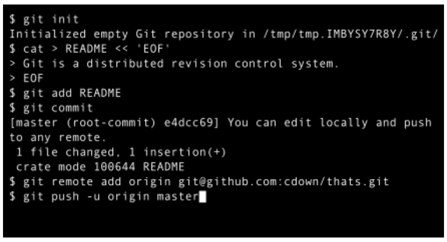
Git

Git (/ɡɪt/)[8] is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different computers).[9][10][11]

Git was originally authored by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development.[12] Since 2005, Junio Hamano has been the core maintainer. As with most other distributed version control systems,

Git



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

Original author(s)	Lin To
Developer(s)	Junio Haman and others[2

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 shared

Initial release	7 April 2005
Stable release	2.41.0 / 1 June 2023
Repository	git.kernel.org/pub/scm/git/git.git (https://git.kernel.org/pub/scm/git/git.git)

under the GPL-2.0-only license.

History

Git development was started by Torvalds in April 2005 when the proprietary source-control management (SCM) system used for Linux kernel development since 2002, BitKeeper, revoked its free license for Linux

Written in	Primarily i C, with GU and programm scripts written in Shell scrip Perl, Tcl ar Python[4][5
Operating system	PC (Li ma So AI) Wi
Available in	English
Type	Version

development.[14][15] The copyright holder of BitKeeper, Larry McVoy, claimed that Andrew Tridgell had created SourcePuller by reverse engineering the BitKeeper protocols.[16] The same incident also spurred the creation of another version-control system, Mercurial.

	control
License	GPL-2.0-only[i][7]
Website	git-scm.com (https://git-scm.com)

Torvalds wanted a distributed system that he could use like BitKeeper, but none of the available free systems met his needs. He 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 criterion, he specified that patching should take no more than three seconds, and added three more goals:[9]

- 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 version-control system in use at the time, 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.[17] Torvalds announced the project on 6 April and became self-hosting the next day.[17][18] The first merge of multiple branches took place on 18 April.[19] 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.[20] On 16 June, Git managed the kernel 2.6.12 release.[21]

Torvalds turned over maintenance on 26 July 2005 to Junio Hamano, a major contributor to the project.[22] Hamano was responsible for the 1.0 release on 21 December 2005.[23]

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'."[24][25] The man page describes Git as "the stupid content tracker".[26] The read-me file of the source code elaborates further:[27]

> *"git" can mean anything, depending on your mood.*
>
> - *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.*

The source code for Git refers to the program as, "the information manager from hell".

Releases

List of Git releases:[28]

Version	Original release date	Latest (patch) version	Patch release date	Notable changes
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	2.14.6	2019-12-07	
2.15	2017-10-30	2.15.4	2019-12-07	
2.16	2018-01-17	2.16.6	2019-12-07	
2.17	2018-04-02	2.17.6	2021-03-09	
2.18	2018-06-21	2.18.5	2021-03-09	
2.19	2018-09-10	2.19.6	2021-03-09	
2.20	2018-12-	2.20.5	2021-03-	

	09		09	
2.21	2019-02-24	2.21.4	2021-03-09	
2.22	2019-06-07	2.22.5	2021-03-09	
2.23	2019-08-16	2.23.4	2021-03-09	
2.24	2019-11-04	2.24.4	2021-03-09	
2.25	2020-01-13	2.25.5	2021-03-09	Sparse checkout management made easy[29]
2.26	2020-03-22	2.26.3	2021-03-09	• Protocol version 2 is now the default • Some new config tricks • Updates to git sparse-checkout [30]
2.27	2020-06-01	2.27.1	2021-03-09	
2.28	2020-07-27	2.28.1	2021-03-09	• Introducing `init.defaultBranch` • Changed-path Bloom filter [31]
2.29	2020-10-19	2.29.3	2021-03-09	• Experimental SHA-256 support • Negative refspecs • New `git shortlog` tricks [32]
2.30	2020-12-27	2.30.9	2023-04-25	• Userdiff for PHP update, Rust, CSS update • The command line completion script (in contrib/) learned that "git stash show" takes the options "git diff" takes. [33]

2.31	2021-03-15	2.31.8	2023-04-25	• `git difftool` adds `--skip-to` option • `--format` enhancements for machine readable • `git pull` warning to specify rebase or merge [34][35]
2.32	2021-06-06	2.32.7	2023-04-25	
2.33	2021-08-16	2.33.8	2023-04-25	
2.34	2021-11-15	2.34.8	2023-04-25	
2.35	2022-01-25	2.35.8	2023-04-25	
2.36	2022-04-18	2.36.6	2023-04-25	
2.37	2022-06-27	2.37.7	2023-04-25	
2.38	2022-10-02	2.38.5	2023-04-25	
2.39	2022-12-12	2.39.3	2023-04-25	
2.40	2023-03-14	2.40.1	2023-04-25	

Legend: Old version Older version, still maintained **Latest version**

Sources:[36][37]

Design

Git's design was inspired by BitKeeper and Monotone.[38][39] 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.[39] The core Git project has since become a complete version-control system that is usable directly.[40] While strongly influenced by BitKeeper, Torvalds deliberately avoided conventional approaches, leading to a unique design.[41]

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:[42]

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.[43]

Compatibility with existing systems and protocols

Repositories can be published via Hypertext Transfer Protocol Secure (HTTPS), 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 existing CVS clients and IDE plugins to access Git repositories. Subversion repositories can be used directly with git-svn.[44]

Efficient handling of large projects

Torvalds has described Git as being very fast and scalable,[45] and performance tests done by Mozilla[46] showed that it was an order of magnitude faster diffing large repositories than Mercurial and GNU Bazaar; fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server.[47]

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.[48] (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.[49] 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.[50]

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.[51]

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`.[52]

Periodic explicit object packing

Git stores each newly created object as a separate file. Although individually compressed, this takes up 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, 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 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.[53]

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.[54] 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.[55] 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.[56][57] Source code files are sometimes split or merged, or simply renamed,[58] 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.[59] (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:[60]

- *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 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*[61]

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

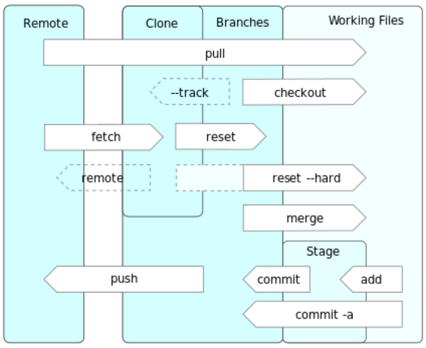
Data structures

Git's primitives are not inherently a source-code management system.

