Articlename: git
Keywords: git revision control system
Date: 19.06.2012, 16:05
Views: 112105
Categoryname: Programmierung

Vorwort

git ist das Revision Control System von Linus Torvalds. Es entstand aus dem BedÃ¼rfnis heraus, ein geeignetes RCS fÃ¼r die Entwicklung des Linux Kernels zu finden.
Den verfÃ¼gbaren Opensource-RCS mangelte es entweder an Performance oder an FlexibilitÃ¤t, weshalb Linus die Entwicklung eines neuen, verteilten Systems begann.

Doch zunÃ¤chst, was ist ein RCS und wofÃ¼r wird es benÃ¶tigt?
Bei der Entwicklung grÃ¶ÃŸerer Software-Projekte ist es von Vorteil, wenn mehrere EntwicklungsstÃ¤nde gesichert und jederzeit wiederhergestellt werden kÃ¶nnen.
ZusÃ¤tzlich kÃ¶nnen mehrere Entwicklungszweige gleichzeitig bestehen, die dann schlieÃŸlich wieder in einen Entwicklungsbaum zusammengefÃ¼hrt werden mÃ¼ssen.
Weitere BedÃ¼rfnisse nach einem RCS ergeben sich automatisch, wenn mehrere Programmierer gleichzeitig an einem Projekt beteiligt sind.

Dieses Tutorial soll nur eine kurze EinfÃ¼hrung beim Umgang mit git sein und ersetzt nicht das Lesen der Manpages.

Wieso git?

git wurde in erster Linie als RCS fÃ¼r den Linux-Kernel entwickelt. Dabei ergeben sich mehrere wichtige Anforderungen:

- es muss sehr schnell sein, um mit der groÃŸen Code-Masse klarzukommen
- nicht jeder darf Patches integrieren, sie mÃ¼ssen vorher durch einen Maintainer kontrolliert werden
- das System muss dezentral arbeiten, um einerseits Bandbreite und Performance zu sparen und damit andererseits nicht bei einem Ausfall die ganze Entwicklung still steht

Vor allem was den Code-Review betrifft, unterstÃ¼tzt git die Programmierer durch Funktionen, um Patches direkt per Mail zu verschicken bzw. aus einer Mailbox direkt zu integrieren und eventuell wieder rÃ¼ckgÃ¤ngig zu machen.

Glossary

- branch - Enwicklungszweig (z.B. stable, und test)
der Standardbranch ist master
- commit - gesicherter Entwicklungsstand
- HEAD - Entwicklungsstand des letzten Commit, man kann durch anhÃ¤ngen von ^ auf den vorletzten Commit, bzw. mit ^^ auf den vorvorletzten (usw.) zugreifen

- HEAD^^ entspricht dabei HEAD~2

Hinweis

Wird lediglich der Befehl git angegeben, erscheint eine Liste mit den gebrÃ¤uchlisten Befehlen.

------- Code -------

```
glua@ike ~ $ git
usage: git [--version] [--exec-path[=GIT_EXEC_PATH]] [-p|--paginate] [--bare]
[--git-dir=GIT_DIR] [--help] COMMAND [ARGS]

The most commonly used git commands are:
   add           Add file contents to the changeset to be committed next
   apply         Apply a patch on a git index file and a working tree
   archive       Create an archive of files from a named tree
   bisect        Find the change that introduced a bug by binary search
   branch        List, create, or delete branches
   checkout      Checkout and switch to a branch
   cherry-pick   Apply the change introduced by an existing commit
   clone         Clone a repository into a new directory
   commit        Record changes to the repository
   diff          Show changes between commits, commit and working tree, etc
   fetch         Download objects and refs from another repository
   grep          Print lines matching a pattern
   init          Create an empty git repository or reinitialize an existing one
   log           Show commit logs
   merge         Join two or more development histories together
   mv            Move or rename a file, a directory, or a symlink
   prune         Prune all unreachable objects from the object database
   pull          Fetch from and merge with another repository or a local branch
   push          Update remote refs along with associated objects
   rebase        Forward-port local commits to the updated upstream head
   reset         Reset current HEAD to the specified state
   revert        Revert an existing commit
   rm            Remove files from the working tree and from the index
   show          Show various types of objects
   show-branch   Show branches and their commits
   status        Show the working tree status
   tag           Create, list, delete or verify a tag object signed with GPG
(use 'git help -a' to get a list of all installed git commands)
```

------- End-Code -------

Ãœblicherweise gibt es keinen Unterschied zwischen z.B. git status und git-status.

