




Git ([/git/](#)^[7]) is a [distributed version-control](#) system for tracking changes in [source code](#) during [software development](#).^[8] It is designed for coordinating work among [programmers](#), but it can be used to track changes in any set of [files](#). Its goals include speed,^[9] [data integrity](#),^[10] and support for distributed, non-linear workflows.^[11]

Git



```
$ git init
Initialized empty Git repository in /tmp/tmp.IMBYSY7R8Y/.git/
$ cat > README << 'EOF'
> Git is a distributed revision control system.
> EOF
$ git add README
$ git commit
[master (root-commit) e4dcc69] You can edit locally and push
to any remote.
 1 file changed, 1 insertion(+)
 create mode 100644 README
$ git remote add origin git@github.com:cdown/thats.git
$ git push -u origin master
```

A command-line session showing repository creation, addition of a file, and remote synchronization

Original author(s)	Linus Torvalds ^[1]
Developer(s)	Junio Hamano and others ^[2]
Initial release	7 April 2005
Stable release	2.24.0 / 4 November 2019 ^[3]
Repository	git.kernel.org/pub/scm/git/git.git/ 
Written in	C, Shell, Perl, Tcl, Python ^[4]
Operating system	POSIX: Linux, Windows, macOS
Available in	English
Type	Version control
License	GPLv2, ^[5] LGPLv2.1, ^[6] and others
Website	git-scm.com 

Git was created by [Linus Torvalds](#) in 2005 for development of the [Linux kernel](#), with other kernel developers contributing to its initial development.^[12] Its current maintainer since 2005 is [Junio Hamano](#). As with most other distributed version-control systems, and unlike most [client–server](#) systems, every Git [directory](#) on every [computer](#) is a full-fledged [repository](#) with complete history and full version-tracking abilities, independent of network access or a central server.^[13] Git is [free](#)

and open-source software distributed under the terms of the [GNU General Public License](#) version 2.

History

Git development began in April 2005, after many developers of the [Linux kernel](#) gave up access to [BitKeeper](#), a proprietary source-control management (SCM) system that they had formerly used to maintain the project.^[14] The copyright holder of BitKeeper, [Larry McVoy](#), had withdrawn free use of the product after claiming that [Andrew Tridgell](#) had created [SourcePuller](#) by [reverse engineering](#) the BitKeeper protocols.^[15] The same incident also spurred the creation of another version-control system, [Mercurial](#).

[Linus Torvalds](#) wanted a distributed system that he could use like BitKeeper, but none of the available free systems met his needs. Torvalds cited an example of a source-control management system needing 30 seconds to apply a patch and update all associated metadata, and noted that this would not scale to the needs of Linux kernel development, where synchronizing with fellow maintainers could require 250 such actions at once. For his design criteria, he specified that patching should take no more than three seconds,^[9] and added three more points:

- Take [Concurrent Versions System](#) (CVS) as an example of what *not* to do; if in doubt, make the exact opposite decision.^[11]
- Support a distributed, BitKeeper-like workflow.^[11]
- Include very strong safeguards against corruption, either accidental or malicious.^[10]

These criteria eliminated every then-extant version-control system, so immediately after the 2.6.12-rc2 Linux kernel development release, Torvalds set out to write his own.^[11]

The development of Git began on 3 April 2005.^[16] Torvalds announced the project on 6 April;^[17] it became [self-hosting](#) as of 7 April.^[16] The first merge of multiple branches took place on 18 April.^[18] Torvalds achieved his performance goals; on 29 April, the nascent Git was benchmarked recording patches to the Linux kernel tree at the rate of 6.7 patches per second.^[19] On 16 June Git managed the kernel 2.6.12 release.^[20]

Torvalds turned over [maintenance](#) on 26 July 2005 to [Junio Hamano](#), a major contributor to the project.^[21] Hamano was responsible for the 1.0 release on 21 December 2005 and remains the project's maintainer.^[22]

Naming



Torvalds sarcastically quipped about the name *git* (which means *unpleasant person* in [British English](#) slang): "I'm an egotistical bastard, and I name all my projects after myself. First '[Linux](#)', now '[git](#)'."^{[23][24]} The [man page](#) describes Git as "the stupid content tracker".^[25] The read-me file of the source code elaborates further:^[26]

The name "git" was given by Linus Torvalds when he wrote the very first version. He described the tool as "the stupid content tracker" and the name as (depending on your way):

- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
- stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
- "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
- "goddamn idiotic truckload of sh*t": when it breaks