Torvalds explains:[62]

> *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,[40] 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 a connection point between the object database and the working tree.

The object database contains five types of objects:[63][53]

- 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.[64]). In git each blob is a version of a file, it holds the file's data.
- 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 history. It contains the name of a tree object (of the top-level source directory), a timestamp, 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.
- A **packfile** object collects various other objects into a zlib-compressed bundle for compactness and ease of transport over network protocols.

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.

Additionally, git stores labels called refs (short for references) to indicate the locations of various commits. They are stored in the reference database and are respectively:[65]

- **Heads (branches)**: Named references that are advanced automatically to the new commit when a commit is made on top of them.
- **HEAD**: A reserved head that will be compared against the working tree to create a commit.
- **Tags**: Like branch references but fixed to a particular commit. Used to label important points in history.

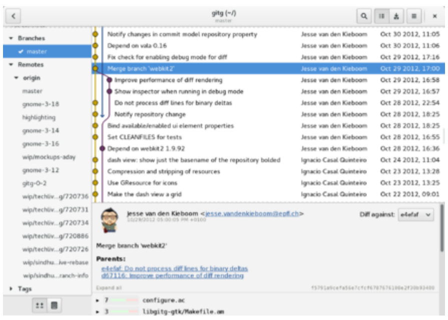
References

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,[66]
- *tags*: see above.

Implementations



gitg is a graphical front-end using GTK+.

Git (the main implementation in C) is primarily developed on Linux, although it also supports most major operating systems, including the BSDs (DragonFly BSD, FreeBSD, NetBSD, and OpenBSD), Solaris, macOS, and Windows.[67][68]

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-w64 port of the GNU Compiler Collection, Perl 5, MSYS2 (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.[69] The git official website currently maintains a build of Git for Windows, still using the MSYS2 environment.[70]

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.[71]

Go-git is an open-source implementation of Git written in pure Go.[72] It is currently used for backing projects as a SQL interface for Git code repositories[73] and providing encryption for Git.[74]

The Dulwich implementation of Git is a pure Python software component for Python 2.7, 3.4 and 3.5.[75]

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.[76] It has bindings for many programming languages, including Ruby, Python, and Haskell.[77][78][79]

JS-Git is a JavaScript implementation of a subset of Git.[80]

Git server

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 a built-in command `git daemon` which starts a simple TCP server running on the GIT protocol.[81] 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.[82] Git servers typically listen on TCP port 9418.[83]

Open source

- Hosting the Git server using the Git Binary.[84]
- Gerrit, a Git server configurable to support code reviews and provide 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, and no database is 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, Git support is not as prominent.[85]
- RhodeCode Community Edition (CE), supporting Git, Mercurial and Subversion with an AGPLv3 license.

- Kallithea, supporting both Git and Mercurial, developed in Python with GPL license.
- External projects like gitolite,[86] which provide scripts on top of Git software to provide fine-grained access control.
- There are several other FLOSS solutions for self-hosting, including Gogs[87] and Gitea, a fork of Gogs, both developed in Go language with MIT license.

Git server as a service

There are many offerings of Git repositories as a service. The most popular are GitHub, SourceForge, Bitbucket and GitLab.[88][89][90][91][92]

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[93] 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.[94] Open-source directory Black Duck Open Hub reports a similar uptake among open-source projects.[95]

Stack Overflow has included version control in their annual developer survey[96] in 2015 (16,694 responses),[97] 2017 (30,730 responses),[98] 2018 (74,298 responses)[99] and 2022 (71,379 reponses).[100] Git was the overwhelming favorite of responding developers in these surveys, reporting as high as 93.9% in 2022.

Version control systems used by responding developers:

Name	2015	2017	2018	2022
Git	69.3%	69.2%	87.2%	93.9%
Subversion	36.9%	9.1%	16.1%	5.2%
TFVC	12.2%	7.3%	10.9%	[ii]
Mercurial	7.9%	1.9%	3.6%	1.1%
CVS	4.2%	[ii]	[ii]	[ii]
Perforce	3.3%	[ii]	[ii]	[ii]
VSS	[ii]	0.6%	[ii]	[ii]
ClearCase	[ii]	0.4%	[ii]	[ii]
Zip file backups	[ii]	2.0%	7.9%	[ii]
Raw network sharing	[ii]	1.7%	7.9%	[ii]
Other	5.8%	3.0%	[ii]	[ii]
None	9.3%	4.8%	4.8%	4.3%

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,[101] ahead of 12.17% for Microsoft Team Foundation Server,[102] 10.60% for Subversion,[103] 1.30% for Mercurial,[104] and 0.48% for Visual SourceSafe.[105]

Extensions

There are many *Git extensions*, like Git LFS (https://github.com/git-lfs/git-lfs) , which started as an extension to Git in the GitHub community and is now widely used by other repositories. Extensions are usually independently developed and maintained by different people, but at some point in the future, a widely used extension can be merged with Git.

Other open-source Git extensions include:

- git-annex, a distributed file synchronization system based on Git

- git-flow, a set of Git extensions to provide high-level repository operations for Vincent Driessen's branching model (https://nvie.com/posts/a-successful-git-branching-model/)
- git-machete (https://github.com/VirtusLab/git-machete) , a repository organizer & tool for automating rebase/merge/pull/push operations

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.[106]

Conventions

Git does not impose many restrictions on how it should be used, but some conventions are adopted in order to organize histories, especially those which require the cooperation of many contributors.

