WIKIPEDIA

Git

⚠ Dies ist eine alte Version dieser Seite, zuletzt bearbeitet am 13. März 2018 um 11:24 Uhr durch 80.146.228.123 (Diskussion) (→Unterstützte Betriebssysteme: Laut https://de.wikipedia.org/wiki/Haiku_(Betriebssystem) ist es KEIN unixartiges Betriebssystem.). Sie kann sich erheblich von der aktuellen Version unterscheiden.
(Unterschied) ← Nächstältere Version | Aktuelle Version (Unterschied) | Nächstjüngere Version → (Unterschied)

Git [gɪt] ist eine freie Software zur verteilten Versionsverwaltung von Dateien, die durch Linus Torvalds initiiert wurde.

Git

Entwickler	Junio C. Hamano, Shawn O. Pearce, Linus Torvalds u. v. a.
Erscheinungsjahr	2005
Aktuelle Version	2.17.1[1] (22. Mai 2018)
Betriebssystem	Linux, FreeBSD, macOS, Solaris u. a. unixoide; Windows; Haiku; …
Programmiersprache	C, Bourne-Shell, Perl
Kategorie	Versionsverwaltung
Lizenz	GNU GPLv2
deutschsprachig	ja[2]

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

Inhaltsverzeichnis

Geschichte

Durch eine Lizenzänderung des bis dahin genutzten, proprietären BitKeeper-Systems konnten die Linux-Kernel-Entwickler dieses nicht mehr kostenlos verwenden und somit blieb vielen Entwicklern der Zugang verwehrt. Daher begann Linus Torvalds im April 2005 mit der Entwicklung einer neuen Quellcode-Management-Software und präsentierte bereits wenige Tage nach deren Ankündigung eine erste Version.

Torvalds wünschte sich ein verteiltes System, das wie BitKeeper genutzt werden konnte und die folgenden Anforderungen erfüllte:

Altes Logo

1. Unterstützung verteilter, BitKeeper-ähnlicher Arbeitsabläufe
2. Sehr hohe Sicherheit gegen sowohl unbeabsichtigte als auch böswillige Verfälschung
3. Hohe Effizienz

Ein bereits existierendes Projekt namens Monotone entsprach den ersten zwei Anforderungen,[3] das dritte Kriterium wurde jedoch von keinem bestehenden System erfüllt.

Torvalds entschied sich dagegen, Monotone an seine Anforderungen anzupassen, und begann stattdessen, ein eigenes System zu entwickeln. Einer der Hauptgründe für diesen Schritt war die Arbeitsweise, für die Monotone nach Torvalds Ansicht optimiert ist. Torvalds argumentierte, dass einzelne Revisionen von einem anderen Entwickler in den eigenen Entwicklungszweig zu importieren zu Rosinenpickerei und unordentlichen Repositories führen würde. Wenn hingegen immer ganze Zweige importiert werden, wären Entwickler gezwungen aufzuräumen. Dazu seien Wegwerf-Zweige notwendig.

> “This is my only real conceptual gripe with ‘monotone’. I like the model, but they make it much harder than it should be to have throw-away trees due to the fact that they seem to be working on the assumption of ‘one database per developer’ rather than ‘one database per tree’. You don’t have to follow that model, but it seems to be what the setup is geared for, and together with their ‘branches’ it means that I think a monotone database easily gets very cruddy. The other problem with monotone is just performance right now, but that’s hopefully not *too* fundamental.”

> „Ich habe nur ein wirkliches konzeptionelles Problem mit ‚monotone‘: Ich mag die Arbeitsweise, aber sie erschwert die Nutzung von Wegwerf-Bäumen durch die scheinbare Arbeitsmethode ‚eine Datenbank je Entwickler‘ statt ‚eine Datenbank je Baum‘. Man muss zwar nicht diesem Modell folgen, aber das Setup scheint darauf ausgerichtet zu sein. Zusammen mit ihren ‚Zweigen‘ befürchte ich ein schnelles Verdrecken der monotone-Datenbank. Ein anderes, hoffentlich nicht *zu* grundlegendes Problem, ist die derzeitige Leistungsfähigkeit von monotone.“
>
> – Linus Torvalds[3]

Gits Gestaltung verwendet einige Ideen aus Monotone sowie BitKeeper, aber keinen Quellcode daraus. Es soll ausdrücklich ein eigenständiges Versionsverwaltungssystem sein.

