Git

aus Wikipedia, der freien Enzyklopädie

Git (dt. [gɪt]) ist eine freie Software zur verteilten Versionsverwaltung von Dateien, die ursprünglich für die Quellcode-Verwaltung des Kernels Linux entwickelt wurde.

Die Entwicklung von Git wurde im April 2005 von Linus Torvalds begonnen, um das bis dahin verwendete Versionskontrollsystem BitKeeper zu ersetzen, das durch eine Lizenzänderung vielen Entwicklern den Zugang verwehrte. Die erste Version erschien bereits wenige Tage nach der Ankündigung. Derzeitiger Maintainer von Git ist Junio Hamano.

Git läuft auf fast allen modernen UNIX-artigen Systemen, wie Linux, Solaris, Mac OS X, FreeBSD, DragonFly BSD, NetBSD, OpenBSD, AIX und IRIX. Auf Microsoft Windows läuft es mit Hilfe der Cygwin-Umgebung, mit Msysgit oder der TortoiseGit-Shell-Erweiterung (ähnlich TortoiseSVN).

Git

Entwickler	Junio C. Hamano, Shawn O. Pearce, Linus Torvalds und viele andere
Aktuelle Version	1.7.9.6 (2. April 2012)
Betriebssystem	Unixoide Systeme, Microsoft Windows
Programmier-sprache	C, Bourne-Shell, Perl
Kategorie	Versionsverwaltung
Lizenz	GNU General Public License
Deutschsprachig	Nein

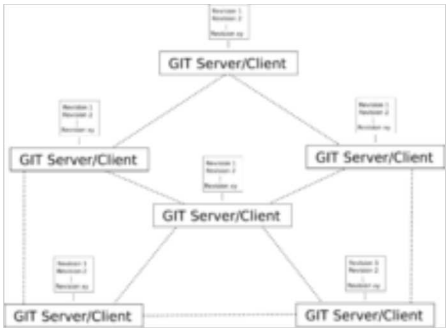
git-scm.com (http://git-scm.com/)

Inhaltsverzeichnis

Eigenschaften

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

Nicht-lineare Entwicklung



Dezentrale Verwaltung des gesamten Repositories mit Hilfe von GIT.


Korrumpiertes Objekt


Gitweb mit den Unterschieden zwischen zwei Commits

Sowohl das Erstellen neuer Entwicklungszweige (*branching*) als auch das Verschmelzen zweier oder mehrerer Zweige (*merging*) sind integraler Bestandteil der Arbeit mit Git, fest in die Git-Werkzeuge eingebaut und sehr performant.[1] 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 Elterncommits, lässt sich die Branchstruktur 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.

Kein zentraler Server

Jeder Benutzer besitzt eine lokale Kopie des gesamten Repositories, 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*-Repositorys 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.

Datentransfer zwischen Repositories

Daten können mit einer Reihe verschiedener Protokolle zwischen Repositories übertragen werden. Git besitzt ein eigenes Protokoll, das den TCP-Port 9418 nutzt. Ebenso kann der Transfer über SSH oder (weniger effizient) über HTTP, HTTPS, FTP oder rsync erfolgen.[2] 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.[3]

Kryptographische Sicherheit der Projektgeschichte

Die Geschichte eines Projektes wird so gespeichert, dass der Name 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 Name der Revision ändert. Einzelne Revisionen können zusätzlich markiert und mit GPG digital signiert werden (*tagging*), beispielsweise um den Zustand zum Zeitpunkt der Veröffentlichung einer neuen Version der Software zu kennzeichnen.

Säubern des Repositories

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 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 existiert eine in Perl geschriebene Weboberfläche.

Geschichte

Die Git-Entwicklung wurde notwendig, nachdem die Linux-Kernel-Entwickler gezwungen waren, die kostenlose Verwendung des proprietären BitKeeper-Systems aufzugeben.

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

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[4], 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 – Git – zu schreiben. 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, würden 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."
>
> – LINUS TORVALDS[4]

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

Verwendung

Die derzeit aktuelle Version wird produktiv für die Entwicklung vieler Open-Source-Projekte eingesetzt, darunter Amarok, Android, BusyBox, Debian, DragonFly BSD, Eclipse, Erlang, Fedora, Git selbst, Gnome, jQuery, JUnit, KDE, LibreOffice, LilyPond, Linux Kernel, Linux Mint, Mediawiki, node.js, One Laptop per Child, OpenFOAM, Perl 5, Parrot und Rakudo (Perl 6), PHP, phpBB,[5] PostgreSQL, Qt, Ruby on Rails, Ruby, Samba, TaskJuggler, VLC media player, Wine, x264,[6], X.org und Drupal. Außerdem wird Git von vielen kleineren Hobby-Projekten genutzt, wie sie beispielsweise auf GitHub zu finden sind.