- The *master* branch is created by default with *git init*[107] and is often used as the branch that other changes are merged into.[108] Correspondingly, the default name of the upstream remote is *origin* and so the name of the default remote branch is *origin/master*. The use of *master* as the default branch name is not universally true. Repositories created in GitHub and GitLab initialize with a *main* branch instead of *master*.[109][110]

- Pushed commits should usually not be overwritten, but should rather be *revert*ed[111] (a commit is made on top which reverses the changes to an earlier commit). This prevents shared new commits based on shared commits from being invalid because the commit on which they are based does not exist in the remote. If the commits contain sensitive information, they should be removed, which involves a more complex procedure to rewrite history.

- The *git-flow*[112] workflow and naming conventions are often adopted to distinguish feature specific unstable histories (feature/*), unstable shared histories (develop), production ready histories (main), and emergency patches to released products (hotfix).
- *Pull requests* are not a feature of git, but are commonly provided by git cloud services. A pull request is a request by one user to merge a branch of their repository fork into another repository sharing the same history (called the *upstream* remote).[113] The underlying

function of a pull request is no different than that of an administrator of a repository pulling changes from another remote (the repository that is the source of the pull request). However, the pull request itself is a ticket managed by the hosting server which initiates scripts to perform these actions; it is not a feature of git SCM.

Security

Git does not provide access-control mechanisms, but was designed for operation with other tools that specialize in access control.[114]

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.[115][116]

Git version 2.6.1, released on 29 September 2015, contained a patch for a security vulnerability (CVE-2015-7545 (https://www.cve.org/CVERecord?id=CVE-2015-7545))[117] that allowed arbitrary code execution.[118] 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.[119] An attacker could use the exploit via a man-in-the-middle attack if the connection was unencrypted,[119] 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.[119]

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.[120][121] Since a demonstration of the SHAttered attack against git in 2017, git was modified to use a SHA-1 variant resistant to this attack. A plan for hash function transition is being written since February 2020.[122]

Trademark

"Git" is a registered word trademark of Software Freedom Conservancy under US500000085961336 (https://www.tmdn.org/tmview/#/tmview/detail/US500000085961336) since 2015-02-03.

See also

- Comparison of version-control software
- Comparison of source-code-hosting facilities

Free and open-source software portal

Linux portal

- List of version-control software

Internet portal

Notes

i. *GPL-2.0-only since 2005-04-11. Some parts under compatible licenses such as LGPLv2.1.*[6]

ii. *Not listed as an option in this survey*

References

1. *"Initial revision of "git", the information manager from hell" (https://github.com/git/git/commit/e83c5163316f89bfbde7d9ab23ca2e25604af290) . GitHub. 8 April 2005. Archived (https://web.archive.org/web/20151116175401/https://github.com/git/git/commit/e83c5163316f89bfbde7d9ab23ca2e25604af290) from the*

original on 16 November 2015. Retrieved 20 December 2015.

2. *"Commit Graph" (https://github.com/git/git/graphs/contributors) . GitHub. 8 June 2016. Archived (https://web.archive.org/web/20160120121816/https://github.com/git/git/graphs/contributors) from the original on 20 January 2016. Retrieved 19 December 2015.*
3. *Junio C Hamano (1 June 2023). "[ANNOUNCE] Git v2.41.0" (https://lore.kernel.org/git/xmqqleh3a3wm.fsf@gitster.g/T/#u) . Retrieved 1 June 2023.*

4. *"Git website" (https://git-scm.com/about/small-and-fast) . Archived (https://web.archive.org/web/20220609042334/https://git-scm.com/about/small-and-fast) from the original on 9 June 2022. Retrieved 9 June 2022.*
5. *"Git Source Code Mirror" (https://github.com/git/git) . GitHub. Archived (https://web.archive.org/web/20220603081319/https://github.com/git/git) from the original on 3 June 2022. Retrieved 9 June 2022.*

6. *"Git's LGPL license at github.com" (https://github.com/git/git/blob/master/LGPL-2.1) . GitHub. 20 May 2011. Archived (https://web.archive.org/web/20160411135049/https://github.com/git/git/blob/master/LGPL-2.1) from the original on 11 April 2016. Retrieved 12 October 2014.*
7. *"Git's GPL license at github.com" (https://github.com/git/git/blob/master/COPYING) . GitHub. 18 January 2010. Archived (https://web.archive.org/web/20160411135124/https://github.com/git/git/blob/master/COPYING) from the original on 11 April 2016. Retrieved 12 October 2014.*

8. *"Tech Talk: Linus Torvalds on git (at 00:01:30)" (https://www.youtube.com/watch?v=4XpnKHJAok8&t=1m30s) . Archived (https://web.archive.org/web/20151220133030/https://www.youtube.com/watch?v=4XpnKHJAok8&t=1m30s) from the original on 20 December 2015. Retrieved 20 July 2014 – via YouTube.*
9. *Torvalds, Linus (7 April 2005). "Re: Kernel SCM saga." (https://marc.info/?l=linux-kernel&m=111288700902396) linux-kernel (Mailing list). Archived (https://web.archive.org/web/20190701210808/https://marc.info/?l=linux-kernel) from the original on 1 July 2019. Retrieved 3 February 2017. "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" (https://marc.info/?l=git&m=118143549107708) . git (Mailing list).*

11. *Linus Torvalds (3 May 2007). Google tech talk: Linus Torvalds on git (https://www.youtube.com/watch?v=4XpnKHJAok8) . Event occurs at 02:30. Archived (https://web.archive.org/web/20070528041814/http://www.youtube.com/watch?v=4XpnKHJAok8) from the original on 28 May 2007. Retrieved 16 May 2007.*

12. *"A Short History of Git" (https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git). Pro Git (https://git-scm.com/book/en/v2) (2nd ed.). Apress. 2014. Archived (https://web.archive.org/web/20151225223054/http://git-scm.com/book/en/v2) from the original on 25 December 2015. Retrieved 26 December 2015.*
13. *Chacon, Scott (24 December 2014). Pro Git (https://git-scm.com/book/en/v2) (2nd ed.). New York, NY: Apress. pp. 29–30. ISBN 978-1-4842-0077-3. Archived (https://web.archive.org/web/20151225223054/http://git-scm.com/book/en/v2) from the original on 25 December 2015.*