Derzeitiger Maintainer von Git ist Junio Hamano.[4]

Name

Der Name „Git“ bedeutet in der britischen Umgangssprache so viel wie „Blödmann“. Linus Torvalds erklärte seine Wahl des ungewöhnlichen Namens mit einem Witz, sowie damit, dass das Wort praktikabel und in der Softwarewelt noch weitgehend unbenutzt war:

> "I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'Git'."

> „Ich bin ein egoistisches Arschloch und ich benenne all meine Projekte nach mir. Zuerst ‚Linux', jetzt eben ‚Git'."
>
> – Linus Torvalds[5]

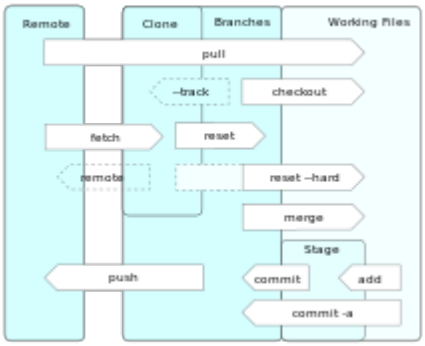
> "The joke 'I name all my projects for myself, first Linux, then git' was just too good to pass up. But it is also short, easy-to-say, and type on a standard keyboard. And reasonably unique and not any standard command, which is unusual."

> „Der Witz ‚Ich benenne alle meine Projekte nach mir, zuerst Linux, nun eben Git' war einfach zu gut, um ihn nicht zu machen. Aber es (der Befehl) ist auch kurz, einfach auszusprechen und zu schreiben auf einer Standardtastatur, dazu einigermaßen einzigartig und kein gewöhnliches Standardkommando – sehr ungewöhnlich."
>
> – Linus Torvalds[6]

Der Name *Linux* wurde anfangs nicht von Torvalds selbst propagiert und nur widerwillig akzeptiert.[7]

Eigenschaften

Git ist ein verteiltes Versionsverwaltungssystem, das sich in einigen Eigenschaften von typischen Versionskontrollsystemen unterscheidet:



Datenfluss

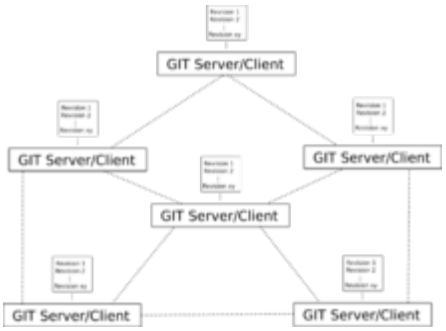
Nicht-lineare Entwicklung

Sowohl das Erstellen neuer Entwicklungszweige (*branching*), als auch das Verschmelzen zweier oder mehrerer Zweige (*merging*) sind integrale Bestandteile der Arbeit mit Git und fest in die Git-Werkzeuge eingebaut.[8] Git enthält Programme, mit deren Hilfe sich die nicht-lineare Geschichte eines Projektes einfach visualisieren lässt und mit deren Hilfe man in dieser Geschichte navigieren kann. Branches in Git sind (im Gegensatz zu anderen SCMs) sehr performant implementiert: Ein Branch stellt nur eine *Reference*, kurz *ref*, eine Textdatei mit einer Commit-ID, dar, die in einem Repository im Verzeichnis `.git/refs/heads` (z. B. `.git/refs/heads/master` für den immer vorhandenen *master*-Branch) liegt und auf einen bestimmten Commit verweist. Über dessen *Parental Commits*, also Eltern-Commits, lässt sich die Branch-Struktur rekonstruieren. Durch diese Eigenschaften lassen sich weiterhin sehr große und effiziente Entwicklungsstrukturen, wie bei Git selbst oder dem Linux-Kernel, realisieren, bei denen jedes Feature und jeder Entwickler einen Branch oder ein eigenes Repository haben, aus dem der Maintainer dann Commits über Merge oder Cherry-pick (Nutzen einzelner Commits) in den Hauptzweig des Projekts (master) übernehmen kann.

Entwicklungsgeschichte mit extensiv genutztem Branching und Merging

Kein zentraler Server

Jeder Benutzer besitzt eine lokale Kopie des gesamten Repositorys, inklusive der Versionsgeschichte (*history*). So können die meisten Aktionen lokal und ohne Netzwerkzugriff ausgeführt werden. Es wird nicht zwischen lokalen Entwicklungszweigen und Entwicklungszweigen entfernter Repositories unterschieden. Obwohl es keinen technischen Unterschied zwischen verschiedenen Repositories gibt (außer dem zwischen *normalen* und *bare*-Repositories auf Servern, bei denen kein *Working-Tree*, also die echten Dateien existiert), gilt die Kopie, auf die von einer Projekt-Homepage aus verwiesen wird, häufig als das offizielle Repository, in das die Revisionen der Entwickler übertragen werden. Es existieren spezielle *Remote-tracking branches*, das sind Referenzen (siehe *Nicht-lineare Entwicklung*), die auf den Stand eines anderen Repositorys zeigen.