Erste Schritte

Nachdem git erfolgreich installiert wurde (es mÃ¼sste mittlerweile in jeder Distribution

enthalten sein), erstellt man zunÃ¤chst ein Verzeichnis, in welchem man folgenden Befehl ausfÃ¼hrt:

------- Code -------

```
glua@ike ~ $ mkdir git-test
glua@ike ~ $ cd git-test/
glua@ike ~/git-test $ git init # frÃ¼her git init-db
Initialized empty Git repository in .git/
glua@ike ~/git-test $
```

------- End-Code -------

Nun wurde in diesem Verzeichnis ein versteckter Ordner namens .git erstellt, in welchem nun sÃ¤mtliche Hintergrundinformationen fÃ¼r git gespeichert werden.

AuÃŸerdem befindet sich dort die Datei config (http://www.kernel.org/pub/software/scm/git/do
cs/git-config.html), in welcher ein paar grundlegende Einstellungen vorgenommen werden kÃ¶nnen.

------- Code -------

```
glua@ike ~/git-test $ cat .git/config
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true

[user]
        name="Julian Golderer"
        email="email@4-mail.net"
```

------- End-Code -------

Nun erstellen wir im aktuellen Verzeichnis irgendeine Datei (z.B.: echo "Hallo" > readme) und fÃ¼gen das ganze Verzeichnis mit dem Befehl git add . zum Repository hinzu (wird wiederum eine neue Datei erstellt, muss der Befehl evtl. nochmal wiederholt werden).

------- Code -------

```
glua@ike ~/git-test $ git add .
```

------- End-Code -------

AbschlieÃŸend speichern wir unseren ersten Arbeitsstand mit git commit.

------- Code -------

```
glua@ike ~/git-test $ git commit -a -m "1. commit"
```

------- End-Code -------

Durch den Parameter -a werden alle DateistÃ¤nde gespeichert, und mit -m kann eine Nachricht angegeben werden. Wird -m weggelassen, so erscheint ein Editor zur Eingabe einer Nachricht.

Es ist auch mÃ¶glich, einem Commit einen Namen zu geben. Dazu wird der Befehl git tag verwendet.

------- Code -------

```
glua@ike ~/git-test $ git tag v0.1
```

------- End-Code -------

Dieser kann dann mit git tag -d v0.1 wieder gelÃ¶scht werden.

Diff usw.

Eines der wichtigsten Features ist natÃ¼rlich das Anzeigen von Ã„nderungen. Dazu gibt es mehrere MÃ¶glichkeiten:
git log - zeigt eine Liste aller Commits
git show - zeigt diff des letzten Commits gegenÃ¼ber dem vorletzten
git diff - zeigt die Unterschiede zwischen dem derzeitigen Stand und der letzten Index-Aktualisierung mittels git add (frÃ¼her: git update-index)
git diff HEAD - zeigt die Underschiede zwischem dem derzeitigen Stand und dem letzten Commit
git diff v0.1 HEAD - zeigt die Unterschiede zwischen dem 'v0.1'-genannten Commits und des letzten Commits

Es ist auch mÃ¶glich, Unterschiede zwischen verschiedenen Branches anzeigen zu lassen, z.B. git diff master test.

Arbeiten mit mehreren Zweigen

Um eine neue Funktion auszuprobieren, kann es recht hilfreich sein, dafÃ¼r einen eigenen Entwicklungszweig (Branch) anzulegen.

------- Code -------

```
glua@ike ~/git-test $ git branch testing
glua@ike ~/git-test $ git branch
* master
  testing
```

------- End-Code -------

Mit git checkout testing wird nun in den testing-branch gewechselt.

Beim Wechseln zwischen verschiedenen Zweigen muss vorher ein Commit stattfinden,
um die Ã„nderungen zu speichern. Um bei Unterbrechungen, welche einen Wechsel
des Branches zur Folge haben, unschÃ¶ne Commits zu vermeiden, empfiehlt es sich,
die man-page zu git reset (http://www.kernel.org/pub/software/scm/git/docs/git-reset.html)
anzuschauen.

Ist beispielsweise die Entwicklung am testing-Branch abgeschlossen, so sollte
dieser wieder mit dem master-Branch vereinigt werden.

------- Code -------

```
glua@ike ~/git-test $ git checkout master
glua@ike ~/git-test $ git pull . testing
```

------- End-Code -------

Kommt es dabei zu Problemen, muss die entsprechende Datei in einem Editor geÃ¶ffnet
werden. Dort werden dann die konkurrierenden Zeilen angezeigt. Ist der Fehler
bereinigt, kann der aktuelle Status mit einem Commit gespeichert werden.

