or Monday: converge on final proposals
- Goal: complete review quickly to submit acceptance or rejections to our current proposals

**Remarks**

- Overall proposals goes over the Friday talk budget but aligns with overall planning
- No conference rooms required for Saturday and Sunday

**Special attention** AFNix general assembly room

- Requires Internet setup for videoconference
- Capacity: ~10–15 people

# Postcards

- **Deadline for order** 2 weeks before event; faster option: 1 week (+€2)
- Kate is aligned with Monday deadline

## Content

- Decide back-of-card content:
  - Important information and emergency contacts
  - Room list / orientation tips

    - WiFi info (if a shorter deadline)

**Assembling method** Punch postcards without plastic coating

**Inspiration:** Good-to-Know PDF from UndoneCS 2026

# Website

- Let DGNum host this website: https://lixcon.dgnum.eu on their S3 (Garage) instead of AFNix except if AFNix redeploys it quickly
- **Announcement post** email + Mastodon (high priority) → owned by @raito
- **Content to finish**
    - Schedule → Pretalx widget
    - Venue info → owned by @soyouzpanda
        - Rooms
        - Internet
    - Travel information
    - Physical Code of Conduct (LixCon/AFNix) → owned by @raito
    - Cross check with https://www.undonecs.org/2026/ information to see if we are lacking something
    - Ideally: RSS feed
- Optional: DNS CNAME `https://con.lix.systems/2026/` → main site (low priority)

# Equipment rental

METRO refers to https://www.metro.fr/ here.

## Fridges and Drinks

- Drinks brought by @hexchen and our METRO provider
- Suggested setup: 1–2 fridges in hacking rooms
- Suggested: additional payment terminal purchase (SumUp Plus) as AFNix one might be used for other purpose
- Deposit may apply to drinks, @sinavir will check with @hexchen directly

## Snacks

- METRO card available
- A car can be lended by @sinavir
- Which snack to buy / who will buy them → Can Anaëlle handle it?

## Seating and tables

- Cozy sitting (beanbags, etc.) for friendliness in hacking rooms → owned by @sinavir
1. Source price
2. Determine quantity

3. Propose quote
4. Execute if approved

### C3VOC equipment

- Coordination for setup → @raito connects @sinavir with @hexchen
- Free parking at ENS: confirm dimensions and availability for the van

## Room planning

- Core team members to book tickets explicitly on Pretix for full visibility on DGNum → owned by @raito
- Expected variance: ±10–20 people
- Goals:
  - 3 public rooms
  - 1 non-public room
  - Release remaining rooms
- More than 4 continuous rooms over the weekend is non-trivial
- Final room list to be confirmed by @soyouzpanda and @sinavir
- @raito wants to be kept in the loop and this topic should be discussed further next weekly

## Network setup

- Backbone v2 functional (Ielo uplink), deployed by @sinavir
- Connect LixCon rooms to Internet: requires coordination with the school IT to switch VLAN on the Ethernet ports in the rooms, owned by @sinavir
- Room setup: 1 switch and 1 AP per room min
  - 4 APs for the amphitheater
- Multiple SSIDs (firewalled and non-firewalled, OWE)
  - Non-firewalled is not a priority
- Forensics logging: DHCP leases and MAC/IPv6 via NDP as per law requires
  - IPv4 is already done in the current network
  - IPv6 will be new

### Priorities

1. Secure VLANs in rooms
2. Secure hardware (APs, switches, cables)
   - Friday: 4 + 3 APs, 4 switches
   - Saturday: 6 APs, 3 switches
   - Sunday: 6 APs, 3 switches
3. Configure router with forensic logging (DHCP + IPv6)
4. Configure switches
5. Configure APs
6. Optional: non-firewalled SSID

lead by @sinavir, will delegate whatever he can to other folks

# Next agenda

- Room planning
- Network setup
- Commit to conference start/end information to communicate to our attendees
- Approve the final operational details to send to our attendees

Discussion notes

# 2026-04-03 - LixCon 2026 weekly

Participants: raito (Lix/AFNix/DGNum), Kate (Lix), soyouzpanda (DGNum), sinavir (DGNum) Topic: LixCon 2026 weekly 2/4

Agenda:

- Calls for Proposals
- Postcards
- Website
- Equipment rentals
- Distinguishing staff from attendees
- Network setup
- Room planning
- Accessibility
- Conference start/end times
- Final operational email to attendees
- Physical code of conduct

## Calls for Proposals (CfP)

- @raito: CfP wrapping up, schedule almost ready, will integrate into website.

