Engineer

git hook

⚓ a script git runs when something happens

What it is

A git hook is a program that git itself starts at a fixed point in its own work: before a commit is recorded, once a commit message has been written, before a push leaves the machine, or when a server receives one. Each hook is a file named after its moment, such as `pre-commit`, `commit-msg` or `pre-push`, with no extension. By default git looks for them in `.git/hooks` inside the repository, and the `core.hooksPath` setting points it elsewhere. The file has to be executable, since git ignores one without the executable bit, but it can be written in any language the machine runs.

Client-side hooks run in a developer's own clone and fire on operations such as committing and merging. Server-side hooks run on the machine that receives pushes, such as a forge, and fire when pushed commits arrive.

What turns a hook from a notification into a guard is its exit status. The `pre-commit` hook takes no arguments, and a non-zero exit aborts the commit. The `commit-msg` hook receives the path of a file holding the proposed message and may reject the message or edit it. The `pre-push` hook gets the remote's name and location as arguments and one line per reference on standard input, and a non-zero exit stops the push before anything is sent. On the server, `pre-receive` runs once per push, reads the old object name, new object name and reference name for each reference, and if it exits non-zero, none of the references are updated. Hooks that run after the event, such as `post-commit`, are meant for notification and cannot change its outcome.

When `git init` creates a repository, it fills the hooks directory with example scripts whose names end in `.sample`. git runs one only after it has been renamed without the suffix.

Use the concept in a decision

Put “What is a git hook?” to work in a real decision. We can help relate the concept to the system you are building or buying. Tested, released and reversible, so changes reach customers on a Tuesday afternoon rather than late on a Friday night.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Automated GIS SaaS deployment with GitHub Actions — deployment, data and reporting, all off the hands

Automated GIS SaaS deployment with GitHub Actions Get in touch →

Why it matters

A hook turns a check that someone has to remember into one that happens on its own, at the cheapest moment. A formatter or a secret scanner that runs as the commit is made catches a problem while the change is still in its author’s head, rather than in review days later or in a CI run after the work has been shared.

Hooks are also how a rule becomes a property of a repository rather than of whoever happens to be committing. A convention in a contributing guide holds only as long as people read the guide; the same convention in a `commit-msg` hook applies to every commit.

The two groups differ in how much they can promise. The Pro Git book points out that cloning a repository does not copy its client-side hooks, and recommends enforcing policy on the server side. A server-side hook such as `pre-receive` applies to every push from everyone, whatever they have installed locally. The cost is time: a hook runs on every commit or every push, and a check that makes committing slow invites people to skip it.

What it looks like in practice

This site’s own repository installs its hooks through the pre-commit framework, configured in `.pre-commit-config.yaml` and installed by `task setup`, which runs `pre-commit install --install-hooks`. The file installs three git hooks. At the pre-commit stage it runs Prettier, markdownlint, stylelint, yamllint, codespell, Ruff’s lint and format checks, a gitleaks secret scan, an emoji check and a tone check. At the commit-msg stage it runs one script that holds messages to a conventional, emoji-free form. At the pre-push stage it runs the whole gate, `task check`, which

takes minutes rather than seconds, a price the comment on that hook accepts because a push is rare. The file's header says never to bypass the hooks with `--no-verify`, and that comment holds to it: a push that cannot wait is not ready.

A commit-message hook also holds four repositories to one history standard: a conventional prefix and scope, a subject under a fixed length, no emojis and no attribution trailers. The hook rejects anything that does not conform in every repository that carries work, so the standard belongs to the repository rather than to whoever commits.

A server-side hook for the company's own forge was rehearsed before any real push reached it. Such a hook executes in a bare repository with no working tree, under an environment the forge sets, reading pushed references from standard input, so running it by hand on a laptop proves little. The rehearsal creates a bare repository, installs the hook, clones, commits, pushes and asserts on both the accept and the reject path. It found two defects that were invisible in the file, both described below.

On one project, the commit hook and the quality gate turned out to run different checks, although a document said the hook ran the gate. A test now resolves both to the checks they actually invoke and fails when the sets diverge. The five that had run only in the gate were the ones reading the CV prose, so the code was guarded on every commit and the writing only when someone remembered.

A client-side hook does not need a framework at all. This one, saved as `.git/hooks/commit-msg` and made executable, rejects a commit whose subject does not start with a conventional type:

```
#!/bin/sh
# git passes the path of the proposed commit message as $1
pattern='^(feat|fix|docs|chore)(\([a-z0-9-]+\))?: .+'
if head -n 1 "$1" | grep -Eq "$pattern"; then
    exit 0
fi
echo "commit-msg: start the subject with a type, as in 'fix(nav): ...'" >&2
exit 1
```

Where it goes wrong

- **A fresh clone arrives without the hooks.** Each clone has to install them, which is the job `task setup` does here. A comment in this site's `Taskfile.yml` spells out the quiet version of the failure: without `pre-commit` on the machine, `task setup` installs nothing, every hook silently does not run, and the result looks exactly like a repository that is passing.
- `--no-verify` **walks straight past them.** `git commit --no-verify` bypasses the `pre-commit` and `commit-msg` hooks, `git push --no-verify` bypasses `pre-push`, and setting `core.hooksPath` to `/dev/null` turns off every hook at once. A rule that must hold belongs in a server-side hook as well.
- **git's environment follows the hook into other repositories.** git exports variables such as `GIT_DIR` so that git commands in a hook find the right repository, and githooks(5) advises clearing them when a hook works in another one. The forge rehearsal met exactly this: the gate checked out the pushed revision and ran the tests inside it, the tests inherited git's pointer to the bare repository, and nineteen of them went red on correct code.
- **Standard input carries the edge cases.** The rehearsal's other defect was a fallback that, with no reference to work with, substituted a placeholder git cannot resolve; because the value arrives on standard input, the failure surfaced as an unrelated error much further down. The page also notes that the forge will see forced updates and branch deletions, which the rehearsal does not construct. In githooks(5), a deleted reference reaches `pre-push` as `(delete)` with an all-zeroes object name.
- **A slow hook gets skipped.** Bringing the five prose checks into the commit hook meant making two of them fast enough for a pre-commit budget, and that page is plain that runtime grows with the content and that there is a point at which a slow hook gets bypassed.
- **The format passes and the meaning goes unchecked.** As the page on the four repositories puts it, the format is checked and the content is not, so a conforming subject line that describes nothing passes exactly as well as one that explains the change.

Further reading

- githooks documentation — every hook git supports, when it runs, what it receives and what its exit status decides.
- Pro Git: Git Hooks — client-side and server-side hooks, how to install one, and why a clone leaves them behind.
- Pro Git: An Example Git-Enforced Policy — a server-side `update` hook that enforces a message format and access rules, with client-side hooks that warn before a push.

- git config — the `core.hooksPath` setting, for moving hooks out of `.git/hooks` or switching them all off.
- git commit — the `--no-verify` option and the two hooks it bypasses.
- git push — the `pre-push` hook and the `--no-verify` option that skips it.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a git hook?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/git-hook/