Git

Ein Open Source Versionskontrollsystem

Dennis Blöte, Oktober 2007

Servertechnologien, Hannes Wölfler

WS 2007/08, Internettechnik, FH JOANNEUM

Inhalt

Was ist Git?

Git[1] ist ein von Versionskontrollsystem, welches 2005 von Linus Torvalds entwickelt wurde, um es zur Quellcodeverwaltung des Linux-Kernels einzusetzen.[2] Die Neuentwicklung wurde begonnen, da das zuvor genutzte Versionskontrollsystem BitKeeper[3] aus Gründen der Lizenzänderung[4] für diesen Zweck unbrauchbar wurde.

Bei Git handelt es sich um ein Open Source Projekt, welches unter der GNU General Public License Version 2[5] steht und zur Zeit aktiv weiterentwickelt wird (Version 1.5.3, Stand: 18.10.2007). Es unterstützt alle gängigen UNIX-Systeme und läuft mit Hilfe der Cygwin-Umgebung[6] auch unter Windows.

Als Referenz steht die sehr gute offizielle Dokumentation[7] bereit und zur Installation und Einarbeitung gibt es mehre englischsprachige Anleitungen[8]. Neben einer inoffiziellen deutschsprachigen Einführung[9] sind bisher nur sehr wenige Quellen auf Deutsch verfügbar und die Kommunikation innerhalb der Community wird über die englisch-sprachige Mailingliste und IRC geführt[10].

Was ist Versionskontrolle?

Versionskontrollsysteme werden in der Softwareentwicklung eingesetzt, um verschiedene Entwicklungsstände eines Projekts zu sichern und diese gegebenenfalls wiederherstellen zu können. Dies eignet sich nicht nur, um Sicherungskopien der einzelnen Fortschritte zu haben, sondern auch zur verteilten Entwicklung einer Software: Arbeiten beispielsweise mehrere Entwickler an einem Projekt, so dient das Versionskontrollsystem als zentrales Quellcodelager (Repository), aus welchem die Entwickler den aktuellen Stand des Projekt beziehen (Check out) und in welches sie ihre Änderungen anschließend wieder einspielen (Check in).

Da Versionskontrolle insbesondere der Unterstützung von verteilter Entwicklung dient, ist eines der zentralen Features das Zusammenführen (Merging) von Codeteilen. Arbeiten zum Beispiel zwei Personen an der gleichen Datei, so sind die meisten Systeme in der Lage, die jeweiligen Änderungen zu erkennen und im Repository zusammenzuführen. Sind die verschiedenen Stände nicht automatisch zusammenzuführen, weil beispielsweise

1 Git – Fast Version Control System, http://git.or.cz/
2 Wikipedia: Git, http://de.wikipedia.org/wiki/Git
3 Bitkeeper – The Scalable Distributed Software Configuration Management System, http://www.bitkeeper.com/
4 No More Free BitKeeper, http://kerneltrap.org/node/4966
5 GNU General Public License Version 2, http://www.gnu.org/licenses/gpl2.txt
6 Cygwin Information and Installation, http://www.cygwin.com/
7 Git(7) Manual Page, http://www.kernel.org/pub/software/scm/git/docs/
8 Introduction to Git, http://download.ikaaro.org/doc/itools/chapter-git.html
9 Git Tutorial, http://www.online-tutorials.net/programmierung/git/tutorials-t-3-263.html
10 Git – Community and Development, http://git.or.cz/#community

beide Entwickler die gleichen Codezeilen bearbeitet haben, so zeigt das System beim Einchecken einen Konflikt an, welcher manuell behoben werden muss.
Darüber hinaus bieten die meisten Versionsverwaltungssysteme Features wie Tags und Branches: Mit Tags lassen sich bestimmte Entwicklungsstände, beispielsweise ausgelieferte Releases kennzeichnen, um diese zu einem späteren Zeitpunkt wiederherzustellen. Branching bezeichnet die Aufspaltung in verschiedene Entwicklungszweige: So kann nach einem Release ein Branch für Bugfixes der ausgelieferten Version angelegt werden, während ein anderer Zweig zur Entwicklung der kommenden Version genutzt wird.

Was sind die Besonderheiten von Git?