14. *Brown, Zack (27 July 2018). "A Git Origin Story" (https://www.linuxjournal.com/content/git-origin-story) . Linux Journal. Linux Journal. Archived (https://web.archive.org/web/20200413113107/https://www.linuxjournal.com/content/git-origin-story) from the original on 13 April 2020. Retrieved 28 May 2020.*
15. *"BitKeeper and Linux: The end of the road?" (https://www.linux.com/news/bitkeeper-and-linux-end-road/) . Linux.com. 11 April 2005. Retrieved 18 May 2023.*

16. *McAllister, Neil (2 May 2005). "Linus Torvalds' BitKeeper blunder" (http://www.infoworld.com/article/2670360/operating-systems/linus-torvalds--bitkeeper-blunder.html) . InfoWorld. Archived (https://web.archive.org/web/20150826064920/http://www.infoworld.com/article/2670360/operating-systems/linus-torvalds--bitkeeper-blunder.html) from the original on 26 August 2015. Retrieved 8 September 2015.*

17. *Torvalds, Linus (27 February 2007). "Re: Trivia: When did git self-host?" (https://marc.info/?l=git&m=117254154130732) . git (Mailing list).*

18. *Torvalds, Linus (6 April 2005). "Kernel SCM saga." (https://marc.info/?l=linux-kernel&m=111280216717070) linux-kernel (Mailing list).*

19. *Torvalds, Linus (17 April 2005). "First ever real kernel git merge!" (https://marc.info/?l=git&m=111377572329534) . git (Mailing list).*

20. *Mackall, Matt (29 April 2005). "Mercurial 0.4b vs git patchbomb benchmark" (https://marc.info/?l=git&m=111475459526688) . git (Mailing list).*

21. *Torvalds, Linus (17 June 2005). "Linux 2.6.12" (https://marc.info/?l=git-commits-head&m=111904216911731) . git-commits-head (Mailing list).*

22. *Torvalds, Linus (27 July 2005). "Meet the new maintainer..." (https://marc.info/?l=git&m=112243466603239) git (Mailing list).*

23. *Hamano, Junio C. (21 December 2005). "Announce: Git 1.0.0" (https://marc.info/?l=git&m=113515203321888) . git (Mailing list).*

24. *"GitFaq: Why the 'Git' name?" (https://git.wiki.kernel.org/index.php/GitFaq#Why_the_.27Git.27_name.3F) . Git.or.cz. Archived (https://web.archive.org/web/20120723224559/https://git.wiki.kernel.org/index.php/GitFaq#Why_the_.27Git.27_name.3F) from the original on 23 July 2012. Retrieved 14 July 2012.*

25. *"After controversy, Torvalds begins work on 'git' " (http://www.pcworld.idg.com.au/article/129776/after_controversy_torvalds_begins_work_git_/). PC World. 14 July 2012. Archived (https://web.archive.org/web/20110201184934/http://www.pcworld.idg.com.au/article/129776/after_controversy_torvalds_begins_work_git_/) 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.' "*

26. *"git(1) Manual Page" (https://git-scm.com/docs/git.html) . Archived (https://web.archive.org/web/20120621133627/http://www.git-scm.com/docs/git.html) from the original on 21 June 2012. Retrieved 21 July 2012.*
27. *"Initial revision of 'git', the information manager from hell · git/git@e83c516" (https://github.com/git/git/blob/e83c5163316f89bfbde7d9ab23ca2e25604af290/README) . GitHub. Archived (https://web.archive.org/web/20171008211145/https://github.com/git/git/blob/e83c5163316f89bfbde7d9ab23ca2e25604af290/README) from the original on 8 October 2017. Retrieved 21 January 2016.*

28. *"Tags" (https://github.com/git/git/tags) . GitHub. Archived (https://web.archive.org/web/20210929124441/https://github.com/git/git/tags) from the original on 29 September 2021. Retrieved 28 January 2022.*

29. *"Highlights from Git 2.25" (https://github.blog/2020-01-13-highlights-from-git-2-25/) . The GitHub Blog. 13 January 2020. Archived (https://web.archive.org/web/20210322043019/https://github.blog/2020-01-13-highlights-from-git-2-25/) from the original on 22 March 2021. Retrieved 27 November 2020. "A sparse checkout is nothing more than a list of file path patterns that Git should attempt to populate in your working copy when checking out the contents of your repository. Effectively, it works like a .gitignore, except it acts on the contents of your working copy, rather than on your index."*

30. *"Highlights from Git 2.26" (https://github.blog/2020-03-22-highlights-from-git-2-26/) . The GitHub Blog. 22 March 2020. Archived (https://web.archive.org/web/20210322043004/https://github.blog/2020-03-22-highlights-from-git-2-26/) from the original on 22 March 2021. Retrieved 25 November 2020. "You may remember when Git introduced a new version of its network fetch protocol way back in 2018. That protocol is now used by default in 2.26, so let's refresh ourselves on what that means. The biggest problem with the old protocol is that the server would immediately list all of the branches, tags, and other references in the repository before the client had a chance to send anything. For some repositories, this could mean*

sending megabytes of extra data, when the client really only wanted to know about the master branch. The new protocol starts with the client request and provides a way for the client to tell the server which references it's interested in. Fetching a single branch will only ask about that branch, while most clones will only ask about branches and tags. This might seem like everything, but server repositories may store other references (such as the head of every pull request opened in the repository since its creation). Now, fetches from large repositories improve in speed, especially when the fetch itself is small, which makes the cost of the initial reference advertisement more expensive relatively speaking. And the

best part is that you won't need to do anything! Due to some clever design, any client that speaks the new protocol can work seamlessly with both old and new servers, falling back to the original protocol if the server doesn't support it. The only reason for the delay between introducing the protocol and making it the default was to let early adopters discover any bugs."

31. *"Highlights from Git 2.28" (https://github.blog/2020-07-27-highlights-from-git-2-28/) . The GitHub Blog. 27 July 2020. Archived (https://web.archive.org/web/20210322043023/https://github.blog/2020-07-27-highlights-from-git-2-28/) from the original on 22 March 2021. Retrieved 25 November 2020.*

32. "Highlights from Git 2.29" (https://github.blog/2020-10-19-git-2-29-released/) . The GitHub Blog. 19 October 2020. Archived (https://web.archive.org/web/20210322043013/https://github.blog/2020-10-19-git-2-29-released/) from the original on 22 March 2021. Retrieved 25 November 2020.