Zum Namen „Git“

Der Name „Git“ bedeutet in der britischen Umgangssprache soviel 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.“
> – LINUS TORVALDS [7]

> „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.“
> – LINUS TORVALDS [8]

Siehe auch

- GitHub – Webbasierter Hosting-Dienst für Git-Projekte
- Source Code Control System (SCCS) – Der POSIX-Standard für Versionsverwaltung
- Monotone – Verteiltes Versionsverwaltungssystem
- BitKeeper – Verteiltes Versionsverwaltungssystem
- Mercurial – Verteiltes Versionsverwaltungssystem
- Bazaar – Verteiltes Versionsverwaltungssystem
- Apache Subversion – Zentrales Versionsverwaltungssystem

Literatur

- Valentin Haenel, Julius Plenz: *Git – Verteilte Versionsverwaltung für Code und Dokumente*. Open Source Press, 2011, ISBN 978-3-941841-42-0.
- Sven Riedel: *Git – kurz & gut*. O’Reilly, 2009, ISBN 978-3-89721-914-4.
- Scott Chacon: *Pro Git*. APress, 2009, ISBN 978-1-4302-1833-3.

Weblinks

- Git Homepage (http://git-scm.com) – Offizielle Webpräsenz von Git
- Git documentation (http://git-scm.com/documentation) – Git-Dokumentation auf der Git-Homepage (englisch)
- Detaillierte und sortierte Git-Dokumentation (http://book.git-scm.com/index.html) (englisch)
- Warum Git besser als X ist (http://de.whygitisbetterthanx.com) – Vorteile von Git im Vergleich zu anderen Versionskontrollsystemen
- Git (http://www.youtube.com/watch?v=8dhZ9BXQgc4) – Randal Schwartz über Git, 12. Oktober 2007 (englisch, Flash Video)
- Linus Torvalds on Git (http://www.youtube.com/watch?v=4XpnKHJAok8#) – Linus Torvalds über Git bei einem Google Tech Talk, 3. Mai 2007 (englisch, YouTube Video)

- Befehlsreferenz (http://weinimo.de/Git-Hilfen) – Für die wichtigsten Anwendungsfälle (deutsch, Wiki)
- Verteilte Versionskontrollsysteme (http://chaosradio.ccc.de/cre130.html) – Chaosradio Express über Verteilte Versionskontrollsysteme und GIT im Besonderen
- Themenschwerpunkt Git (http://www.thomas-krenn.com/de/wiki/Kategorie:Git) – Anleitungen zum Arbeiten mit Git (deutsch, Thomas Krenn Wiki)

Einzelnachweise

1. http://www.kernel.org/pub/software/scm/git/docs/user-manual.html#repositories-and-branches
2. http://www.kernel.org/pub/software/scm/git/docs/user-manual.html#exporting-via-http
3. http://www.kernel.org/pub/software/scm/git/docs/user-manual.html#submitting-patches
4. Linus Torvalds: *Kernel SCM saga.*. (http://www.uwsg.iu.edu/hypermail/linux/kernel/0504.0/1540.html) In: *Linux-Kernel Archive*. 6. April 2005, abgerufen am 13. September 2011 (englisch).
5. *phpBB moves source code versioning from Subversion to Git*. (http://www.phpbb.com/community/viewtopic.php?f=14&t=2015905) 7. März 2010, abgerufen am 13. September 2011 (englisch).
6. *git.videolan.org Git - x264.git summary*. (http://git.videolan.org/gitweb.cgi?p=x264.git) Abgerufen am 13. September 2011 (englisch).
7. https://git.wiki.kernel.org/articles/g/i/t/GitFaq_ebc3.html#Why_the_.27git.27_name.3F
8. *Lord of the Files: How GitHub Tamed Free Software (And More)*. (http://www.wired.com/wiredenterprise/2012/02/github/) 6. April 2005, abgerufen am 24. Februar 2012 (englisch).

Kategorien: Freie Versionsverwaltungssoftware | Repository | Mac-OS-Software | Linux-Software | Unix-Software | BSD-Software | Solaris-Software

---