## Postcards

- Kate: designs ready, awaiting approval.
- raito: aligned with Monday deadline.
- Stickers for attendee names:
  - soyouzpanda: can buy sticky papers and print names.
  - @sinavir: will handle small materials purchase.

## Website

- Kate: possible redesign to fit more into the Lix theme, low priority.
- @raito: schedule to include Pretalx widget; DGNum reviewing/polish ongoing.
  - Physical Code of Conduct is blocked on Lix team.
  - RSS feed pending on soyouzpanda.
  - Target website announcement around Monday.

## Equipment rentals

### Drinks

- Number of Fritz Cola: @sinavir to confirm with @hexchen.
- Deposit questions:
  - Kate: standard German bottle deposit applies; returning bottles reduces next purchase cost.
  - @sinavir: hexchen will collect bottles; can adjust order accounting.
  - @raito: money handling to be figured out later.

## Fridges

- Proposal by Mathieu; @sinavir to obtain a quote.

## Payment terminals

- AFNix: 3 terminals (2 @thubrecht, 1 @delroth).
- DGNum: 2 terminals already ordered and obtained.

Largely enough.

## Snacks

- @sinavir: plan to buy usual student bar items; leftovers can be redistributed.
- @raito: requests list of student bar items.
- @soyouzpanda: list is here.
- @raito: will create final list for snacks/coffee.

## Seating and Tables

- Discussion on this topic adjourned due to @sinavir leaving early.

## C3VOC Equipment

- @sinavir coordinating with @hexchen.
- Remote speaker: pre-recorded session + live interactive Q&A.
- @sinavir: confirm ENS parking max height for C3VOC van.

# Distinguishing staff from attendees

- soyouzpanda: staff badge holders need distinction; color-coded lanyards suggested.
- Kate: suggest to procure lanyards with text to identify staff.
- soyouzpanda: student association has 12 organizer armbands.
- @raito: both lanyards and armbands to be used.
- soyouzpanda: owns procurement of lanyards.

# Network setup

- @raito: school uplink not used due to IT human resource limits.
  - Consequence: no IPv6.

    - Nonetheless: wired Internet, APs, private VLAN available.
- @soyouzpanda: alternative solutions may exist (@sinavir).
- Alternative: routers connecting via VPN; MTU adjustments likely.
- Action items:
  1. Communicate limitations → @raito
  2. Invite attendees to rely on LTE/5G → @raito
  3. Prepare local Nix(OS) cache → @raito
  4. Prepare B plan → @sinavir

# Room planning

- @sinavir: all fine according to previous communications.
- Requirements (from prior meeting):
    - 3 public rooms including quiet room
    - 1 non-public room

# Accessibility

- Kate: per European accessibility act:
    - Audit all graphics (screen reader alternatives)
    - Provide paper documentation at 200% scale
    - Comply with subtitle/sign language requests
    - Publish a11y contact email and designate on-site contact
- Action items:
    - @soyouzpanda: audit website with RGAA checker, publish a11y contact
    - @raito: check with @hexchen regarding captioning/sign language; announce the on-site contact at the intro talk

# Conference start/end times

- Friday 8:00 staff / 9:00 public check-in → 10:00 first talk → 18:00 public end → 20:00 staff
- Saturday: 9:00 public check-in → 18:00 public end → 19:00 late (no entrance anymore) → 20:00 staff
- Sunday: same as Saturday

Plan for teardown: grab volunteers, no formal plan needed.