33. "Git 2.30 Release Notes" (https://raw.githubusercontent.com/git/git/master/Documentation/RelNotes/2.30.0.txt) . Git Downloads. 27 December 2020. Archived (https://web.archive.org/web/20210322042959/https://raw.githubusercontent.com/git/git/master/Documentation/RelNotes/2.30.0.txt) from the original on 22 March 2021. Retrieved 27 December 2020.

34. "Git 2.31 Release Notes" (https://raw.githubusercontent.com/git/git/master/Documentation/RelNotes/2.31.1.txt) . Git Downloads. 3 April 2021. Retrieved 3 April 2021.
35. "Highlights from Git 2.31" (https://github.blog/2021-03-15-highlights-from-git-2-31/) . The GitHub Blog. 15 March 2021. Retrieved 2 July 2021.
36. "git/git" (https://github.com/git/git) . GitHub. Archived (https://web.archive.org/web/20170208051639/https://github.com/git/git) from the original on 8 February 2017. Retrieved 1 December 2011.

37. *Hamano, Junio (21 November 2007). "How to maintain Git" (https://github.com/git/git/blob/master/Documentation/howto/maintain-git.txt) . GitHub. Archived (https://web.archive.org/web/20210322043004/https://github.com/git/git/blob/master/Documentation/howto/maintain-git.txt) from the original on 22 March 2021. Retrieved 1 August 2020.*
38. *Torvalds, Linus (5 May 2006). "Re: [ANNOUNCE] Git wiki" (https://marc.info/?l=git&m=114685143200012) . linux-kernel (Mailing list). "Some historical background" on Git's predecessors*

39. *Torvalds, Linus (8 April 2005). "Re: Kernel SCM saga" (https://marc.info/?l=linux-kernel&m=111293537202443) . linux-kernel (Mailing list). Archived (https://web.archive.org/web/20210322043025/https://marc.info/?l=linux-kernel&m=111293537202443) from the original on 22 March 2021. Retrieved 20 February 2008.*

40. *Torvalds, Linus (23 March 2006). "Re: Errors GITtifying GCC and Binutils" (https://marc.info/?l=git&m=114314642000462) . git (Mailing list). Archived (https://web.archive.org/web/20210322043017/https://marc.info/?l=git&m=114314642000462) from the original on 22 March 2021. Retrieved 3 February 2017.*

41. *Torvalds, Linus (20 October 2006). "Re: VCS comparison table" (https://marc.info/?l=git&m=116129092117475) . git (Mailing list). Archived (https://web.archive.org/web/20210322043021/https://marc.info/?l=git&m=116129092117475) from the original on 22 March 2021. Retrieved 3 February 2017. A discussion of Git vs. BitKeeper.*

42. *"Git – A Short History of Git" (https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git) . git-scm.com. Archived (https://web.archive.org/web/20210322043028/https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git) from the original on 22 March 2021. Retrieved 15 June 2020.*

43. *"Git – Distributed Workflows" (https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows) . git-scm.com. Archived (https://web.archive.org/web/20141022020026/http://git-scm.com/book/en/Distributed-Git-Distributed-Workflows) from the original on 22 October 2014. Retrieved 15 June 2020.*
44. *Gunjal, Siddhesh (19 July 2019). "What is Version Control Tool? Explore Git and GitHub" (https://medium.com/analytics-vidhya/what-is-version-control-tool-explore-git-and-github-e8c4e719bc05) . Medium. Retrieved 25 October 2020.*
45. *Torvalds, Linus (19 October 2006). "Re: VCS comparison table" (https://marc.info/?l=git&m=116128307511686) . git (Mailing list).*

46. *Jst's Blog on Mozillazine "bzr/hg/git performance" (https://web.archive.org/web/20100529094107/http://weblogs.mozillazine.org/jst/archives/2006/11/vcs_performance.html) . Archived from the original (http://weblogs.mozillazine.org/jst/archives/2006/11/vcs_performance.html) on 29 May 2010. Retrieved 12 February 2015.*

47. *Dreier, Roland (13 November 2006). "Oh what a relief it is" (http://digitalvampire.org/blog/index.php/2006/11/16/oh-what-a-relief-it-is/) . Archived (https://web.archive.org/web/20090116175841/http://digitalvampire.org/blog/index.php/2006/11/16/oh-what-a-relief-it-is/) 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.*

48. "Trust" (https://www.kernel.org/pub/software/scm/git/docs/user-manual.html#trust) . Git Concepts. Git User's Manual. 18 October 2006. Archived (https://web.archive.org/web/20170222053056/https://www.kernel.org/pub/software/scm/git/docs/user-manual.html#trust) from the original on 22 February 2017.
49. Torvalds, Linus. "Re: VCS comparison table" (https://marc.info/?l=git&m=116118369005954) . git (Mailing list). Retrieved 10 April 2009., describing Git's script-oriented design

50. *iabervon (22 December 2005). "Git rocks!" (https://lwn.net/Articles/165202/) . Archived (https://web.archive.org/web/20160914100946/https://lwn.net/Articles/165202/) from the original on 14 September 2016., praising Git's scriptability.*
51. *"Git – Git SCM Wiki" (https://git.wiki.kernel.org/index.php/Git) . git.wiki.kernel.org. Retrieved 25 October 2020.*
52. *"Git User's Manual" (https://mirrors.edge.kernel.org/pub/software/scm/git/docs/git-gc.html) . 10 March 2020. Archived (https://web.archive.org/web/20200510190720/https://mirrors.edge.kernel.org/pub/software/scm/git/docs/git-gc.html) from the original on 10 May 2020.*

53. *"Git – Packfiles" (https://git-scm.com/book/en/v2/Git-Internals-Packfiles) . git-scm.com.*
54. *Torvalds, Linus (10 April 2005). "Re: more git updates." (https://marc.info/?l=linux-kernel&m=111314792424707) linux-kernel (Mailing list).*
55. *Haible, Bruno (11 February 2007). "how to speed up 'git log'?" (https://marc.info/?l=git&m=117119479505638) . git (Mailing list).*
56. *Torvalds, Linus (1 March 2006). "Re: impure renames / history tracking" (https://marc.info/?l=git&m=114123702826251) . git (Mailing list).*

