**Git** ([/ɡɪt/](https://en.wikipedia.org/wiki/Help:IPA/English))[[7]](https://en.wikipedia.org/wiki/Git#cite_note-:0-7) is software for tracking changes in any set of [files](https://en.wikipedia.org/wiki/Computer_file), usually used for coordinating work among [programmers](https://en.wikipedia.org/wiki/Programmer) collaboratively developing [source code](https://en.wikipedia.org/wiki/Source_code) during [software development](https://en.wikipedia.org/wiki/Software_development). Its goals include speed, [data integrity](https://en.wikipedia.org/wiki/Data_integrity), and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).[[8]](https://en.wikipedia.org/wiki/Git#cite_note-kernel_SCM_saga-8)[[9]](https://en.wikipedia.org/wiki/Git#cite_note-integrity_goals-9)[[10]](https://en.wikipedia.org/wiki/Git#cite_note-linusGoogleTalk-10)

Git was created by [Linus Torvalds](https://en.wikipedia.org/wiki/Linus_Torvalds) in 2005 for development of the [Linux kernel](https://en.wikipedia.org/wiki/Linux_kernel), with other kernel developers contributing to its initial development.[[11]](https://en.wikipedia.org/wiki/Git#cite_note-pro-git-1.2-11) Since 2005, Junio Hamano has been the core maintainer. As with most other [distributed version control](https://en.wikipedia.org/wiki/Distributed_version_control) systems, and unlike most [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server) systems, every Git [directory](https://en.wikipedia.org/wiki/Directory_(computing)) on every [computer](https://en.wikipedia.org/wiki/Node_(networking)) is a full-fledged [repository](https://en.wikipedia.org/wiki/Repository_(version_control)) with complete history and full version-tracking abilities, independent of network access or a central server.[[12]](https://en.wikipedia.org/wiki/Git#cite_note-12) Git is [free and open-source software](https://en.wikipedia.org/wiki/Free_and_open-source_software) distributed under [GNU General Public License Version 2](https://en.wikipedia.org/wiki/GNU_General_Public_License#Version_2).

Git's design is a synthesis of Torvalds's experience with Linux in maintaining a large distributed development project, along with his intimate knowledge of file-system performance gained from the same project and the urgent need to produce a working system in short order. These influences led to the following implementation choices:[[40]](https://en.wikipedia.org/wiki/Git#cite_note-40)

**Strong support for non-linear development**

Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. In Git, a core assumption is that a change will be merged more often than it is written, as it is passed around to various reviewers. In Git, branches are very lightweight: a branch is only a reference to one commit. With its parental commits, the full branch structure can be constructed.[[*improper synthesis?*](https://en.wikipedia.org/wiki/Wikipedia:No_original_research#Synthesis_of_published_material)]

**Distributed development**

Like [Darcs](https://en.wikipedia.org/wiki/Darcs), [BitKeeper](https://en.wikipedia.org/wiki/BitKeeper), [Mercurial](https://en.wikipedia.org/wiki/Mercurial), [Bazaar](https://en.wikipedia.org/wiki/Bazaar_(software)), and [Monotone](https://en.wikipedia.org/wiki/Monotone_(software)), Git gives each developer a local copy of the full development history, and changes are copied from one such repository to another. These changes are imported as added development branches and can be merged in the same way as a locally developed branch.[[41]](https://en.wikipedia.org/wiki/Git#cite_note-41)

**Compatibility with existent systems and protocols**

Repositories can be published via [Hypertext Transfer Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) (HTTP), [File Transfer Protocol](https://en.wikipedia.org/wiki/File_Transfer_Protocol) (FTP), or a Git protocol over either a plain socket or [Secure Shell](https://en.wikipedia.org/wiki/Secure_Shell) (ssh). Git also has a CVS server emulation, which enables the use of existent CVS clients and IDE plugins to access Git repositories. [Subversion](https://en.wikipedia.org/wiki/Apache_Subversion) repositories can be used directly with git-svn.[[42]](https://en.wikipedia.org/wiki/Git#cite_note-42)

**Efficient handling of large projects**

Torvalds has described Git as being very fast and scalable,[[43]](https://en.wikipedia.org/wiki/Git#cite_note-43) and performance tests done by Mozilla[[44]](https://en.wikipedia.org/wiki/Git#cite_note-44) showed that it was an [order of magnitude](https://en.wikipedia.org/wiki/Order_of_magnitude) faster than some version-control systems; fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server.[[45]](https://en.wikipedia.org/wiki/Git#cite_note-45)

**Cryptographic authentication of history**

The Git history is stored in such a way that the ID of a particular version (a *commit* in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed. The structure is similar to a [Merkle tree](https://en.wikipedia.org/wiki/Merkle_tree), but with added data at the nodes and leaves.[[46]](https://en.wikipedia.org/wiki/Git#cite_note-46) ([Mercurial](https://en.wikipedia.org/wiki/Mercurial) and [Monotone](https://en.wikipedia.org/wiki/Monotone_(software)) also have this property.)