# Final operational email to attendees

Dependencies: website ready, conference schedule and start/end finalized

Checklist:

- Conference start/end times
- Travel info (public transport)

- Schedule of talks
- Lunch/dinner info
- Encourage interest-based group formation

# Physical code of conduct

- @raito to coordinate with Kate; implement quickly.

# Next agenda

- Seating and tables (beanbags)
- Snacks, coffee, and hacking snack plan
- Equipment rentals update (quotes, purchases)
- Accessibility updates
- Final drinks order (Fritz Cola count, etc.)

Discussion notes

# 2026-04-01 Governance weekly

- Date: 2026-04-01
- Participants: raito, horrors, Qyriad, kate

## Scheduling release retrospective

- In-real-life at LixCon if time permits
- Meeting otherwise
- Owner: Qyriad

## Roadmap focus meeting

- Produce a draft based on technical vision document / previous conversations
- Owner: Raito

## Status update on LixCon

- Having a list of people who are expert/relevant on certain topics
- Will status update on the next weekly

## Schedule a sync Lix/AFNix regarding governance/funding/etc. at LixCon

- Should be discussed with all of the core team, in text
- Owner: Raito

Discussion notes

# 2026-04-18 LixCon meetup

- Date: 2026-04-18
- Participants: raito, rootile, piegames, Jade, lunaphied, qyriad, horrors

## Agenda

- Code owners
- +2/-2 review semantics (somewhat adjacent to code owners)
- AI policy

## Code owners

- piegames: I want to see some consensus and have written it down; what code ownership means to us? which permissions go with that? how do we distribute +1/+2 rights?
- jade: at the technical level, code ownership just sets recommended reviewers for a tree and also sets who is required to approve a change for a tree
  - +2 does not override codeowners
- piegames: so we can give +2 while keeping code owners coverage or we have +2 rights and code owners who are +2 rights for something?
- jade: those two looks the same to me because if you give a way +2 candy and you also give code owners of particularly directory like candy, this doesn't give particular broad merge capabilities; if you give +2 and code owners to F2 maintainers, they can merge tests
- [description of the system]
- piegames: it looks like the current system is quite complicated, is it worth keeping this level of flexibility?
- raito: i still think that we do not have the current right configuration for code owners, esp. wrt to security
- jade: if people self merge, that's bad and should be audited
- horrors: but the releng process has shown prior self merges
- jade: but this is automatically generated
- horrors: yes but that doesn't exempt it from code review
- [could not catch the rest of the conversation]
- piegames: now i understand how it works but i would like to think about how should it work; i would like a more traditional style
- jade: what we have at work is we have gh owners and it setups so it send pings if certain files changes, these people are not required to approve to merge things; you get someone from some other team and goes stamp a change to your code and that other engineer merges it within 15 minutes, this is obviously a culture failing but i also think this does happen in nixpkgs, code owners are not blocking