Neben Git gibt es eine Reihe weiterer Versionskontrollsysteme, zu den populärsten darunter zählen CVS[11] und sein Nachfolger SVN[12]. Da Git speziell zur Verwaltung des Kernel-Quellcodes von Linux entwickelt wurde, ergeben sich einige spezifische Unterschiede zu anderen Systemen[13].
Effizienz und *Geschwindigkeit* sind die primären Anforderungen, die sich aus der großen Masse an Quellcode ergeben, mit der das System konfrontiert wird. Git skaliert und ist vergleichsweise schnell[14] im Umgang mit großen Projekten und langen Projektgeschichten. Da Git vor allem von *nicht-linearer Entwicklung* ausgeht, kommt diese Effizienz insbesondere beim Branching und Merging zum tragen. Darüber hinaus enthält Git Tools zur Visualisierung und zur Navigation durch verschiedene Entwicklungszweige.
Im Gegensatz zu anderen Versionskontrollsystemen gibt es bei Git keinen zentralen Projektserver. Jeder Entwickler läd das komplette Projekt (inklusive Historie) in seine lokale Umgebung (Working Copy) und es findet keine Unterscheidung zwischen lokalen und entfernten Entwicklungszweigen statt. Durch diese *Dezentralität* wird zusätzlich die Ausfallsicherheit des Entwicklungsverlaufs gewährleistet.
Ein weiterer Unterschied liegt in der Unterstützung von *Code-Reviews*: Es gibt die Möglichkeit, alle Änderungen (Patches) zunächst von einem Teammitglied (Maintainer) überprüfen zu lassen, bevor sie in das Repository eingecheckt werden können. Dieser Arbeitsablauf wird durch E-Mail unterstützt, so das Patches per Mail verschickt und anschließend aus der Mailbox des Maintainers integriert werden können[15].

---

11 CVS – Concurrent Versions System, http://www.nongnu.org/cvs/
12 SVN – Subversion, http://subversion.tigris.org/
13 About Git, http://git.or.cz/#about
14 Git Benchmarks, http://git.or.cz/gitwiki/GitBenchmarks
15 Git Tutorial, http://www.online-tutorials.net/programmierung/git/tutorials-t-3-263.html

Installation

Die folgende Installationsanleitung bezieht sich auf Git ab Version 1.5 unter Ubuntu. Git kann über den Paketmanager Aptitude bezogen werden und ist daher recht einfach in folgenden Schritten zu installieren:

1. Um die aktuelle Git-Verson zu installieren, müssen zuerst die Backports als Installationsressourcen zu `/etc/apt/sources.list` hinzugefügt werden. Dazu sind die zwei Zeilen, welche den Verweis auf die (feisty-)backports enthalten auszukommentieren
2. **`#: apt-get update`** ausführen, um die Paketliste zu aktualisieren
3. Jetzt können die für Git benötigten Pakete installiert werden:
   `#: apt-get install git-core git-doc git-email gitk curl`
4. Mit folgendem Befehl lässt sich feststellen, ob die richtige Version installiert ist:
   `#: git --version` (soll > 1.5 sein)

Konfiguration

1. Es müssen der Name und die E-Mail-Adresse des Entwickler angegeben werden:
   `#: git-config --global user.name „Dennis Bloete“`
   `#: git-config --global user.email mail@dennisbloete.de`
2. Zusätzlich lassen sich weitere Einstellungen wie farbige Ausgabe der Verzeichnis- und Dateilisten im Terminal vornehmen:
   `#: git-config --global color.branch auto`
   `#: git-config --global color.diff auto`
   `#: git-config --global color.status auto`
3. Sollen Patches per E-Mail gesendet werden, muss ein SMTP-Server angegeben werden:
   `#: git-config --global sendemail.smtpserver smtp.domain.de`
4. Die komplette Liste der Konfigurationsvariablen erhält man mit
   `#: git-config --help`
5. Die Konfiguration wird in der Datei `.gitconfig` im Home-Verzeichnis des Benutzers abgelegt, wo sie sich überprüfen und manuell editieren lässt:

```
[user]
name = Dennis Bloete
email = mail@dennisbloete.de
...
```

Erste Schritte

Nach der Konfiguration kann entweder ein neues Repository angelegt oder ein bestehendes importiert werden. Mit folgendem Befehl lässt sich das Repository eines bestehenden Projekts übernehmen:

```
#: git clone http://url-des-repository.de
```

Zur besseren Veranschaulichung legen wir ein neues Projekt an und starten von vorne:

```
#: mkdir project
#: cd project
#: git init
-> Initialized empty Git repository in .git/
```