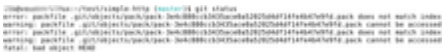


Dezentrale Verwaltung des gesamten Repositorys mit Hilfe von Git

Datentransfer zwischen Repositorys

Daten können neben dem Übertragen auf Dateisystemebene (`file://`) mit einer Reihe verschiedener Netzwerkprotokolle zwischen Repositorys übertragen werden. Git hat ein eigenes sehr effizientes Protokoll, das den TCP-Port 9418 nutzt (`git://`), allerdings nur zum *Fetchen* und *Clonen* genutzt werden kann, also dem Lesen eines Repositorys. Ebenso kann der Transfer über SSH (`ssh://`, das gängigste Protokoll für Schreiboperationen), HTTP (`http://`), HTTPS (`https://`) oder über (weniger effizient) FTP (`ftp://`) oder rsync (`rsync://`) erfolgen.[9] Die Übertragung in das offizielle Repository eines Projekts erfolgt häufig in Form von Patches, die via E-Mail an den Entwickler oder eine Entwicklungs-Mailing-Liste geschickt werden. Alternativ kann auch ein Review-System wie Gerrit verwendet werden.[10] Für Projekte, die auf Websites wie GitHub oder Bitbucket gehostet werden, kann eine Änderung einfach durch das *Pushen* eines Branches vorgeschlagen werden, der dann bei Bedarf durch ein *Merge* in das Projekt integriert wird.

Kryptographische Sicherheit der Projektgeschichte


Beschädigtes Objekt

Die Geschichte eines Projektes wird so gespeichert, dass der Hash einer beliebigen Revision *(commit)* auf der vollständigen Geschichte basiert, die zu dieser Revision geführt hat. Dadurch ist es nicht möglich, die Versionsgeschichte nachträglich zu manipulieren, ohne dass sich der Hash der Revision ändert. Einzelne Revisionen können zusätzlich markiert *(tagging)* und optional mit GPG digital signiert werden *(signed tag)*, beispielsweise um den Zustand zum Zeitpunkt der Veröffentlichung einer neuen Version der Software zu kennzeichnen.

Speichersystem und Dateiversionierung

Es gibt Versionsverwaltungssysteme (beispielsweise CVS), die für jede Datei (und jedes Verzeichnis) eigene, von allen anderen Dateien unabhängige Revisionsnummern verwalten. Für eine Datei wird nur dann eine neue Nummer erzeugt, wenn die jeweilige Datei Teil einer Commit-Operation ist. Im Gegensatz dazu weist Git bei jedem

Commit allen im Repository verwalteten Dateien (und Verzeichnissen) eine neue, für alle Dateien gleiche Revisionsnummer zu. Das Abspeichern selbst erfolgt, indem im Commit-Objekt ein Verweis auf die Projektwurzel als *tree*-Objekt gespeichert wird, das wiederum Verweise auf *blobs* (*binary large objects*, die reinen Inhalte der Dateien ohne Identifizierung) und weitere *trees* (Verzeichnisse) enthält. Ein *tree*-Objekt verweist (wie ein Verzeichnis-Inode) mit seinen Einträgen auf SHA1-Checksummen, die weitere *trees* und *blobs* identifizieren, ähnlich Inode-Nummern in Dateisystemen. Wenn eine Datei in einem Commit nicht geändert wird, ändert sich auch die Checksumme nicht und sie muss nicht nochmals gespeichert werden. Die Objekte liegen im Projekt unter `.git/objects`. Über Git-„Bordmittel“ lässt sich jeder beliebige Commit über den zugeordneten Hash (die Prüfsumme/Checksumme) eindeutig identifizieren, separat auslesen, verschmelzen oder gar als Abzweigungspunkt nutzen – vergleichbar mit den Revisionsnummern in anderen Systemen.

Säubern des Repositorys

Die Daten gelöschter und zurückgenommener Aktionen und Entwicklungszweige bleiben vorhanden (und können wiederhergestellt werden), bis sie explizit gelöscht werden.