**Toolkit-based design**

Git was designed as a set of programs written in [C](https://en.wikipedia.org/wiki/C_(programming_language)) and several shell scripts that provide wrappers around those programs.[[47]](https://en.wikipedia.org/wiki/Git#cite_note-47) Although most of those scripts have since been rewritten in C for speed and portability, the design remains, and it is easy to chain the components together.[[48]](https://en.wikipedia.org/wiki/Git#cite_note-48)

**Pluggable merge strategies**

As part of its toolkit design, Git has a well-defined model of an incomplete merge, and it has multiple algorithms for completing it, culminating in telling the user that it is unable to complete the merge automatically and that manual editing is needed.[[49]](https://en.wikipedia.org/wiki/Git#cite_note-49)

[**Garbage**](https://en.wikipedia.org/wiki/Garbage_(computer_science))**accumulates until collected**

Aborting operations or backing out changes will leave useless dangling objects in the database. These are generally a small fraction of the continuously growing history of wanted objects. Git will automatically perform [garbage collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) when enough loose objects have been created in the repository. Garbage collection can be called explicitly using git gc.[[50]](https://en.wikipedia.org/wiki/Git#cite_note-50)

**Periodic explicit object packing**

Git stores each newly created object as a separate file. Although individually compressed, this takes a great deal of space and is inefficient. This is solved by the use of *packs* that store a large number of objects [delta-compressed](https://en.wikipedia.org/wiki/Delta_encoding) among themselves in one file (or network byte stream) called a *packfile*. Packs are compressed using the [heuristic](https://en.wikipedia.org/wiki/Heuristic_(computer_science)) that files with the same name are probably similar, without depending on this for correctness. A corresponding index file is created for each packfile, telling the offset of each object in the packfile. Newly created objects (with newly added history) are still stored as single objects, and periodic repacking is needed to maintain space efficiency. The process of packing the repository can be very computationally costly. By allowing objects to exist in the repository in a loose but quickly generated format, Git allows the costly pack operation to be deferred until later, when time matters less, e.g., the end of a workday. Git does periodic repacking automatically, but manual repacking is also possible with the git gc command. For data integrity, both the packfile and its index have an [SHA-1](https://en.wikipedia.org/wiki/SHA-1) checksum inside, and the file name of the packfile also contains an SHA-1 checksum. To check the integrity of a repository, run the git fsck command.[[51]](https://en.wikipedia.org/wiki/Git#cite_note-Git_-_Packfiles-51)

Another property of Git is that it snapshots directory trees of files. The earliest systems for tracking versions of source code, [Source Code Control System](https://en.wikipedia.org/wiki/Source_Code_Control_System) (SCCS) and [Revision Control System](https://en.wikipedia.org/wiki/Revision_Control_System) (RCS), worked on individual files and emphasized the space savings to be gained from [interleaved deltas](https://en.wikipedia.org/wiki/Interleaved_deltas) (SCCS) or [delta encoding](https://en.wikipedia.org/wiki/Delta_encoding) (RCS) the (mostly similar) versions. Later revision-control systems maintained this notion of a file having an identity across multiple revisions of a project. However, Torvalds rejected this concept.[[52]](https://en.wikipedia.org/wiki/Git#cite_note-52) Consequently, Git does not explicitly record file revision relationships at any level below the source-code tree.

These implicit revision relationships have some significant consequences:

* It is slightly more costly to examine the change history of one file than the whole project.[[53]](https://en.wikipedia.org/wiki/Git#cite_note-53) To obtain a history of changes affecting a given file, Git must walk the global history and then determine whether each change modified that file. This method of examining history does, however, let Git produce with equal efficiency a single history showing the changes to an arbitrary set of files. For example, a subdirectory of the source tree plus an associated global header file is a very common case.
* Renames are handled implicitly rather than explicitly. A common complaint with [CVS](https://en.wikipedia.org/wiki/Concurrent_Versions_System) is that it uses the name of a file to identify its revision history, so moving or renaming a file is not possible without either interrupting its history or renaming the history and thereby making the history inaccurate. Most post-CVS revision-control systems solve this by giving a file a unique long-lived name (analogous to an [inode](https://en.wikipedia.org/wiki/Inode) number) that survives renaming. Git does not record such an identifier, and this is claimed as an advantage.[[54]](https://en.wikipedia.org/wiki/Git#cite_note-54)[[55]](https://en.wikipedia.org/wiki/Git#cite_note-55) [Source code](https://en.wikipedia.org/wiki/Source_code) files are sometimes split or merged, or simply renamed,[[56]](https://en.wikipedia.org/wiki/Git#cite_note-56) and recording this as a simple rename would freeze an inaccurate description of what happened in the (immutable) history. Git addresses the issue by detecting renames while browsing the history of snapshots rather than recording it when making the snapshot.[[57]](https://en.wikipedia.org/wiki/Git#cite_note-57) (Briefly, given a file in revision *N*, a file of the same name in revision *N* − 1 is its default ancestor. However, when there is no like-named file in revision *N* − 1, Git searches for a file that existed only in revision *N* − 1 and is very similar to the new file.) However, it does require more [CPU](https://en.wikipedia.org/wiki/Central_processing_unit)-intensive work every time the history is reviewed, and several options to adjust the heuristics are available. This mechanism does not always work; sometimes a file that is renamed with changes in the same commit is read as a deletion of the old file and the creation of a new file. Developers can work around this limitation by committing the rename and the changes separately.

Git implements several merging strategies; a non-default strategy can be selected at merge time:[[58]](https://en.wikipedia.org/wiki/Git#cite_note-58)

* *resolve*: the traditional [three-way merge](https://en.wikipedia.org/wiki/Three-way_merge) algorithm.
* *recursive*: This is the default when pulling or merging one branch, and is a variant of the three-way merge algorithm.

When there are more than one common ancestors that can be used for a three-way merge, it creates a merged tree of the common ancestors and uses that as the reference tree for the three-way merge. This has been reported to result in fewer merge conflicts without causing mis-merges by tests done on prior merge commits taken from Linux 2.6 kernel development history. Also, this can detect and handle merges involving renames.

— Linus Torvalds[[59]](https://en.wikipedia.org/wiki/Git#cite_note-59)

* *octopus*: This is the default when merging more than two heads.