Releases^{[27][28]}



List of Git releases:^[29]

Version	Original release date	Latest (patch) version	Release date (of patch)
0.99	2005-07-11	0.99.9n	2005-12-15
1.0	2005-12-21	1.0.13	2006-01-27
1.1	2006-01-08	1.1.6	2006-01-30
1.2	2006-02-12	1.2.6	2006-04-08
1.3	2006-04-18	1.3.3	2006-05-16
1.4	2006-06-10	1.4.4.5	2008-07-16
1.5	2007-02-14	1.5.6.6	2008-12-17
1.6	2008-08-17	1.6.6.3	2010-12-15
1.7	2010-02-13	1.7.12.4	2012-10-17
1.8	2012-10-21	1.8.5.6	2014-12-17
1.9	2014-02-14	1.9.5	2014-12-17
2.0	2014-05-28	2.0.5	2014-12-17
2.1	2014-08-16	2.1.4	2014-12-17
2.2	2014-11-26	2.2.3	2015-09-04
2.3	2015-02-05	2.3.10	2015-09-29
2.4	2015-04-30	2.4.12	2017-05-05
2.5	2015-07-27	2.5.6	2017-05-05
2.6	2015-09-28	2.6.7	2017-05-05
2.7	2015-10-04	2.7.6	2017-07-30
2.8	2016-03-28	2.8.6	2017-07-30
2.9	2016-06-13	2.9.5	2017-07-30
2.10	2016-09-02	2.10.5	2017-09-22
2.11	2016-11-29	2.11.4	2017-09-22
2.12	2017-02-24	2.12.5	2017-09-22
2.13	2017-05-10	2.13.7	2018-05-22
2.14	2017-08-04	2.14.5	2018-09-27
2.15	2017-10-30	2.15.3	2018-09-27
2.16	2018-01-17	2.16.5	2018-09-27
2.17	2018-04-02	2.17.2	2018-09-27
2.18	2018-06-21	2.18.1	2018-09-27

2.19	2018-09-10	2.19.2	2018-11-21
2.20	2018-12-09	2.20.1	2018-12-15
2.21	2019-02-24	2.21.0	2019-02-24
2.22	2019-06-07	2.22.1	2019-08-10
2.23	2019-08-16	2.23.0	2019-08-16
2.24	2019-11-04	2.24.0	2019-11-04
Legend: ■ Old version ■ Older version, still supported ■ Latest version ■ Latest preview version			

Design

Git's design was inspired by [BitKeeper](#) and [Monotone](#).^{[30][31]} Git was originally designed as a low-level version-control system engine, on top of which others could write front ends, such as [Cogito](#) or [StGIT](#).^[31] The core Git project has since become a complete version-control system that is usable directly.^[32] While strongly influenced by BitKeeper, Torvalds deliberately avoided conventional approaches, leading to a unique design.^[33]

Characteristics



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:

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.

Distributed development

Like [Darcs](#), [BitKeeper](#), [Mercurial](#), [Bazaar](#), and [Monotone](#), 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.

Compatibility with existent systems and protocols

Repositories can be published via [Hypertext Transfer Protocol](#) (HTTP), [File Transfer Protocol](#) (FTP), or a Git protocol over either a plain socket, or [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](#) repositories can be used directly with git-svn.

Efficient handling of large projects

Torvalds has described Git as being very fast and scalable,^[34] and performance tests done by Mozilla^[35] showed that it was an [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.^[36]

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](#), but with added data at the nodes and leaves.^[37] ([Mercurial](#) and [Monotone](#) also have this property.)

Toolkit-based design

Git was designed as a set of programs written in [C](#) and several shell scripts that provide wrappers around those programs.^[38] 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.^[39]

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.

Garbage 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](#) when enough loose objects have been created in the repository. Garbage collection can be called explicitly using `git gc --prune`.^[40]

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](#) among themselves in one file (or network byte stream) called a *packfile*. Packs are compressed using the [heuristic](#) that files with the same name are probably similar, but do not depend on it 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 work day. 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](#) 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.