Anschließend können erste Dateien erstellt und eingecheckt werden:

```
#: echo „Erste Datei“ > file1
#: echo „Zweite Datei“ > file2
#: git add file1 file2
#: git commit -m „Zwei Dateien hinzugefügt“
```

Der Befehl `add` fügt die Dateien zunächst dem Index, welcher vergleichbar mit der Working Copy anderer Systeme ist, hinzu. Erst nach dem `commit` befinden sich die Dateien im Repository und somit unter Versionkontrolle. Ändern wir beispielsweise den Inhalt von `file1`, so können wir uns die genauen Unterschiede zur Version im Repository anzeigen lassen:

```
#: git diff
```

Einen einfachen Überblick aller Änderungen bekommt man über die Status-Abfrage:

```
#: git status
```

Will man die komplette Historie eines Projekts oder einer Datei (optional den Dateinamen mit angeben) sehen, so kann man sich das Log anzeigen lassen:

```
#: git log [dateiname]
```

Eine Liste aller Kommandos lässt sich mit dem Befehl **`git`** ausgeben. Die am häufigsten verwendeten Kommandos und ihre Paramater finden sich im Cheat Sheet.

Branches

Mit folgendem Befehl lässt sich ein neuer Entwicklungszweig anlegen:

```
#: git branch my_branch
```

Um zu überprüfen, in welchem Entwicklungszweig man sich befindet, kann man den Befehl `git branch` ohne Angabe eines Namens nutzen:

```
#: git branch
-> * master
     my_branch
```

Wir man sieht, befinden wir uns aktuell noch im Branch `master`, dem Standard-entwicklungszweig. Um in den neu erstellten Branch zu wechseln, müssen wir diesen auschecken:

```
#: git checkout my_branch
-> Switched to branch „my_branch“
```

In `my_branch` können jetzt Änderungen vorgenommen werden, ohne dass sie den Haupt-entwicklungszweig betreffen, so dass hier jetzt beispielsweise Bugfixes eingebaut werden, während in `master` schon neue Features implementiert werden.
Dies wird ersichtlich, wenn wir in `my_branch` eine Datei hinzufügen und anschließend wieder zurück in `master` wechseln:

```
#: echo „Eine neue Datei in my_branch“ > neu
#: git add neu
#: git commit -m „Neue Datei hinzugefügt“
```

Vor dem Wechseln in einen anderen Entwicklungszweig muss der Stand des aktuellen Branch durch einen Commit gespeichert werden. Wechseln wir nun in `master` stellen wir fest, dass die in `my_branch` erstellte Datei `neu` dort noch nicht vorhanden ist:

```
#: git checkout master
-> Switched to branch „master“
#: ls
-> file1  file2
```

Um das zu Erreichen, müssen die beiden Branches zusammengeführt werden.

Zur Vereinigung zweier Entwicklungszweige wird das `pull` Kommando verwendet:

```
#: git pull . my_branch
```

Der Punkt dient als Pfadangabe, das Kommando führt `my_branch` und `master` im aktuellen Verzeichnis zusammen. Wurde in beiden Zweigen die gleiche Datei bearbeitet, versucht Git die jeweiligen Abweichungen zu erkennen und die Änderungen aus `my_branch` in die Datei in `master` einzufügen. Ist dies nicht ohne weiteres möglich, zum Beispiel weil in beiden Versionen die gleiche Zeile verändert wurde, zeigt Git einen Konflikt an, welcher manuell gelöst werden muss. Die betroffene Datei kann dann in einem Editor geöffnet werden und enthält an den jeweiligen Konfliktstellen die Zeilen aus beiden Versionen:

```
<<<<<<< HEAD:file1
Eine in master geänderte Zeile in file1
=======
Änderung in der gleichen Zeile aus my_branch
>>>>>>> iz3j1he1j2retrez2nm12n3j1h2123mnmnm21:file1
```

Sind die etwaigen Konflikte behoben, kann der zusammengeführte Stand commited werden:

```
#: git commit -a -m „my_branch und master zusammengeführt“
```

Von einem zentralen Server aus arbeiten[16]