- piegames: ok, i think then code owners are doing too many things at the same time; assuming code owners are blocking code owners, does that allow them to merge only code they own? should it?
- rootile: i.e. Should code-owners have scoped +2 permission on their ownership
  - jade note: (this is what happens when someone has "+2" on gerrit in general. we could eliminate +2 rights as a requirement altogether and just use codeowners. then have global approvers list?)
- Qyriad: if the answer to that question is no, either we can give +2 for that purpose or i would suggest having a model where say: if two different code owners of the same tree approve the change,
  - Raito: s/2/N where N is number of codeowners-trees changed
- piegames: What even are "committers" in this model?
- rootile: Keeping ownership dynamically up-to-date will require some maintenance and might lag behind
- Raito: what do you mean by "governance overhead"?
  - My vision is like: you're working on the subsystem a lot, and then you're absent for a while, right?
  - In a reasonable amount of time it'd be good to communicate that you're not going to be active and so you should update your OWNERS file
  - rootile: Who's responsible for poking people to check their OWNERS file?
  - Raito: could make an automated proposal after N inactivity
    - Yes it is bookkeeping but if the bookkeeping provides value then it's worth it
    - jade note: some trees don't move that fast and so idk if it should be per-tree but maybe across the entire repo
  - jade: see also: the psychological fear of, say, adding new owners to a tree
- horrors: any meaningful change should go through a short discussion
- Qyriad: any meaningful modification of the code owners should probably NOT go solely through Gerrit
- Raito: I think the initial vision behind, e.g., handing out these things easily, was to make onboarding people much more accessible (unlike, say, CppNix)
  - However we've been handing out access but that hasn't corresponded to people taking responsibility on these things
    - Qyriad: It doesn't help that people weren't clear on what codeowners meant
  - So now we have accumulated ownership but people may or may not really care
- Raito:
  - Re: docs on these things
- jade: agreement with those present that we can delegate ownership of things like tests to the maintainers of those areas. ie. functional2 owners can add more functional2 owners at a governance level.
- piegames wishlist:
  - List of all code owners -> where? global approvers?
    - OWNERS files in the Lix tree
      - -> jade note: backports, old branches? probably need an all-branches thing for global approvers at least
      - some form of merged summary?
  - List of all people with +2 rights

    - refs/meta/config in the Lix tree (requires some privileges to read)
  - Documentation of +2 and codeowners
  - Documentation of submit requirements
  - Automatically keep that documentation up to date
  - A definition of "committers" for the purpose of our governance document

### Action items

```
- [piegames] Write documentation
- [raito] go through the permissions and review them
- [raito] make `refs/meta/config` public
```

# +2/-2 review semantics

- raito: +1 is someone else should look at it, +2 is this can go, -1 is meh, needs more discussion, -2 is strong disagreement, might require escalation to core team vote
- Core question: is -2 disagreement with the change in its current form, or with the general idea of the change
- rootile: my interpretation is that -2 means "the concept is fundementally flawed and this should not be worked on again"
- jade: -2 is a blocker, the reviewer needs to re-review and aggree; -2 does not mean bad concept
- piegames: my interpretation is: -1 is a fixable issue, -2 is an unfixable/needs major rework issue
- piegames: From a contributor perspective, -2 might be a critical but easily fixable issue, or it might be a fundamental rejection of the change, and only comments will tell them apart. This is a bad experience
- jade: -2 is just the "request changes" thing on github
- qyriad: i want a distinction between desirable and undesirable
- rootile: Suggestion: Introduce -3 to distinguish those
  - Qyriad: when would we need -3?
    - rootile: say something was ported from CppNix that is just not something we want at all
      - Raito: we need to document that
- piegames: seems like blocking vs unlocking is kind of orthogonal
  - Raito: maybe we should make Unresolved comments blocking?
- Agreement on current state and change being necessary; though not on a solution
- needs re-discussion

### Action items

- No general agreement found, will have to rediscuss

☐ Rename abandon to close (maybe fuck with the translation file)

☐ Change the label on -2 and -1 to e.g. "... see comment"

☐ Allow committers to abandon/close CLs by other people (document that doing this is generally rude and should be avoided)

### Deferred

- Final semantics of -1/-2

## AI Policy

Strongly shortened version:

- Anthropic has been working on things that would probably be good for Lix, but they have been unable to contribute to Lix
  - e.g., a system to compile code in a fully trusted and sandboxed and enclaved way; supply chain security stuff
- Lix is an AFNix project. as-of rn, lix is inheriting the AI policy of AFNix
- if lix wants to deviate, lix needs to discuss it with AFNix
- I'm strongly against allowing any kind of generated code within changes because once we allow some amount of AI-generated code, people will be tempted to find out the limits.
  - People will submit slop if we accept anything at all.
  - The act of dealing with bad changes is a lot of emotional/review capacity work.
- How do people become an expert? Easy contributions *should not be done with AI*, should be done by new contributors.
- If we do the easy work with AI as SMEs, we starve the newcomers of opportunities to do things.
- Having to switch to another infra than AFNix would be a lot of work! Their board is not interested in changing the AI policy.
- All vulns should be hand tested before submission.
- If your thing is good enough, it's not possible to know it's AI generated
- We might want to recommend against trying to use AI to understand the codebase because it might lie.
- **Everyone present agrees with the AFNix policy.**

Discussion notes

# 2026-04-22 Governance weekly

- Date: 2026-04-22
- Participants: ???