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](#) (SCCS) and [Revision Control System](#) (RCS), worked on individual files and emphasized the space savings to be gained from [interleaved deltas](#) (SCCS) or [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.^[41] 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.^[42] 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](#) 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](#) number) that survives renaming. Git does not record such an identifier, and this is claimed as an advantage.^{[43][44]} [Source code](#) files are sometimes split or merged, or simply renamed,^[45] 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.^[46] (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](#)-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.^[47]

- *resolve*: the traditional [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 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^[48]

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

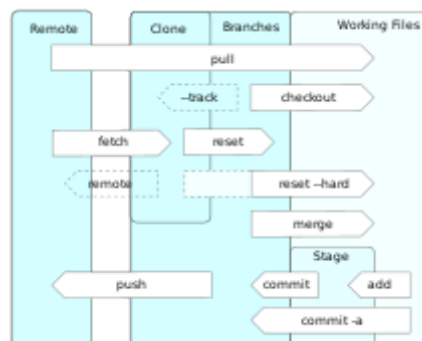
Data structures



Git's primitives are not inherently a [source-code management](#) system. Torvalds explains:^[49]

In many ways you can just see git as a filesystem – it's [content-addressable](#), and it has a notion of versioning, but I really designed it coming at the problem from the viewpoint of a *filesystem* person (hey, kernels is what I do), and I actually have absolutely *zero* interest in creating a traditional SCM system.

From this initial design approach, Git has developed the full set of features expected of a traditional SCM,^[32] with features mostly being created as needed, then refined and extended over time.



Some data flows and storage levels in the Git revision control system

Git has two [data structures](#): a mutable *index* (also called *stage* or *cache*) that caches information about the working directory and the next revision to be committed; and an immutable, append-only *object database*.

The index serves as connection point between the object database and the working tree.

The object database contains four types of objects:

- A *blob* ([binary large object](#)) is the content of a [file](#). Blobs have no proper file name, time stamps, or other metadata. (A blob's name internally is a hash of its content.)
- A *tree* object is the equivalent of a directory. It contains a list of file names, each with some type bits and a reference to a blob or tree object that is that file, symbolic link, or directory's contents. These objects are a snapshot of the source tree. (In whole, this comprises a [Merkle tree](#), meaning that only a single hash for the root tree is sufficient and actually used in commits to precisely pinpoint to the exact state of whole tree structures of any number of sub-directories and files.)
- A *commit* object links tree objects together into a history. It contains the name of a tree object (of the top-level source directory), a time stamp, a log message, and the names of zero or more parent commit objects.
- A *tag* object is a container that contains a reference to another object and can hold added meta-data related to another object. Most commonly, it is used to store a [digital signature](#) of a commit object corresponding to a particular release of the data being tracked by Git.

Each object is identified by a SHA-1 [hash](#) of its contents. Git computes the hash and uses this value for the object's name. The object is put into a directory matching the first two characters of its hash. The rest of the hash is used as the file name for that object.

Git stores each revision of a file as a unique blob. The relationships between the blobs can be found through examining the tree and commit objects. Newly added objects are stored in their entirety using [zlib](#) compression. This can consume a large amount of disk space quickly, so objects can be combined into *packs*, which use [delta compression](#) to save space, storing blobs as their changes relative to other blobs.

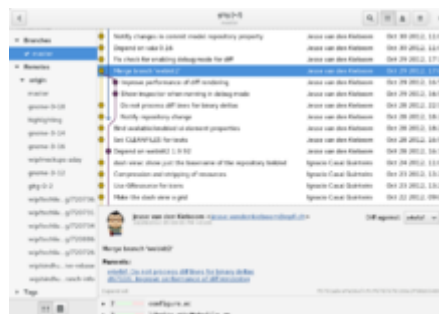
References



Every object in the Git database that is not referred to may be cleaned up by using a garbage collection command or automatically. An object may be referenced by another object or an explicit reference. Git knows different types of references. The commands to create, move, and delete references vary. "git show-ref" lists all references. Some types are:

- *heads*: refers to an object locally,
- *remotes*: refers to an object which exists in a remote repository,
- *stash*: refers to an object not yet committed,
- *meta*: e.g. a configuration in a bare repository, user rights; the refs/meta/config namespace was introduced retrospectively, gets used by [Gerrit](#),^[50]
- *tags*: see above.