Interoperabilität

Es gibt Hilfsprogramme, die Interoperabilität zwischen Git und anderen Versionskontrollsystemen herstellen. Solche Hilfsprogramme existieren unter anderem für GNU arch *(git-archimport)*, CVS *(git-cvsexportcommit*, *git-cvsimport* und *git-cvsserver)*, Darcs (*darcs-fastconvert*, *darcs2git* und andere), Quilt *(git-quiltimport)* und Subversion *(git-svn)*.

Web-Interface

Mit Gitweb gibt es eine in Perl geschriebene Weboberfläche. Der Team Foundation Server von Microsoft verfügt über eine Git-Anbindung (*Git-tf*).

Gitweb mit den Unterschieden zwischen zwei Commits

Verwendung

Die aktuelle Version wird produktiv für die Entwicklung vieler Open-Source-Projekte eingesetzt, darunter Amarok, Android, BusyBox, CMake, Debian, DragonFly BSD, Drupal, Eclipse, Erlang, Fedora, FlightGear, Git selbst, Gnome, Joomla, jQuery, JUnit, KDE, LibreOffice, LilyPond, Linux-Kernel, Linux Mint, MediaWiki, node.js, One Laptop per Child, OpenFOAM, OpenStreetMap, Perl 5, Parrot und Rakudo (Perl 6), PHP, phpBB[11], Plone, PostgreSQL, Python, Qt, Ruby on Rails, Ruby, Samba, Scala, TaskJuggler, TYPO3, VLC media player, Wine, x264[12] und X.org. Außerdem wird Git von einer Vielzahl weiterer Open-Source-Projekte genutzt, wie sie beispielsweise auf Plattformen wie GitHub, GitLab,[13] Gitorious oder BitBucket zu finden sind.

Git wird auch in kommerziellen Projekten verwendet. So gab beispielsweise Microsoft im August 2017 bekannt, die Versionsverwaltung von Windows auf Git umgestellt zu haben.[14]

Verwaltung von Inhalt

Obwohl Git primär zur Versionsverwaltung von Quellcode entwickelt wurde, wird es auch zum Speichern von flach strukturierten (im Gegensatz zu relationalen Strukturen) Datensätzen direkt als Datei genutzt. So können Funktionen wie Versionsverwaltung, Hook, diff, Replikation und Offline-Nutzung auch für Inhalte ohne Datenbank genutzt werden.[15] Die Nutzung ist ähnlich zu NoSQL, auch wenn Git keine Indexstruktur, Abfrage oder Gleichzeitigkeit erlaubt.

Der Einsatz erstreckt sich auch auf inhaltlich einfach strukturierte Systeme wie CMS[15] oder Wikis.[16][17]

Unterstützte Betriebssysteme

Unix/Linux

Git läuft auf fast allen modernen unixartigen Systemen wie Linux, Solaris, macOS, FreeBSD, DragonFly BSD, NetBSD, OpenBSD, AIX, IRIX.

Windows

Es gibt mehrere Portierungen von Git für Microsoft Windows:

- Git for Windows,[18] die Portierung des Git-Projektes (früher entwickelt unter dem Namen *msysGit*[19])
- TortoiseGit, eine Erweiterung für den Windows-Explorer (*Windows Shell Extension*). TortoiseGit benötigt zusätzlich eine Git-Installation wie z. B. *Git for Windows*.
- Die Cygwin-Umgebung enthält Git.

Siehe auch

- GitHub, GitLab, Bitbucket – Webbasierte Hosting-Dienste für Git-Projekte
- Source Code Control System (SCCS) – der POSIX-Standard für Versionsverwaltung
- Apache Subversion – zentrales Versionsverwaltungssystem
- Andere verteilte Versionsverwaltungssysteme:
  - Monotone
  - BitKeeper
  - Mercurial
  - Bazaar
  - GNU arch

Literatur

- Valentin Haenel, Julius Plenz: *Git – Verteilte Versionsverwaltung für Code und Dokumente*. Open Source Press, München 2011, ISBN 3-941841-42-4 (Online-Version (http://gitbu.ch/)).
- Sven Riedel: *Git – kurz & gut*. O'Reilly Verlag, Köln 2009, ISBN 3-89721-914-X.
- Jon Loeliger, Matthew McCullough: *Version Control with Git*. 2. Auflage. O'Reilly Media, Sebastopol 2012, ISBN 978-1-4493-1638-9 (eBook (https://it-ebooks.info/book/919/)).
- Scott Chacon: *Pro Git*. APress, New York 2009, ISBN 1-4302-1833-9.