57. *Hamano, Junio C. (24 March 2006). "Re: Errors GITtifying GCC and Binutils" (https://marc.info/?l=git&m=114316047119262) . git (Mailing list).*

58. *Hamano, Junio C. (23 March 2006). "Re: Errors GITtifying GCC and Binutils" (https://marc.info/?l=git&m=114315795227271) . git (Mailing list).*

59. *Torvalds, Linus (28 November 2006). "Re: git and bzr" (https://marc.info/?l=git&m=116473016012824) . git (Mailing list)., on using `git-blame` to show code moved between source files.*

60. *Torvalds, Linus (18 July 2007). "git-merge(1)" (https://www.kernel.org/pub/software/scm/git/docs/git-merge.html) . Archived (https://web.archive.org/web/20160716100147/https://www.kernel.org/pub/software/scm/git/docs/git-merge.html) from the original on 16 July 2016.*
61. *Torvalds, Linus (18 July 2007). "CrissCrossMerge" (https://web.archive.org/web/20060113122252/http://revctrl.org/CrissCrossMerge) . Archived from the original (http://revctrl.org/CrissCrossMerge) on 13 January 2006.*
62. *Torvalds, Linus (10 April 2005). "Re: more git updates..." (https://marc.info/?l=linux-kernel&m=111314792424707) linux-kernel (Mailing list).*

63. *"Git – Git Objects" (https://git-scm.com/book/en/v2/Git-Internals-Git-Objects) . git-scm.com.*
64. *Some of git internals (https://yurichev.com/news/20201220_git/)*
65. *"Git – Git References" (https://git-scm.com/book/en/v2/Git-Internals-Git-References) . git-scm.com.*
66. *"Project Configuration File Format" (https://gerrit-review.googlesource.com/Documentation/config-project-config.html) . Gerrit Code Review. Archived (https://web.archive.org/web/20201203033602/https://gerrit-review.googlesource.com/Documentation/config-project-config.html) from the original on 3 December 2020. Retrieved 2 February 2020.*

67. *"downloads" (https://git-scm.com/downloads) . Archived (https://web.archive.org/web/20120508021315/http://git-scm.com/downloads) from the original on 8 May 2012. Retrieved 14 May 2012.*

68. *"git package versions – Repology" (https://web.archive.org/web/20220119103832/https://repology.org/project/git/versions) . Archived from the original (https://repology.org/project/git/versions) on 19 January 2022. Retrieved 30 November 2021.*

69. *"msysGit" (https://github.com/msysgit/msysgit) . GitHub. Archived (https://web.archive.org/web/20161010143600/https://github.com/msysgit/msysgit) from the original on 10 October 2016. Retrieved 20 September 2016.*

70. *"Git – Downloading Package" (https://git-scm.com/download/win) . git-scm.com. (source code (https://github.com/git-for-windows/git))*
71. *"JGit" (https://www.eclipse.org/jgit/) . Archived (https://web.archive.org/web/20120831201256/http://eclipse.org/jgit/) from the original on 31 August 2012. Retrieved 24 August 2012.*
72. *"Git – go-git" (https://git-scm.com/book/en/v2/Appendix-B%3A-Embedding-Git-in-your-Applications-go-git) . git-scm.com. Retrieved 19 April 2019.*
73. *"SQL interface to Git repositories, written in Go." (https://github.com/src-d/gitbase) , github.com, retrieved 19 April 2019*

74. *"Keybase launches encrypted git" (https://keybase.io/blog/encrypted-git-for-everyone) . keybase.io. Retrieved 19 April 2019.*

75. *"Dulwich" (https://www.samba.org/~jelmer/dulwich/) . Archived (https://web.archive.org/web/20120529053253/http://www.samba.org/~jelmer/dulwich/) from the original on 29 May 2012. Retrieved 27 August 2012.*

76. *"libgit2" (https://github.com/libgit2/libgit2/blob/main/README.md) . GitHub. Archived (https://web.archive.org/web/20160411135623/https://github.com/libgit2/libgit2/blob/master/README.md) from the original on 11 April 2016. Retrieved 24 August 2012.*

77. "rugged" (https://github.com/libgit2/rugged) . GitHub. Archived (https://web.archive.org/web/20130724042431/https://github.com/libgit2/rugged) from the original on 24 July 2013. Retrieved 24 August 2012.

78. "pygit2" (https://github.com/libgit2/pygit2) . GitHub. Archived (https://web.archive.org/web/20150805001221/https://github.com/libgit2/pygit2) from the original on 5 August 2015. Retrieved 24 August 2012.

79. "hlibgit2" (https://hackage.haskell.org/package/hlibgit2) . Archived (https://web.archive.org/web/20130525064750/http://hackage.haskell.org/package/hlibgit2) from the original on 25 May 2013. Retrieved 30 April 2013.

80. *"js-git: a JavaScript implementation of Git" (https://github.com/creationix/js-git) . GitHub. Archived (https://web.archive.org/web/20130807173550/https://github.com/creationix/js-git) from the original on 7 August 2013. Retrieved 13 August 2013.*
81. *"Git – Git Daemon" (https://git-scm.com/book/en/v2/Git-on-the-Server-Git-Daemon) . git-scm.com. Retrieved 10 July 2019.*

82. *4.4 Git on the Server – Setting Up the Server (https://git-scm.com/book/en/Git-on-the-Server-Setting-Up-the-Server) Archived (https://web.archive.org/web/20141022015944/http://git-scm.com/book/en/Git-on-the-Server-Setting-Up-the-Server) 22 October 2014 at the Wayback Machine, Pro Git.*
83. *"1.4 Getting Started – Installing Git" (https://git-scm.com/book/ch4-1.html#The-Git-Protocol) . git-scm.com. Archived (https://web.archive.org/web/20131102192025/http://git-scm.com/book/ch4-1.html#The-Git-Protocol) from the original on 2 November 2013. Retrieved 1 November 2013.*