Implementations



[gitg](#) is a graphical front-end using [GTK+](#)

Git is primarily developed on [Linux](#), although it also supports most major operating systems, including [BSD](#), [Solaris](#), [macOS](#), and [Windows](#).^[51]

The first Windows [port](#) of Git was primarily a Linux-emulation framework that hosts the Linux version. Installing Git under Windows creates a similarly named Program Files directory containing the [MinGW](#) port of the [GNU Compiler Collection](#), [Perl](#) 5, [msys2.0](#) (itself a fork of [Cygwin](#), a Unix-like emulation environment for Windows) and various other Windows ports or emulations of Linux utilities and libraries. Currently native Windows builds of Git are distributed as 32- and 64-bit installers.^[52]

The JGit implementation of Git is a pure [Java](#) software library, designed to be embedded in any Java application. JGit is used in the [Gerrit](#) code-review tool and in EGit, a Git client for the [Eclipse](#) IDE.^[53]

[go-git](#) is an [open-source](#) implementation of Git written in pure [Go](#).^[54] It is currently used for backing projects as a [SQL](#) interface for Git code repositories^[55] and providing [encryption](#) for

Git.^[56]

The Dulwich implementation of Git is a pure [Python](#) software component for Python 2.7, 3.4 and 3.5.^[57]

The libgit2 implementation of Git is an ANSI C software library with no other dependencies, which can be built on multiple platforms, including Windows, Linux, macOS, and BSD.^[58] It has bindings for many programming languages, including [Ruby](#), Python, and [Haskell](#).^{[59][60][61]}

JS-Git is a [JavaScript](#) implementation of a subset of Git.^[62]

Git GUIs

Git server

! This section needs additional citations for verification.

[Learn more](#)



Screenshot of Gitweb interface showing a commit [diff](#)

As Git is a distributed version-control system, it could be used as a server out of the box. It's shipped with built-in command `git daemon` which starts simple TCP server running on the GIT protocol.^[63] Dedicated Git HTTP servers help (amongst other features) by adding access control, displaying the contents of a Git repository via the web interfaces, and managing multiple repositories. Already existing Git repositories can be cloned and shared to be used by others as a centralized repo. It can also be accessed via remote shell just by having the Git software installed and allowing a user to log in.^[64] Git servers typically listen on [TCP port 9418](#).^[65]

Open source

- gitolite, scripts on top of git software to provide fine-grained access control.
- [Gerrit](#), a git server configurable to support code reviews and providing access via ssh, an integrated [Apache MINA](#) or OpenSSH, or an integrated [Jetty](#) web server. Gerrit provides integration for LDAP, Active Directory, OpenID, OAuth, Kerberos/GSSAPI, X509 https client

certificates. With Gerrit 3.0 all configurations will be stored as git repositories, no database required to run. Gerrit has a pull-request feature implemented in its core but lacks a GUI for it.

- [Phabricator](#), a spin-off from Facebook. As Facebook primarily uses [Mercurial](#), the git support is not as prominent.^[66]
- [Trac](#), supporting git, [Mercurial](#), and [Subversion](#) with a [modified BSD license](#).
- [Rhodecode](#) Community Edition (CE), supporting git, [Mercurial](#) and [Subversion](#) with a [AGPLv3 license](#).
- [Kallithea](#), supporting both git and [Mercurial](#), developed in [Python](#) with [GPL license](#).
- There are several other FLOSS full solutions for self-hosting, including Gogs and [Gitea](#), a fork of Gogs, both developed in [Go language](#) with [MIT license](#),

As service



Best known are probably [GitHub](#) and [Bitbucket](#) offerings, but many others are available, like [GitLab](#), [Gitea](#), [GerritForge](#), etc.

Adoption

The [Eclipse Foundation](#) reported in its annual community survey that as of May 2014, Git is now the most widely used source-code management tool, with 42.9% of professional software developers reporting that they use Git as their primary source-control system^[67] compared with 36.3% in 2013, 32% in 2012; or for Git responses excluding use of [GitHub](#): 33.3% in 2014, 30.3% in 2013, 27.6% in 2012 and 12.8% in 2011.^[68] Open-source directory [Black Duck Open Hub](#) reports a similar uptake among open-source projects.^[69]

[Stack Overflow](#) has included [Version control](#) in their annual developer survey^[70] in 2015 (16,694 responses),^[71] 2017 (30,730 responses)^[72] and 2018 (74,298 responses).^[73] Git was the overwhelming favorite of responding developers in these surveys, reporting as high as 87.2% in 2018. Version control systems used by responding developers: Git (69.3% in 2015, 69.2% in 2017 and 87.2% in 2018), [Subversion](#) (36.9% in 2015, 9.1% in 2017 and 16.1% in 2018), Microsoft [Team Foundation Server](#) (12.2% in 2015, 7.3% in 2017 and 10.9% in 2018), [Mercurial](#) (7.9% in 2015, 1.9% in 2017 and 3.6% in 2018), [CVS](#) (4.2% in 2015; not listed in 2017 or 2018), [Perforce](#) (3.3% in 2015; not listed in 2017 or 2018), [Microsoft Visual SourceSafe](#) (0.6% in 2017; not listed in 2015 or 2018), [Rational ClearCase](#) (0.4% in 2017; not listed in 2015 or 2018), Zip file backups (not listed in 2015; 2.0% in 2017 and 7.9% in 2018), Copying and pasting files to network shares (not listed in 2015; 1.7% in 2017 and 7.9% in 2018), Other (5.8% in 2015, 3.0% in 2017; not listed in 2018) and None (9.3% in 2015, 4.8% in 2017 and 4.8% in 2018).