## Meeting notes / Discussion

## Agenda

- Codeownership of f2/testlib (removal of more broad owners)
- Perfectionism vs. incremental changes
- Flakes eval/language restrictions
- Discuss moving the testlib bits outside of the monorepo to address Python dependencies issues

## Code ownership of F2/testlib

- rootile: Entire test folder is owned by everyone. It makes sense, everyfew can make tests, delete them, etc. This does includes the testlib directory. IMO, while it belongs to tests, it is its own codebase, it should not be changed or owned by everyone.
- raito: proposal, put OWNERS in testlib with disabled inheritance and put the ownership to F2's codeowners.
- rootile: I'd also like to remove helle from code ownership of F2 because they have not been responsive in the past, if they become active again, I'd add them back. The only codeowner of F2 is me then.
- raito: currently, there's "root code owners" / "default code owners", so you are not the only CO on F2 in practice
- kate: I'd suggest code owners for F2 pieces a couple of core team members e.g. Osiria, just there, so that there's review distribution besides rootile.
- jade: a general comment on things of the shape of testlib, this is a thing that is well trodden, the google stuff is in a testing subdirectory of a source directory, we do not have a good place for python sources, it might be reasonable put testing sources in a src directory as well. Folks mentioned they would like to use F2 for other code outside of Lix, perhaps, we could consider F2 as its own software outside of Lix.
- kate: let's consider using other repositories even if Raito is a monorepo maximalist for things that are not just for Lix.
- qyriad: there might be python packaging concerns
- jade: i am very against moving anything outside of the lix repo for now because you have to deal with lockfiles, nix-eval-jobs was moving inside because it was breaking it all the time; we could use copybara which is how i release snowydeer, to create an export of that part of the lix tree if that's something useful

- moved into its own topic: moving Python outside the monorepo

## Action items

- `set noparent; set rootile` and add folks who wants to help
- disboard helle from the F2 code ownership

## Perfectionism vs incremental changes

- raito: [insert later the contextualization] (editor's not: lolsob)
- jade: this was written in the context of trying to get the work of unpacking changes in and it did not feel like we were start incrementally merging under a flag
- kate: i feel like there's a bunch of topics that was merged into this loaded topic; what is our obligation and standards of what we were doing, what are expectations for experimental, code that is going to grow vs. code we know that is less likely to have bugs — i don't find the culture of code perfectionism does much to address the number of bugs there, i do think it's a complete separate topic from one of the thing we have issues with: communication, plans on how to get things merged, what is acceptable, general communication about it — there's a definitive issue i have seen a lot, it's that people have a lot of trauma, their own expectations of themselves, things they expect to burn them in a codebase ; someone make an issue, "i don't want to deal with this trauma issue again" and it creates a situation where this is not even about a technical argument anymore because it is going to cause something negative to me happening — in a summary: we have a problem of discussing how to communicate to each other, what to do about the fact that a lot of us are afraid to drive changes in the codebase
- piegames: i'd like to split up the perfectionism from the culture of wanting to do things right, i do think it's a very good thing, that i think lix prides itself with, this can be a thinner line at times; one of the values i wrote back for the first fork and i think lix adopted was that we are willing to go to the extra mile and to do things right and prefer solutions that are long term and sustainable than quick and easy hacks to get things working if that's possible
- jade: i totally agree with kate that the underlying problem is the fear more so than it is anything else, i'd really like to decrease the fear, we can improve our testing story, we can start having fuzzing, we can start having other ways to hunt down bugs before merging, there's verification methodology to decrease the sense of fear, it's a pretty large problem; other methodologies could include having ways to persist warnings, we could surface later
- raito: [oh no no one took note :D]
- kate: the thing i said i realize could be taken the way raito took it :D — there's a difference between doing perfectionism and having a culture of doing things well; perfectionism encapsules to me the standard of "i have a way i think this should be done and i won't tolerate deviating from the way i think this should be done", i think we have accomplished a lot compared to CppNix in how we are taking time to plan and execute them properly, i think this is a culture of doing things right and i think this is different from a culture of perfectionism; can we introduce bugs as part of experimental features as long as they are caught in user testing? these kind of things are implicit in the idea of perfectionism

