Engineer

Git

the version control this site's history lives in

What it is

Git is a distributed version control system: a program that records the state of a set of files as a chain of snapshots, so that any earlier state can be looked at, compared with the present or brought back. The project writes the name as Git and the command as `git` ; both mean the same tool.

Each snapshot is a commit. A commit stores what every tracked file looked like at that moment, who made the change and when, a message saying why, and a pointer to the commit or commits that came before it. Git stores a file it has already seen only once: when a file is unchanged between two commits, the new snapshot points at the copy already on disk instead of writing it again, so a long history costs far less space than a full copy per commit would.

Everything inside is named by its content. Before git stores a file, a directory listing or a commit, it computes a checksum over the bytes, in the classic format a SHA-1 hash written as 40 hexadecimal characters, and that checksum becomes the object’s name. Change a single byte and the name changes with it, along with the name of every commit built on top. That is how git notices corruption, and it is why a commit identifier still points at exactly the same code years later.

Distributed means every copy is complete. Cloning a repository brings down the entire history, not just the latest files, so reading the log, comparing two versions or making a commit all happen on the local disk with no server involved. Sharing work is a separate, deliberate step: you push your commits to another copy, or fetch commits from one and merge them. Any copy can act as the server; in most teams one agreed copy, often on a hosted service, plays that part by convention rather than by design.

A day's work moves through three places. The working tree holds the files you edit. The staging area, also called the index, holds what the next commit will contain, which lets you commit part of your changes and keep the rest for later. The git directory, `.git` at the top of the project, holds the object database and every reference. A branch is nothing more than a movable name pointing at one commit, which is why creating one is instant.

Git was written in 2005 by Linus Torvalds and the Linux kernel community, after the free-of-charge status of BitKeeper, the proprietary tool they had been using, was revoked. The goals they set were speed, a simple design, strong support for thousands of parallel branches, full distribution, and the capacity to handle a project the size of the kernel.

Use the concept in a decision

Put "What is git?" 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

Version control answers the questions that arrive as soon as more than one person, or one person over more than a week, touches the same files: what changed, when, by whom and why. Kept by hand, the answers live in memory, in file names like `final-v3` and in email threads. With git they live beside the code, attached to each change, where anyone can search them.

The distributed design changes how a team works day to day. Every clone holds the whole history, so nobody waits on a central server to read the log or to commit, and a server outage leaves everyone able to keep working. Branches are cheap, so every feature, fix or experiment can get one, be reviewed on its own and be merged when it is ready, while the main line stays fit to ship.

The same design makes git conservative with data. Nearly every operation only adds to the object database, so committed work is hard to lose: a branch deleted by mistake, a rebase that went sideways or a reset to the wrong commit can

usually be undone. The safety net is the reflog, a local record of where each branch pointed and when, kept by default for 90 days for commits that are still reachable and 30 days for those that are not.

Git is also the common ground under most of the other tools in a delivery chain. Forges, review tools, build systems, deploy scripts and static site generators all read from it, which turns the history into a source of facts well beyond the code: the day a page last changed, the commit a release was built from, the change a bug arrived with.

What it looks like in practice

This company's own code shows three sides of that.

The source used to live on a third-party hosting service. It now sits on a git server the company runs on its own hardware: a single binary supervised by the operating system, picked for being SSH-first with no administrative web interface. Clones and pushes travel over SSH to port 23231, which listens on loopback alone, so a remote outside the host reaches it through a jump host; anonymous and keyless sessions are both refused, and every declared repository is marked private. The page is plain about the price: running your own forge means owning authentication, access control, storage and backups as well.

Full copies of history on several machines are easy to mistake for a backup. The company's backup and restore-test implementation was designed to extend recovery beyond a provider snapshot. Its restic scripts and systemd units support a database dump where configured, encrypted deduplicated snapshots to storage at another provider over SFTP, retention pruning and integrity checks. A separate monthly workflow restores a known file to a scratch location and compares its bytes. That test must actually run before it proves a recovery position: the implementation record dated 2026-10-27 says the timers were disabled without a configured repository and records no completed live restore. No number of clones on laptops supplies that missing evidence.

The history also feeds the site itself. The site's credits page records that the W3C Feed Validator warns that most entries in the feeds share a modification date, and that the warning is correct: the dates come from git history, and most of those pages really were written in a handful of commits. The site would rather show the warning than make a date up. The history says what happened, including that much of it happened at once.

One command reads such a fact back. This prints the short identifier, the commit date in ISO 8601 form and the subject of the last commit that touched one file:

```
git log -1 --format='%h %cI %s' -- README.md
```

Where it goes wrong

- **A secret committed once stays in history.** Deleting the file in the next commit removes it from the latest snapshot and leaves it in the one before, which every clone already carries. Treat the credential as exposed and rotate it; rewriting history afterwards narrows the spread, but copies made in between keep the old commits.
- **A force push replaces someone else's work.** `git push --force` switches off the check that the remote branch is an ancestor of what you push, so commits a colleague pushed in the meantime drop off the branch. `--force-with-lease` updates the remote only while it still points where you last saw it.
- **Clones get treated as the backup.** A laptop copy lags behind, holds only the branches its owner happened to fetch, and has never been restored from on purpose. Issues, access rules and the rest of a forge's own data sit outside the repository altogether.
- **A large binary file bloats every clone for good.** Git keeps each version of each file and every clone downloads all of them, so a video committed by accident and deleted the next day still travels to every machine that clones the project afterwards.
- **Work kept in one place disappears with that place.** The reflog that rescues a bad reset lives on one machine and expires. Commits that were never pushed exist nowhere else, and a branch left alone for months drifts so far that merging it back becomes a project of its own.

Further reading

- Git documentation — the reference manual, the book and the videos, from the project itself.
- Pro Git — the whole book online, from a first commit to the object database underneath.
- What is Git? — snapshots, local operations, checksums and the three states in one chapter.
- A Short History of Git — why the kernel community wrote its own tool in 2005 and what it set out to do.
- git-reflog — how to find where a branch pointed before the mistake, and how long that record lasts.

- git-push — the force options, and what each one protects.

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 git?” 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/