Arbeitet man mit mehreren Entwicklern an einem Projekt, empfiehlt es sich, ein zentrales Repository zu haben, aus welchem die einzelnen Personen ihren Stand beziehen und in dem sie ihre Fortschritte für die anderen wieder zur Verfügung stellen können.
Dazu kann man entweder ein neues Repository angelegt werden

```
#: mkdir project
#: cd project
#: git init --shared
#: mv .git project.git
```

oder ein bestehendes geklont:

```
#: git clone .git /neuer/pfad/project.git --bare
```

Das so entstandene Verzeichnis `project.git` kann anschließend auf einen Server geladen werden, auf den alle am Projekt beteiligten Personen Zugriff haben. Über die Vergabe von Zugriffsrechten lassen sich detailierte Einstellungen vornehmen, wer zu welchen Operationen berechtigt ist. So können beispielsweise alle Benutzer der Gruppe `git` hinzugefügt werden, welche Schreibrechte auf das Repository besitzt und einzelne Verzeichnisse nur von bestimmten Personen beschreibbar gemacht werden.
Das ist zum Beispiel dann hilfreich, wenn es einen Maintainer geben soll, welcher den Hauptentwicklungszweig `master` überwacht und alle Branches kontrolliert, bevor er sie mit `master` zusammenführt.

Jeder Person hat so lokal sein eigenes Repository, in dem sie entwickelt und die Stände anschließend in das zentrale Repository einspielt. „Bei der gemeinsamen Entwicklung kann es häufig vorkommen, dass der Entwicklungsstand auf dem Server neuer ist als derjenige, an dem man gerade arbeitet. In solchen Situationen empfiehlt es sich, zuerst ein `git fetch` auszuführen, damit der origin/*-Zweig aktualisiert wird. Danach wird die eigene Entwicklung mit `git rebase origin/master master` dem aktuellen Stand angepasst.“[17]

16 A tutorial introduction to Git, http://www.kernel.org/pub/software/scm/git/docs/tutorial.html
17 Git Tutorial, http://www.online-tutorials.net/programmierung/git/tutorials-t-3-263.html

Git Cheat Sheet

Im folgenden finden sich die wichtigsten Git-Kommandos und ihre Entsprechung in SVN[18].

Aufgabe	Git	SVN

Repository anlegen	`git init`
	`git add .`
	`git commit`
		`svnadmin create` *`repo`*
		`svn import` *`file://repo`*

Repository laden	`git clone` *`url`*	`svn checkout` *`url`*
Datei hinzufügen	`git add` *`file`*	`svn add` *`file`*
Datei verschieben	`git mv` *`file`*	`svn mv` *`file`*
Datei löschen	`git rm` *`file`*	`svn rm` *`file`*
Committen	`git commit -a -m „Text“`	`svn ci -m „Text“`
Stand remote einchecken	`git push` *`remote`*	`svn ci -m „Text“`
Aktuellen Stand laden	`git pull`	`svn update`
Älteren Stand laden	`git checkout` *`rev`* `git checkout` *`prevbranch`*	`svn update -r` *`rev`* `svn update`
Diff anzeigen	`git diff`	`svn diff`
Änderungen anzeigen	`git status`	`svn status`
Zur letzten Revision zurückkehren	`git checkout` *`path`*	`svn revert` *`path`*
Tag erstellen	`git tag -a` *`name`*	`svn copy http://svn.de/trunk` `http://svn.de/tags/`*`name`*

Branch erstellen	`git branch` *`branch`*
	`git checkout` *`branch`*
		`svn copy http://svn.de/trunk`
		`http://svn.de/branches/`*`branch`*
		`svn switch`
		`http://svn.de/branches/`*`branch`*

Liste aller Branches	`git branch`	`svn list http://svn.de/branches/`
Branches mergen	`git merge` *`branch`*	`svn merge -r` *`rev`*`:HEAD` `http://svn.de/branches/`*`branch`*
Historie anzeigen	`git log`	`svn log`
Patch ausführen	`git apply`	`patch -p0`

Eine ausführliche Beschreibung dieser und weiterer Kommandos findet sich in der Git-Referenz unter „Everyday GIT with 20 commands or so“[19]. Die komplette Liste aller Kommandos ist im Git-Manual[20] einsehbar.

18 Git – SVN Crash Course, http://git.or.cz/course/svn.html
19 Everyday GIT with 20 commands or so, http://www.kernel.org/pub/software/scm/git/docs/everyday.html
20 Git Manual, http://www.kernel.org/pub/software/scm/git/docs/