Weblinks

Commons: Git (https://commons.wikimedia.org/wiki/Category:Git?uselang=de) – Sammlung von Bildern, Videos und Audiodateien

- Git Homepage (https://git-scm.com/) – offizielle Webpräsenz von Git
- Git documentation (https://git-scm.com/documentation) – Git-Dokumentation auf der Git-Website (englisch)
- Pro Git (https://git-scm.com/book/de/v1) – deutsche Übersetzung des Buches *Pro Git* auf der Git-Website
- Git for Windows (https://git-for-windows.github.io/) (englisch)
- *Git: a brief introduction* (https://www.youtube.com/watch?v=8dhZ9BXQgc4) auf YouTube mit Randal L. Schwartz, seit 2005 ein Wegbereiter von Git, am 12. Oktober 2007 (englisch)
- *Linus Torvalds on Git – Linus Torvalds über Git bei einem Google Tech Talk, 3. Mai 2007* (https://www.youtube.com/watch?v=4XpnKHJAok8) auf YouTube (englisch)

Einzelnachweise

1. *git.kernel.org*. (https://git.kernel.org/pub/scm/git/git.git/tag/?h=v2.17.1) (abgerufen am 2. Juni 2018).
2. *German translations for Git.* (https://github.com/git/git/blob/master/po/de.po) Abgerufen am 9. April 2015.
3. Linus Torvalds: *Kernel SCM saga..* (https://lkml.org/lkml/2005/4/6/121) In: *Linux-Kernel Archive.* 6. April 2005, abgerufen am 26. Februar 2017 (englisch).
4. *Vor 10 Jahren: Linus Torvalds baut Git.* (https://www.heise.de/developer/meldung/Vor-10-Jahren-Linus-Torvalds-baut-Git-2596654.html) In: *Heise online.* 8. April 2015, abgerufen am 17. Juli 2015.
5. *Git FAQ: Why the 'Git' name?* (https://git.wiki.kernel.org/index.php/GitFaq#Why_the_.27Git.27_name.3F) 9. März 2013, abgerufen am 12. Januar 2017 (englisch).
6. *Lord of the Files: How GitHub Tamed Free Software (And More).* (https://www.wired.com/2012/02/github-2/) 6. April 2005, abgerufen am 26. Februar 2017 (englisch).
7. Siehe dazu Geschichte von Linux #Der Name Linux
8. *Chapter 1. Repositories and Branches.* (https://www.kernel.org/pub/software/scm/git/docs/user-manual.html#repositories-and-branches) In: *Git User's Manual.* Abgerufen am 26. Februar 2017 (englisch).
9. *Exporting a git repository via http.* (https://www.kernel.org/pub/software/scm/git/docs/user-manual.html#exporting-via-http) In: *Git User's Manual.* Abgerufen am 26. Februar 2017 (englisch).
10. *Submitting patches to a project.* (https://www.kernel.org/pub/software/scm/git/docs/user-manual.html#submitting-patches) In: *Git User's Manual.* Abgerufen am 26. Februar 2017 (englisch).
11. *phpBB moves source code versioning from Subversion to Git.* (https://www.phpbb.com/community/viewtopic.php?f=14&t=2015905) 7. März 2010, abgerufen am 26. Februar 2017 (englisch).
12. *git.videolan.org Git – x264.git summary.* (https://git.videolan.org/gitweb.cgi?p=x264.git) Abgerufen am 26. Februar 2017 (englisch).
13. *Explore GitLab.* (https://gitlab.com/explore/projects) Abgerufen am 10. September 2014 (englisch).

14. Rainald Menge-Sonnentag: *Microsoft nutzt ab sofort Git zur Windows-Entwicklung. (https://heise.de/-3810273)* In: *Heise online*. 23. August 2017. Abgerufen am 25. August 2017.
15. Brandon Keepers: *Git: the NoSQL Database.* (https://speakerdeck.com/bkeepers/git-the-nosql-database) 21. April 2012, abgerufen am 9. April 2015 (englisch).
16. Adam Feber: *How we Moved 2.3 Million Wiki Pages to Git.* (https://blog.assembla.com/assemblablog/tabid/12618/bid/104945/How-we-Moved-2-3-Million-Wiki-Pages-to-Git.aspx) 4. Februar 2014, abgerufen am 26. Februar 2017 (englisch).
17. Wiki-Implementierung mit gollum/gollum. (https://github.com/gollum/gollum)
18. *Git for Windows verlässt Preview-Status.* (https://www.heise.de/newsticker/meldung/Git-for-Windows-verlaesst-Preview-Status-2785765.html) heise.de News, zuletzt abgerufen am 19. August 2015.
19. *Relationship to Git for Windows* (https://github.com/msysgit/msysgit/wiki/Relationship-to-Git-for-Windows)