The UK IT jobs website itjobswatch.co.uk reports that as of late September 2016, 29.27% of UK permanent software development job openings have cited Git,^[74] ahead of 12.17% for Microsoft [Team Foundation Server](#),^[75] 10.60% for [Subversion](#),^[76] 1.30% for [Mercurial](#),^[77] and 0.48% for [Visual SourceSafe](#).^[78]

Extensions



There are many *Git extensions*, like [Git LFS](#),^[79] which started as an extension to Git in the GitHub community and now is widely used by other repositories. Extensions are usually independently developed and maintained by different people, but in some point in the future a widely used extension can be merged to Git.

Microsoft developed the [Virtual File System for Git](#) (VFS for Git; formerly Git Virtual File System or GVFS) extension to handle the size of the [Windows](#) source-code tree as part of their 2017 migration from [Perforce](#). VFS for Git allows cloned repositories to use placeholders whose contents are downloaded only once a file is accessed.^[79]

Security

Git does not provide access-control mechanisms, but was designed for operation with other tools that specialize in access control.^[80]

On 17 December 2014, an exploit was found affecting the [Windows](#) and [macOS](#) versions of the Git client. An attacker could perform [arbitrary code execution](#) on a target computer with Git installed by creating a malicious Git tree (directory) named `.git` (a directory in Git repositories that stores all the data of the repository) in a different case (such as `.GIT` or `.Git`, needed because Git does not allow the all-lowercase version of `.git` to be created manually) with malicious files in the `.git/hooks` subdirectory (a folder with executable files that Git runs) on a repository that the attacker made or on a repository that the attacker can modify. If a Windows or Mac user *pulls* (downloads) a version of the repository with the malicious directory, then switches to that directory, the `.git` directory will be overwritten (due to the case-insensitive trait of the Windows and Mac filesystems) and the malicious executable files in `.git/hooks` may be run, which results in the attacker's commands being executed. An attacker could also modify the `.git/config` configuration file, which allows the attacker to create malicious Git aliases (aliases for Git commands or external commands) or modify extant aliases to execute malicious commands when run. The vulnerability was patched in version 2.2.1 of Git, released on 17 December 2014, and announced the next day.^{[81][82]}

Git version 2.6.1, released on 29 September 2015, contained a patch for a security vulnerability (CVE-2015-7545)^[83] that allowed arbitrary code execution.^[84] The vulnerability was exploitable if

an attacker could convince a victim to clone a specific URL, as the arbitrary commands were embedded in the URL itself.^[85] An attacker could use the exploit via a [man-in-the-middle attack](#) if the connection was unencrypted,^[85] as they could redirect the user to a URL of their choice. Recursive clones were also vulnerable, since they allowed the controller of a repository to specify arbitrary URLs via the `gitmodules` file.^[85]

Git uses [SHA-1](#) hashes internally. Linus Torvalds has responded that the hash was mostly to guard against accidental corruption, and the security a [cryptographically secure hash](#) gives was just an accidental side effect, with the main security being signing elsewhere.^{[86][87]}

See also

- [Comparison of version control software](#)
- [Comparison of source code hosting facilities](#)
- [List of revision control software](#)
