- horrors: phrasing this entire thing in testing or the fear, it does touch on important points, but not all of them, regarding the macOS stuff that created this topic, there were bugs surfaced and due to communication disasters, it ended up being a minor incident; if we add features that people really obviously want and there's dangerous, they should come with grave warnings they are dangerous, we only have experimental features with various degrees of communicating dangers from none to "hey this is stable!"; experimental features alone is just not a good framing for the kind of rare-to-common bugs that such features may bring and we do need get better to communicate to our users too
- piegames: about the testing, in spirit of doing things that are long term sustainable, i'm really fan of formal verification, strong type system, things that are resilient/correct/secure by design, fault tolerant by design; as much as I appreciate the efforts like doing the fuzzing stuff, doing these things have an high opportunity cost on NOT writing new code in Rust, NOT writing new systems that can improve upon the old ones that doesn't have failure modes in the first place. About the experimental features, everything is tangled with everything and all failures are very tricky and catastrophic. The focus to me is rather on the modularization and kinda refactoring in Rust and being fearless over there, if something goes wrong, it won't be that bad.
- kate: first, I'd like to address what piegames brought with formal verification, for 10 years, I was the head of being computer division, making a very very high assurance product with a full budget, trying to make it happen and it's — in scale — formal verification is not used because it's in general intractable for something that is a side effect manager. It cannot be the sole method of architecturing code, throwing so much money did not get fractions of the Linux kernel running correctly. I don't think fuzzing is taking time away from formal methods, because formal methods are inherently very hard to apply.
- piegames: Having type systems or encoding grammars are also acts of formal verification in my eyes.
- kate: language constructs as a FV is also insufficient — making a sound language that is capable of having things like that typing depends on data, you start more and more into mathematics and formalism. I don't think getting to Rust is going to change dramatically this. Things that can find bugs in other methods can inform on the overall software engineering. This is kind-of like defense in depth for testing.
- kate: we should talk about ways on reducing conflicts we had, for example, with the archival CL, something that could have solved the whole thing is that if there was a communication: I'm going to do this and I will architect and can we figure out the whole way on how to solve the problem? That didn't seem to happen in a public space.
- jade: I did!
- kate: We did find a better solution by doing investigations on this topic (wrt filesystem slowness on macOS) and we were able to come up with a better plan for virtual filesystem management that both solves the NAR archival by engaging a group of people.
- jade: I completely agree with many of this, it would be incorrect to say this was not discussed before it got written ; the implementation we came up was suggested by horrors. Anyway, we should lean on how other projects improve their quality: we can do somethingASAN, we can fuzz more bugs to appear or to be found. We can decrease the scariness of things. This is what I want to work on in Lix in the next few months.
- kate: As one of the two Darwin maintainers, the filesystem issue was not ever brought up to me or Osiria in a way that this is the plan we are going to do and discussing it. We

should agree on a policy to develop expectations on developing of any piece of development.
- raito: re: Kate's thing about formal verification
  - Worked on translating Rust to Lean
  - There's a project called "Signal Shot" (?)
  - Formal verification is still worth investigating
  - Though overall agrees with Kate
  - Thinks the true problem is in the communication
- Raito: Re: Darwin maintainers were not made aware of the unpacking thing design
  - Doesn't think that Kate/Qyriad were not yet considered the true Darwin maintainers
  - Thinks what horrors tried to do was give incremental feedback as they found issues
  - Thinks it's not about fear of the codebase and more about discovering things as we go
  - When you open the Lix devshell it asks you to reach out
    - We don't do that that much these days

