Git

DIE VERBREITETSTE VERSIONSVERWALTUNGSSOFTWARE

PHILIPP RECKNAGEL

Inhaltsverzeichnis

Geschichte

Bis zum 30. Juni 2005 besaß die Versionskontrollsoftware „*BitKeeper*“ eine Community Version, welche das Linux-Entwicklerteam kostenlos verwenden konnte. Bereits im April des besagten Jahres wurde die freie Lizenz zurückgezogen und die Software wurde kostenpflichtig.

Aufgrund dieser Änderung entschied sich Linus Torvalds dazu, eine eigene Software zu erstellen welche ähnlich wie „*BitKeeper*“ ausfallen sollte und ebenfalls Verbesserungen beinhaltet. Torvalds‘ Voraussetzungen waren Abläufe ähnlich wie bei „*BitKeeper*“, hohe Sicherheit gegen böswillige und unbeabsichtigte Änderungen sowie eine hohe Effizienz.

Ein bereits existierendes Projekt namens „*Monotone*“ erfüllte bereits die ersten beiden Anforderungen, doch die hohe Effizienz wurde weder von „*Monotone*“ noch von BitKeeper erreicht.

> „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.“
>
> (Torvalds, Re: Kernel SCM saga.., 2005)

In diesem Schreiben meint Torvalds, dass aufgrund des Datenbankkonzepts die Datenbank von „*Monotone*“ sehr schnell verdrecken würde. Die Software verwendet im Gegensatz zu Torvalds‘ Wünschen eine Daten pro Entwickler anstatt eine Datenbank pro Branch. Ebenfalls beschwert er sich über die Leistungsfähigkeit der Software.

Nachdem Torvalds sein neues Projekt ankündigte, präsentierte er bereits wenige Tage nachher eine erste Version von Git. Bei der Programmierung von Git verwendete Torvalds ausschließlich Ideen von „*BitKeeper*“ und „*Monotone*“ zu verwenden und keinen Quellcode dieser.

Der Name Git stammt vom britischen Wort „Git“ welches nicht weniger als Blödmann, Depp, Idiot oder Dummkopf bedeutet. Begründet wird diese Entscheidung von Torvalds durch einen Witz.

> “I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'Git'.”
> (Torvalds, kein Datum)

Alternativ meint Torvalds aber auch, dass man Git auch anders interpretieren kann:

- 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 light suddenly fills the room.
- "Goddamn idiotic truckload of sh*t": when it breaks

Dezentralisierung

Git benötigt keinen Server, um zu funktionieren. Git speichert diese Daten im Repository, einem lokalen Ordner auf dem Rechner des Projektes. Repositories können sich jedoch auf einem externen Server befinden, mit welchen man sein lokales Repository synchronisieren kann.

Lokale Funktionen von Git

Commit

Commits sind Zustände des Repository zu einem bestimmten Zeitpunkt. Man kann Commits rückgängig machen. Auch sind Commits nur lokal gespeichert und haben nur Einfluss auf das lokale Repository.

Status

Der Status eines Repository ist eine Liste, welche die Änderungen beinhalten, welche bei dem nächsten Commit durchgeführt werden.

Git-Funktionen für Remote-Repositories

Push

Bei einem Push werden die lokalen Änderungen am Remote-Repository angewandt und dort übernommen. Der Push ist sozusagen die Remote-Version des Commits.

Pull

Pull fragt die letzten Änderungen vom Remote-Repository ab und wendet sie auf das lokale Repository an. Die lokalen Dateien werden hier auf den aktuellen Stand des Remote-Repository aktualisiert.

Clone

Um an erster Stelle die Dateien des Remote-Repository in einen lokalen Ordner zu speichern, muss das Remote-Repository geklont werden. Das Remote-Repository wird hier vom Server heruntergeladen und in einem vorgesehenen Ordner abgelegt.

Branches

Branches sind Kopien des Stammprojektes. Diese Abzweigungen sind isoliert und Änderungen an einer Branch haben keinen Einfluss auf andere. Als best-practice wird das Erstellen von Branches gesehen, wenn man an einem neuen Feature arbeitet, da man dann nicht den funktionierenden Hauptcode gefährdet.

Branch-Merge

Der Merge von Branches ist das Verschmelzen von zwei Zweigen. Dies geschieht meist, wenn man den Code aus einer Entwicklungs-Branch in den Hauptzweig bringen möchte.

Git-Bash Befehle

Konfigurieren von Git

git config –global user.name “[NAME]”	Festlegen des lokalen Nutzernamen
git config --global user.email [EMAIL]	Festlegen der E-Mail-Adresse des Nutzers

Lokale Befehle

git init	Erstellt das Repository in dem Ordner in dem sich die Bash befindet
git status	Inhalt des nächsten Commits anzeigen
git add [DATEI]	Fügt Datei mit angegebenen Namen im Ordner der Bash ins Repository hinzu
git add -a	Fügt alle Dateien im Ordner der Bash in das Repository hinzu
git commit -m "[KOMMENTAR]"	Commitet alle Änderungen mit dem Angegebenen Kommentar

Remote-Repository Befehle

git remote add [NAME] [URL]	Verlinken eines Remote-Repository mit dem lokalen Repository
git push [NAME] [BRANCH]	Pusht die lokalen Änderungen in die Angegebene Branch des Remote-Repository
git pull [NAME] [BRANCH]	Pullt die letzten Änderungen vom Remote-Repository und aktualisiert das lokale Repository
git clone [URL]	Klont das Remote-Repository in den lokalen Ordner

Branch-Befehle

git branch [NAME]	Erstellen einer Brach mit dem gegebenen Namen
git checkout [NAME]	Wechsel von aktueller Branch in die gegebene Brach
git merge [NAME]	Verschmilzt den gegebenen Zweig mit der Branch in welcher sich die Bash aktuell befindet
git branch -d [NAME]	Löschen der gegebenen Branch

Sonstige Befehle

git log	Zeigt Liste aller Commits und deren IDs an
git show [ID]	Zeigt Commit mit der gegebenen ID an
git checkout [ID] [DATEI]	Setzt die gegebene Datei auf den Stand des gegebenen Commits zurück
git revert HEAD	Macht den letzten Commit rückgängig
git revert [ID]	Macht gegebenen Commit rückgängig