References

1. ["Initial revision of "git", the information manager from hell"](#) [↗](#). Github. 8 April 2005. [Archived](#) [↗](#) from the original on 16 November 2015. Retrieved 20 December 2015.
2. ["Commit Graph"](#) [↗](#). Github. 8 June 2016. [Archived](#) [↗](#) from the original on 20 January 2016. Retrieved 19 December 2015.
3. ["Releases - git/git"](#) [↗](#). Retrieved 5 November 2019.
4. ["Git Source Code Mirror"](#) [↗](#). [Archived](#) [↗](#) from the original on 8 February 2017. Retrieved 1 January 2017.
5. ["Git's GPL license at github.com"](#) [↗](#). *github.com*. 18 January 2010. [Archived](#) [↗](#) from the original on 11 April 2016. Retrieved 12 October 2014.
6. ["Git's LGPL license at github.com"](#) [↗](#). *github.com*. 20 May 2011. [Archived](#) [↗](#) from the original on 11 April 2016. Retrieved 12 October 2014.
7. ["Tech Talk: Linus Torvalds on git \(at 00:01:30\)"](#) [↗](#). YouTube. [Archived](#) [↗](#) from the original on 20 December 2015. Retrieved 20 July 2014.
8. Scopatz, Anthony; Huff, Kathryn D. (2015). *Effective Computation in Physics* [↗](#). O'Reilly Media, Inc. p. 351. [ISBN 9781491901595](#). [Archived](#) [↗](#) from the original on 7 May 2016. Retrieved 20 April 2016.
9. Torvalds, Linus (7 April 2005). ["Re: Kernel SCM saga."](#) [↗](#) *linux-kernel* (Mailing list). "So I'm writing some scripts to try to track things a whole lot faster."

10. [Torvalds, Linus](#) (10 June 2007). "Re: fatal: serious inflate inconsistency" [↗](#). *git* (Mailing list).
11. [Linus Torvalds](#) (3 May 2007). *Google tech talk: Linus Torvalds on git* [↗](#). Event occurs at 02:30. [Archived](#) [↗](#) from the original on 28 May 2007. Retrieved 16 May 2007.
12. "A Short History of Git" [↗](#). *Pro Git* [↗](#) (2nd ed.). Apress. 2014. [Archived](#) [↗](#) from the original on 25 December 2015. Retrieved 26 December 2015.
13. Chacon, Scott (24 December 2014). *Pro Git* [↗](#) (2nd ed.). New York, NY: [Apress](#). pp. 29–30. ISBN 978-1-4842-0077-3. [Archived](#) [↗](#) from the original on 25 December 2015.
14. [BitKeeper and Linux: The end of the road?](#) [↗](#) [linux.com](#) [↗](#) [Archived](#) [↗](#) 8 June 2017 at the [Wayback Machine](#)
15. McAllister, Neil (2 May 2005). "Linus Torvalds' BitKeeper blunder" [↗](#). *InfoWorld*. IDG. [Archived](#) [↗](#) from the original on 26 August 2015. Retrieved 8 September 2015.
16. [Torvalds, Linus](#) (27 February 2007). "Re: Trivia: When did git self-host?" [↗](#). *git* (Mailing list).
17. [Torvalds, Linus](#) (6 April 2005). "Kernel SCM saga." [↗](#) *linux-kernel* (Mailing list).
18. [Torvalds, Linus](#) (17 April 2005). "First ever real kernel git merge!" [↗](#). *git* (Mailing list).
19. [Mackall, Matt](#) (29 April 2005). "Mercurial 0.4b vs git patchbomb benchmark" [↗](#). *git* (Mailing list).
20. [Torvalds, Linus](#) (17 June 2005). "Linux 2.6.12" [↗](#). *git-commits-head* (Mailing list).
21. [Torvalds, Linus](#) (27 July 2005). "Meet the new maintainer..." [↗](#) *git* (Mailing list).
22. [Hamano, Junio C.](#) (21 December 2005). "Announce: Git 1.0.0" [↗](#). *git* (Mailing list).
23. "GitFaq: Why the 'Git' name?" [↗](#). [Git.or.cz](#). [Archived](#) [↗](#) from the original on 23 July 2012. Retrieved 14 July 2012.
24. "After controversy, Torvalds begins work on 'git' " [↗](#). *PC World*. 14 July 2012. [Archived](#) [↗](#) from the original on 1 February 2011. "Torvalds seemed aware that his decision to drop BitKeeper would also be controversial. When asked why he called the new software, 'git', [British](#) slang meaning 'a rotten person', he said. 'I'm an egotistical bastard, so I name all my projects after myself. First Linux, now git.'"
25. "git(1) Manual Page" [↗](#). [Archived](#) [↗](#) from the original on 21 June 2012. Retrieved 21 July 2012.
26. "Initial revision of 'git', the information manager from hell · git/git@e83c516" [↗](#). *GitHub*. [Archived](#) [↗](#) from the original on 8 October 2017. Retrieved 21 January 2016.
27. "Git Github main page" [↗](#).
28. "How to maintain Git, Official Git documentation" [↗](#).