84. *Chacon, Scott; Straub, Ben (2014). "Git on the Server – Setting Up the Server" (https://git-scm.com/book/en/v2/Git-on-the-Server-Setting-Up-the-Server) . Pro Git (2nd ed.). Apress. ISBN 978-1484200773.*

85. *Diffusion User Guide: Repository Hosting (https://secure.phabricator.com/book/phabricator/article/diffusion_hosting/) Archived (https://web.archive.org/web/20200920235815/https://secure.phabricator.com/book/phabricator/article/diffusion_hosting/) 20 September 2020 at the Wayback Machine.*

86. *"Gitolite: Hosting Git Repositories" (https://gitolite.com/gitolite/index.html) .*

87. *"Gogs: A painless self-hosted Git service" (https://gogs.io/) .*

88. *Krill, Paul (28 September 2016). "Enterprise repo wars: GitHub vs. GitLab vs. Bitbucket" (https://www.infoworld.com/article/3123244/enterprise-repo-wars-github-vs-gitlab-vs-bitbucket.html) . InfoWorld. Retrieved 2 February 2020.*

89. *"github.com Competitive Analysis, Marketing Mix and Traffic" (https://web.archive.org/web/20130331175229/http://www.alexa.com/siteinfo/github.com) . Alexa. Archived from the original (https://www.alexa.com/siteinfo/github.com) on 31 March 2013. Retrieved 2 February 2020.*

90. *"sourceforge.net Competitive Analysis, Marketing Mix and Traffic" (https://www.alexa.com/siteinfo/sourceforge.net) . Alexa. Archived (https://web.archive.org/web/20201020111244/https://www.alexa.com/siteinfo/sourceforge.net) from the original on 20 October 2020. Retrieved 2 February 2020.*
91. *"bitbucket.org Competitive Analysis, Marketing Mix and Traffic" (https://web.archive.org/web/20170623162208/http://www.alexa.com/siteinfo/bitbucket.org) . Alexa. Archived from the original (https://www.alexa.com/siteinfo/bitbucket.org) on 23 June 2017. Retrieved 2 February 2020.*

92. *"gitlab.com Competitive Analysis, Marketing Mix and Traffic" (https://www.alexa.com/siteinfo/gitlab.com) . Alexa. Archived (https://web.archive.org/web/20171130123149/https://www.alexa.com/siteinfo/gitlab.com) from the original on 30 November 2017. Retrieved 2 February 2020.*

93. *"Eclipse Community Survey 2014 results | Ian Skerrett" (https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/) . Ianskerrett.wordpress.com. 23 June 2014. Archived (https://web.archive.org/web/20140625152145/http://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/) from the original on 25 June 2014. Retrieved 23 June 2014.*

94. *"Results of Eclipse Community Survey 2012" (https://eclipse.org/org/community_survey/Survey_Final_Results_2012.xls) . eclipse.org. Archived (https://web.archive.org/web/20160411135719/http://www.eclipse.org/org/community_survey/Survey_Final_Results_2012.xls) from the original on 11 April 2016.*

95. *"Compare Repositories – Open Hub" (https://www.openhub.net/repositories/compare) . Archived (https://web.archive.org/web/20140907051024/https://www.openhub.net/repositories/compare) from the original on 7 September 2014.*

96. *"Stack Overflow Annual Developer Survey" (https://insights.stackoverflow.com/survey) . Stack Exchange, Inc. Retrieved 9 January 2020. "Stack Overflow's annual Developer Survey is the largest and most comprehensive survey of people who code around the world. Each year, we field a survey covering everything from developers' favorite technologies to their job preferences. This year marks the ninth year we've published our annual Developer Survey results, and nearly 90,000 developers took the 20-minute survey earlier this year."*

97. *"Stack Overflow Developer Survey 2015" (https://web.archive.org/web/20190504144447/https://insights.stackoverflow.com/survey/2015#tech-sourcecontrol) . Stack Overflow. Archived from the original (https://insights.stackoverflow.com/survey/2015#tech-sourcecontrol) on 4 May 2019. Retrieved 29 May 2019.*
98. *"Stack Overflow Developer Survey 2017" (https://web.archive.org/web/20190529004901/https://insights.stackoverflow.com/survey/2017#work-_-version-control) . Stack Overflow. Archived from the original (https://insights.stackoverflow.com/survey/2017#work-_-version-control) on 29 May 2019. Retrieved 29 May 2019.*

99. *"Stack Overflow Developer Survey 2018" (https://web.archive.org/web/20190530142357/https://insights.stackoverflow.com/survey/2018/#work-_-version-control). Stack Overflow. Archived from the original (https://insights.stackoverflow.com/survey/2018#work-_-version-control) on 30 May 2019. Retrieved 29 May 2019.*

100. *"Stack Overflow Developer Survey 2022" (https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems). Stack Overflow. Retrieved 4 August 2022.*

101. *"Git (software) Jobs, Average Salary for Git Distributed Version Control System Skills" (https://www.itjobswatch.co.uk/jobs/uk/git%20%28software%29.do) . Itjobswatch.co.uk. Archived (https://web.archive.org/web/20161008072321/http://www.itjobswatch.co.uk/jobs/uk/git%20(software).do) from the original on 8 October 2016. Retrieved 30 September 2016.*

102. *"Team Foundation Server Jobs, Average Salary for Microsoft Team Foundation Server (TFS) Skills" (https://www.itjobswatch.co.uk/jobs/uk/team%20foundation%20server.do) . Itjobswatch.co.uk. Archived (https://web.archive.org/web/20161029185314/http://www.itjobswatch.co.uk/jobs/uk/team%20foundation%20server.do) from the original on 29 October 2016. Retrieved 30 September 2016.*

103. *"Subversion Jobs, Average Salary for Apache Subversion (SVN) Skills" (https://www.itjobswatch.co.uk/jobs/uk/subversion.do) . Itjobswatch.co.uk. Archived (https://web.archive.org/web/20161025011418/http://www.itjobswatch.co.uk/jobs/uk/subversion.do) from the original on 25 October 2016. Retrieved 30 September 2016.*

