# 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!

---

Revision #3
Created 27 March 2025 14:08:59 by kfearsoff
Updated 29 March 2025 14:39:12 by kfearsoff