29. <https://github.com/git/git/releases> ↗
30. Torvalds, Linus (5 May 2006). "Re: [ANNOUNCE] Git wiki" ↗. *linux-kernel* (Mailing list). "Some historical background" on Git's predecessors
31. Torvalds, Linus (8 April 2005). "Re: Kernel SCM saga" ↗. *linux-kernel* (Mailing list). Retrieved 20 February 2008.
32. Torvalds, Linus (23 March 2006). "Re: Errors GITtifying GCC and Binutils" ↗. *git* (Mailing list).
33. Torvalds, Linus (20 October 2006). "Re: VCS comparison table" ↗. *git* (Mailing list). A discussion of Git vs. BitKeeper.
34. Torvalds, Linus (19 October 2006). "Re: VCS comparison table" ↗. *git* (Mailing list).
35. Jst's Blog on Mozillazine "bzt/hg/git performance" ↗. Archived from [the original](#) ↗ on 29 May 2010. Retrieved 12 February 2015.
36. Dreier, Roland (13 November 2006). "Oh what a relief it is" ↗. Archived ↗ from the original on 16 January 2009., observing that "git log" is 100x faster than "svn log" because the latter must contact a remote server.
37. "Trust" ↗. *Git Concepts*. Git User's Manual. 18 October 2006. Archived ↗ from the original on 22 February 2017.
38. Torvalds, Linus. "Re: VCS comparison table" ↗. *git* (Mailing list). Retrieved 10 April 2009., describing Git's script-oriented design
39. iabervon (22 December 2005). "Git rocks!" ↗. Archived ↗ from the original on 14 September 2016., praising Git's scriptability.
40. "Git User's Manual" ↗. 5 August 2007. Archived ↗ from the original on 22 February 2017.
41. Torvalds, Linus (10 April 2005). "Re: more git updates." ↗ *linux-kernel* (Mailing list).
42. Haible, Bruno (11 February 2007). "how to speed up "git log"?" ↗. *git* (Mailing list).
43. Torvalds, Linus (1 March 2006). "Re: impure renames / history tracking" ↗. *git* (Mailing list).
44. Hamano, Junio C. (24 March 2006). "Re: Errors GITtifying GCC and Binutils" ↗. *git* (Mailing list).
45. Hamano, Junio C. (23 March 2006). "Re: Errors GITtifying GCC and Binutils" ↗. *git* (Mailing list).
46. Torvalds, Linus (28 November 2006). "Re: git and bzt" ↗. *git* (Mailing list)., on using `git-blame` to show code moved between source files.
47. Torvalds, Linus (18 July 2007). "git-merge(1)" ↗. Archived ↗ from the original on 16 July 2016.

48. Torvalds, Linus (18 July 2007). "CrissCrossMerge" [↗](#). Archived from [the original](#) [↗](#) on 13 January 2006.
49. Torvalds, Linus (10 April 2005). "Re: more git updates..." [↗](#) *linux-kernel* (Mailing list).
50. [Gerrit Code Review – Project Configuration File Format](#) [↗](#).
51. "downloads" [↗](#). Archived [↗](#) from the original on 8 May 2012. Retrieved 14 May 2012.
52. "msysGit" [↗](#). Archived [↗](#) from the original on 10 October 2016. Retrieved 20 September 2016.
53. "JGit" [↗](#). Archived [↗](#) from the original on 31 August 2012. Retrieved 24 August 2012.
54. "Git - go-git" [↗](#). *git-scm.com*. Retrieved 19 April 2019.
55. "SQL interface to Git repositories, written in Go." [↗](#), *github.com*, retrieved 19 April 2019
56. "Keybase launches encrypted git" [↗](#). *keybase.io*. Retrieved 19 April 2019.
57. "Dulwich" [↗](#). Archived [↗](#) from the original on 29 May 2012. Retrieved 27 August 2012.
58. "libgit2" [↗](#). Archived [↗](#) from the original on 11 April 2016. Retrieved 24 August 2012.
59. "rugged" [↗](#). Archived [↗](#) from the original on 24 July 2013. Retrieved 24 August 2012.
60. "pygit2" [↗](#). Archived [↗](#) from the original on 5 August 2015. Retrieved 24 August 2012.
61. "hlibgit2" [↗](#). Archived [↗](#) from the original on 25 May 2013. Retrieved 30 April 2013.
62. "js-git: a JavaScript implementation of Git" [↗](#). Archived [↗](#) from the original on 7 August 2013. Retrieved 13 August 2013.
63. "Git - Git Daemon" [↗](#). *git-scm.com*. Retrieved 10 July 2019.
64. [4.4 Git on the Server – Setting Up the Server](#) [↗](#) Archived [↗](#) 22 October 2014 at the [Wayback Machine](#), Pro Git.
65. "1.4 Getting Started – Installing Git" [↗](#). *git-scm.com*. Archived [↗](#) from the original on 2 November 2013. Retrieved 1 November 2013.
66. [Diffusion User Guide: Repository Hosting](#) [↗](#).
67. "Eclipse Community Survey 2014 results | Ian Skerrett" [↗](#). *ianskerrett.wordpress.com*. 23 June 2014. Archived [↗](#) from the original on 25 June 2014. Retrieved 23 June 2014.
68. "Results of Eclipse Community Survey 2012" [↗](#). Archived [↗](#) from the original on 11 April 2016.
69. "Compare Repositories – Open Hub" [↗](#). Archived [↗](#) from the original on 7 September 2014.
70. [Stack Overflow Annual Developer Survey](#) [↗](#)