## Action items

- Raito: thinks that further chat about this can be asynchronous
  - Can create Zulip topics about e.g. fuzzing, reducing conflicts, developing expectations (reinforcing the policy we have on the devshell contribmsg)

## Flakes eval/language restrictions

- piegames: Flakes uses a lang subset, which is full of defects and is an enormous layer of violations; this has been shakey grounds because of the layer violations, my current plans with bytecode conflicts with that piece of code. I'd like to make some breaking changes: removing all of the language restrictions in flake.nix. I don't know enough about Flakes to fully grasp the consequences but I do know that it will break CppNix unless they also do something, this will require coordination.
- Qyriad: this is an incompatibility we would be introducing with CppNix users (rather than a Breaking Change™ for Lix users).
- jade: exactly the same thing. Actually, it's less bad than we would see from touching libfetchers, we have some existing incompatibility in libfetchers, this will probably break some Lix users. We have been wanting to do this for a quite while. also, there's holes in the current model which makes it possible to evaluate complicated thunks in CppNix by abusing dynattrs!

(continued in next meeting)

Discussion notes

# 2026-04-29 Governance weekly

- Date: 2026-04-29

## Conclusion / Action items

## Flakes eval/lang restrictions

- piegames: let's rehash things to see if we agree or not, there has been conversations in the zulip
- piegames: basically, the main unresolved question is: do we feel strongly about those checks in the first place? how do we feel about changing them? how do we feel about removing them? from what i understood: we see those checks as mostly a guideline we can change over time but we want to keep them. one proposal that stuck out how to modify them would be to lift all the parser restrictions and go to only having a "no-function-calls" eval restriction.
  - Raito: full agreement on "from what I understood"
    - Hates those checks
    - However would be a huge pain in the ass to deal with users that are confused by the discrepancy of CppNix vs Lix
      - Because users often send us traces from not realizing it' CppNix instead of Lix
      - Unsolved problem of making users more sure of this
      - Predicts we get a lot of reports of "ahh everything is broken" because they try to eval CppNix-incompatable flakes not realizing they're using CppNix
  - Lunaphied: why do we want to remove the checks anyway?
  - raito: because in the bytecode world, there's no AST anymore.
  - qyriad: it would make error messages way better.
  - Qyriad: Good point on getting reports not realizing they're using CppNix. I would like though to consider how many people are going to make use of non-trivial flake.nix entrypoints and especially users that are not rea
  - Raito: two kinds of users of Nix:
    - Users of things like flake-parts
    - Knowledgeable about Nix; understand the two "languages"
    - They might exploit intentional features of Lix
    - People already do this; e.g. Lix can have a devshell with systemd; CppNix cannot
    - Other kind of users will fall into these errors
  - horrors: the delta between "no-function-calls" and "no weird AST" is thin and CppNix will probably follow this

- piegames: if we are having an issue having breaking changes, i don't understand the difference between relaxing changes and removing them, it will cause the same kind of incompatibility either way, it's a question of magnitude in a way, we will always have leaky abstractions; imagine someone doing a toString interpolation, with Lix now, we have `${32}` with `coerce-integers`, those details will always leak. I think the entire concept of restricting evaluation of expressively strong language is fundamentally the wrong approach, unless Flakes manage to go to JSON/TOML or something, those will always happen.
- raito: in full agreement with horrors
  - Also agree with piegames
  - The magnitude will be very low
  - Most users won't run into this
- raito: thinks we can merge Qyriad's original CL (Q: yay!)
- raito: envisions two outcomes:
  - People that want to replace npins/whatever
  - Granted without function calls this is pretty restricted
  - Taking a step back: we want some interface that works quickly, and flakes try to make that happen
    - This is a failure
    - The general idea, of enumerating metadata without long eval times (+ IFD, etc) is an interesting and valuable idea
- Qyriad: regarding the magnitude of concerns, it is low enough we can handle it, if it's not, then, i think we can put efforts to ensure that users can discover if they're not using lix or not; yes this idea of Flakes staying fast to enumerate is a complete failure, when it works, it barely works, the restriction are so much — most reasonable usecases don't even bother and go for legacy packages. The only thing that you can enumerate in Flakes without guaranteeing relatively short eval are the inputs and inputs alone. Everything else can take as long as it wants. I think if we want specifically make something that is quick to eval, we want to decide what are actual things we want to make it quick eval and put it in a TOML format.
- Raito: behind the current idea; just wanted to add some nuance
- piegames: lol i dont care