104. *"Mercurial Jobs, Average Salary for Mercurial Skills" (https://www.itjobswatch.co.uk/jobs/uk/mercurial.do) . Itjobswatch.co.uk. Archived (https://web.archive.org/web/20160923081538/http://www.itjobswatch.co.uk/jobs/uk/mercurial.do) from the original on 23 September 2016. Retrieved 30 September 2016.*

105. *"VSS/SourceSafe Jobs, Average Salary for Microsoft Visual SourceSafe (VSS) Skills" (https://www.itjobswatch.co.uk/jobs/uk/vss/sourcesafe.do) . Itjobswatch.co.uk. Archived (https://web.archive.org/web/20161029043610/http://www.itjobswatch.co.uk/jobs/uk/vss/sourcesafe.do) from the original on 29 October 2016. Retrieved 30 September 2016.*

106. *"Windows switch to Git almost complete: 8,500 commits and 1,760 builds each day" (https://arstechnica.com/information-technology/2017/05/90-of-windows-devs-now-using-git-creating-1760-windows-builds-per-day/) . Ars Technica. 24 May 2017. Archived (https://web.archive.org/web/20170524171707/https://arstechnica.com/information-technology/2017/05/90-of-windows-devs-now-using-git-creating-1760-windows-builds-per-day/) from the original on 24 May 2017. Retrieved 24 May 2017.*

107. *"git-init" (https://git-scm.com/docs/git-init) . Git. Archived (https://web.archive.org/web/20220315095632/https://git-scm.com/docs/git-init) from the original on 15 March 2022.*

108. *"Git – Branches in a Nutshell" (https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell#ch03-git-branching) . git-scm.com. Archived (https://web.archive.org/web/20201220123258/http://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell#ch03-git-branching) from the original on 20 December 2020. Retrieved 15 June 2020. "The "master" branch in Git is not a special branch. It is exactly like any other branch. The only reason nearly every repository has one is that the git init command creates it by default and most people don't bother to change it."*
109. *github/renaming (https://github.com/github/renaming) , GitHub, 4 December 2020, retrieved 4 December 2020*

110. *Default branch name for new repositories now main (https://about.gitlab.com/releases/2021/06/22/gitlab-14-0-released/#default-branch-name-for-new-repositories-now-main)* , GitLab, 22 June 2021, retrieved 22 June 2021
111. *"Git Revert | Atlassian Git Tutorial" (https://www.atlassian.com/git/tutorials/undoing-changes/git-revert)* . Atlassian. "Reverting has two important advantages over resetting. First, it doesn't change the project history, which makes it a "safe" operation for commits that have already been published to a shared repository."

112. *"Gitflow Workflow | Atlassian Git Tutorial" (https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow) . Atlassian. Retrieved 15 June 2020.*
113. *"Forking Workflow | Atlassian Git Tutorial" (https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow) . Atlassian. Retrieved 15 June 2020.*
114. *"Git repository access control" (https://wincent.com/wiki/git_repository_access_control) . Archived (https://web.archive.org/web/20160914114143/https://wincent.com/wiki/git_repository_access_control) from the original on 14 September 2016. Retrieved 6 September 2016.*

115. *Pettersen, Tim (20 December 2014). "Securing your Git server against CVE-2014-9390" (https://developer.atlassian.com/blog/2014/12/securing-your-git-server/) . Archived (https://web.archive.org/web/20141224012942/https://developer.atlassian.com/blog/2014/12/securing-your-git-server/) from the original on 24 December 2014. Retrieved 22 December 2014.*

116. *Hamano, J. C. (18 December 2014). "[Announce] Git v2.2.1 (and updates to older maintenance tracks)" (https://web.archive.org/web/20141219024646/http://article.gmane.org/gmane.linux.kernel/1853266) . Newsgroup: gmane.linux.kernel (news:gmane.linux.kernel) . Archived from the original (https://article.gmane.org/gmane.linux.kernel/1853266) on 19 December 2014. Retrieved 22 December 2014.*

117. *"CVE-2015-7545" (https://people.canonical.com/~ubuntu-security/cve/2015/CVE-2015-7545.html) . 15 December 2015. Archived (https://web.archive.org/web/20151226232616/http://people.canonical.com/~ubuntu-security/cve/2015/CVE-2015-7545.html) from the original on 26 December 2015. Retrieved 26 December 2015.*

118. *"Git 2.6.1" (https://github.com/git/git/commit/22f698cb188243b313e024d618283e0293e37140) . GitHub. 29 September 2015. Archived (https://web.archive.org/web/20160411135802/https://github.com/git/git/commit/22f698cb188243b313e024d618283e0293e37140) from the original on 11 April 2016. Retrieved 26 December 2015.*

119. *Blake Burkhart; et al. (5 October 2015). "Re: CVE Request: git" (http://seclists.org/oss-sec/2015/q4/67) . Archived (https://web.archive.org/web/20151227054727/http://seclists.org/oss-sec/2015/q4/67) from the original on 27 December 2015. Retrieved 26 December 2015.*

120. *"hash – How safe are signed git tags? Only as safe as SHA-1 or somehow safer?" (https://security.stackexchange.com/questions/67920/how-safe-are-signed-git-tags-only-as-safe-as-sha-1-or-somehow-safer) . Information Security Stack Exchange. 22 September 2014. Archived (https://web.archive.org/web/20160624232415/https://security.stackexchange.com/questions/67920/how-safe-are-signed-git-tags-only-as-safe-as-sha-1-or-somehow-safer) from the original on 24 June 2016.*

121. *"Why does Git use a cryptographic hash function?" (https://stackoverflow.com/questions/28792784/why-does-git-use-a-cryptographic-hash-function) . Stack Overflow. 1 March 2015. Archived (https://web.archive.org/web/20160701214638/http://stackoverflow.com/questions/28792784/why-does-git-use-a-cryptographic-hash-function) from the original on 1 July 2016.*

122. *"Git – hash-function-transition Documentation" (https://git-scm.com/docs/hash-function-transition/) . git-scm.com.*

External links

- Official website (https://git-scm.com)

Wikimedia Commons has media

- Git (https://www.openhub.net/p/git) at Open Hub

related to *Git*.

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

Retrieved from "https://en.wikipedia.org/w/index.php?title=Git&oldid=1158514077"