71. ["Stack Overflow Developer Survey 2015"](#) . Stack Overflow. [Archived](#)  from the original on 4 May 2019. Retrieved 29 May 2019.
72. ["Stack Overflow Developer Survey 2017"](#) . Stack Overflow. [Archived](#)  from the original on 4 May 2019. Retrieved 29 May 2019.
73. ["Stack Overflow Developer Survey 2018"](#) . Stack Overflow. [Archived](#)  from the original on 4 May 2019. Retrieved 29 May 2019.
74. ["Git \(software\) Jobs, Average Salary for Git Distributed Version Control System Skills"](#) . Itjobswatch.co.uk. [Archived](#)  from the original on 8 October 2016. Retrieved 30 September 2016.
75. ["Team Foundation Server Jobs, Average Salary for Microsoft Team Foundation Server \(TFS\) Skills"](#) . Itjobswatch.co.uk. [Archived](#)  from the original on 29 October 2016. Retrieved 30 September 2016.
76. ["Subversion Jobs, Average Salary for Apache Subversion \(SVN\) Skills"](#) . Itjobswatch.co.uk. [Archived](#)  from the original on 25 October 2016. Retrieved 30 September 2016.
77. ["Mercurial Jobs, Average Salary for Mercurial Skills"](#) . Itjobswatch.co.uk. [Archived](#)  from the original on 23 September 2016. Retrieved 30 September 2016.
78. ["VSS/SourceSafe Jobs, Average Salary for Microsoft Visual SourceSafe \(VSS\) Skills"](#) . Itjobswatch.co.uk. [Archived](#)  from the original on 29 October 2016. Retrieved 30 September 2016.
79. ["Windows switch to Git almost complete: 8,500 commits and 1,760 builds each day"](#) . *Ars Technica*. [Archived](#)  from the original on 24 May 2017. Retrieved 24 May 2017.
80. ["Archived copy"](#) . [Archived](#)  from the original on 14 September 2016. Retrieved 6 September 2016.
81. Pettersen, Tim (20 December 2014). ["Securing your Git server against CVE-2014-9390"](#) . [Archived](#)  from the original on 24 December 2014. Retrieved 22 December 2014.
82. Hamano, J. C. (18 December 2014). ["\[Announce\] Git v2.2.1 \(and updates to older maintenance tracks\)"](#) . Newsgroup: [gmane.linux.kernel](#) . Archived from [the original](#)  on 19 December 2014. Retrieved 22 December 2014.
83. ["CVE-2015-7545"](#) . 15 December 2015. [Archived](#)  from the original on 26 December 2015. Retrieved 26 December 2015.
84. ["Git 2.6.1"](#) . 29 September 2015. [Archived](#)  from the original on 11 April 2016. Retrieved 26 December 2015.
85. Blake Burkhart; et al. (5 October 2015). ["Re: CVE Request: git"](#) . [Archived](#)  from the original on 27 December 2015. Retrieved 26 December 2015.

86. "hash - How safe are signed git tags? Only as safe as SHA-1 or somehow safer?" [↗](#). Information Security Stack Exchange. 22 September 2014. [Archived](#) [↗](#) from the original on 24 June 2016.
87. "Why does Git use a cryptographic hash function?" [↗](#). Stack Overflow. 1 March 2015. [Archived](#) [↗](#) from the original on 1 July 2016.

External links

Wikimedia Commons has media related to **Git**.

Wikibooks has a book on the topic of: **Git**

- [Official website](#) [↗](#) [✎](#)
- [Git](#) [↗](#) at [Open Hub](#)

 **Last edited 4 days ago** by K144pl