### Summary / Action items

- Remove the AST-shape check
- forbid function calls (maxCallDepth=1)
- typecheck the evaluated result instead of checking the AST
- don't do anything about dyn-attrs (leave them as-is)
- notify the CppNix team

## Discuss moving the testlib bits outside of the monorepo to address Python dependencies issues

- rootile: folks want to use F2 outside of the Lix codebase, what is the best way to facilitate that? snix used F2 as well; in my opinion, we might want to move testlib out of the F2 folder and make it a Python package which then you can use inside of test/f2 and even publish it to PyPI so that people outside of Lix can use it too
- raito: someone could try to do an experiment of using f2 testlib outside the monorepo and show what it looks like and how it could be improved
  - For users of f2 you'd probably want both the Nix interpreter and the Python dependencies
  - Probably *wouldn't* publish it to Pypi
    - Since it depends on the Nix interpreter. Kinda weird
    - Then we have to maintain a publishing pipeline and stuff
  - If it's just a Nixlang entry point they can just put it in their build inputs and go
  - Wonders if the rest could be moved to async discussion
  - rootile: thinks that's fine

# Trailing slashes in fetchurl (re: snix)

Context: https://git.lix.systems/lix-project/lix/issues/1185

- Difference between CppNix and Lix
- raito: What should the behavior be?
- horrors: we should change it; the current behavior makes no sense
- raito: :+1:

# Quick check-in

- Raito: limited availablity
  - Flakes extraction is affected
    - Either need more help or move to next release cycle
  - Can still tackle merge queue
  - Also planning a LixCon retrospective

# Contributor permissions and Code Owners

One might wonder how permissions work on Lix and how one can help out with certain tasks. This is subject to ongoing governance work. This page is a summary of the current (pretty vibes-based) state of how it works.

Feel free to ask questions about this in the Lix development Matrix room.

## Trusted contributors

We give away this permission freely to everyone who has commits in Lix and most people who ask for it with a reason to need it. It confers the rights to:

- Triage issues on Lix project repos
- Edit the wiki
- Access login-only pads on https://pad.lix.systems

## Lix approvers

This is the Code-Review+2 permission on Gerrit.

We're working on setting up Code Owners on Gerrit so that changes to specific subsystems require review from a subsystem author or an override. We will likely give away approver access to most active contributors at that point. This permission is given out to responsible contributors to Lix who have been working on Lix for a bit and have shown good understanding of expectations in the project.

There will probably be a more specific process in the future. Currently it's by nomination by people on the Lix team.

## Lix team

This category is very much subject to governance work to clarify how it should work. People in this group share administrative infrastructure access with infra maintainers and the Lix community team.

You can ask for help with any infrastructure issues from the Lix team.

# Gerrit Code Owners

Gerrit has a concept of "Code Owners". The idea of this is that people responsible for subsystems are required to approve changes to code in their subsystem.

This is hierarchical in terms of specificity, that is to say, the emails of users in the nearest `OWNERS` file will be suggested first for review requests when clicking "Suggest Owners", before going up to more general reviewers and if all else fails, the global approvers.

If you work on an area in Lix and want to review changes to your area, feel free to add yourself to the `OWNERS` file for the relevant directory (or the rules for individual files in there).

As of July 2025, we are still working on crystallizing how the code owners plugin should interact with our workflow.