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.
- 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.

Revision #1
Created 29 March 2025 13:25:46 by kfearsoff
Updated 29 March 2025 14:37:12 by kfearsoff