Verstecken von Dateien

Ãœber die Konfigurationsdatei .git/info/exclude ist es mÃ¶glich, Dateien von
der Ãœberwachung durch git auszuschlieÃŸen. Dies ist beispielsweise bei temporÃ¤ren
Sicherungsdateien (datei.php~) oder Dateien mit geheimen Konfigurationseinstellungen
(MySQL-PasswÃ¶rter, ..) hilfreich. Die EintrÃ¤ge dÃ¼rfen dabei regulÃ¤re AusdrÃ¼cke
enthalten.

Dateien kÃ¶nnen auch auf Verzeichnisebene unterschiedlich behandelt werden. Diese
Einstellungen werden dann in der Datei .gitignore des jeweiligen Verzeichnisses
hinterlegt.

Zusammenarbeit mit mehreren Entwicklern

Als Entwickler ist es natÃ¼rlich wichtig, immer das aktuelle Repository zu besitzen.
Liegt noch kein lokales vor, so wird Ã¼ber folgenden Befehl das Verzeichnis gespiegelt.

------- Code -------

```
glua@ike ~ $ git clone git://localhost/ git-clone
Generating pack...
Done counting 73 objects.
Deltifying 73 objects.
 100% (73/73) done
Total 73, written 73 (delta 10), reused 0 (delta 0)
glua@ike ~/git-clone $ git branch -a
* master
  origin/HEAD
```

```
  origin/master
  origin/testing

------- End-Code -------
```

Dabei werden automatisch neue Branches namens origin/ erzeugt, die die Orginalversion enthalten. Um spÃ¤ter das Verzeichnis zu aktualisieren, genÃ¼gt es, git pull auszufÃ¼hren.

```
------- Code: Beispiel -------

glua@ike ~/git-clone $ git pull
Generating pack...
Done counting 9 objects.
Result has 4 objects.
Deltifying 4 objects.
 100% (4/4) done
Total 4, written 4 (delta 1), reused 0 (delta 0)
Unpacking 4 objects
 100% (4/4) done
Updating from 2862f923d4960887dfd42c2eff71809fbd04ba0e to
764cc99a9066b33ea9d8d535254843b20b
003acb
Fast forward
 testdatei  |    1 +
 readme |    1 +
 2 files changed, 2 insertions(+), 0 deletions(-)

------- End-Code -------
```

Am Ball bleiben

Bei der gemeinsamen Entwicklung kann es hÃ¤ufig 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 auszufÃ¼hren, damit der origin/*-Zweig aktualisiert wird. Danach wird die eigene Entwicklung mit git rebase origin/master master dem aktuellen Stand angepasst.

Zentralen git-Server aufsetzen

Damit mehrere Personen gemeinsam arbeiten kÃ¶nnen, ist immer eine zentrale DatenlagerstÃ¤tte
 nÃ¶tig. Dazu wird zuerst das eigene git-Repository mit git clone --bare .git /pfad/zum/neuen/gitrepo.git geklont und mit scp o.Ã¤. auf den Server verschoben.

Legt man ein neues git-Repository an, empfiehlt es sich, dies gleich mit git init --shared zu tun. Danach fÃ¼hrt man seinen ersten Commit aus und benennt das .git-Verzeichnis um (z.B. in repo.git). Dieses kann dann auf den Server verschoben werden.

Als nÃ¤chstes werden auf dem Server Accounts fÃ¼r die einzelnen User sowie eine zentrale Gruppe fÃ¼r das Projekt angelegt. Die User sollten dabei diese Gruppe und /usr/bin/git-shell als Shell zugewiesen bekommen. FÃ¼r mich hat es sich bewÃ¤hrt, alle git-Repositories in /home/git oder /srv/git zu speichern und die User git-username zu nennen. Ãœber chown wird dem Repo ein entsprechender User/Gruppe zugewiesen und danach mit chmod fÃ¼r die Gruppe beschreibbar gemacht.

Damit verschiedene Zugriffrechte auf Userebene gesetzt werden kÃ¶nnen, muss die Datei update im hooks-Ordner durch die diese (http://www.kernel.org/pub/software/scm/git/doc
s/howto/update-hook-example.txt) Datei ersetzt werden (Achtung..zahlreiche Zeilen mÃ¼ssen gelÃ¶scht werden). Nun kÃ¶nnen die User-Rechte in info/allowed-users gesetzt werden.

------- Code: $GITDIR/info/allowed-users -------

```
refs/branch/master    git-user1 git-user2
refs/branch/.*        git-chef
refs/tags/v.*         git-user2
```

------- End-Code -------

Laut diesen Angaben dÃ¼rften git-user1 und git-user2 in den master-Branch hochladen und git-user2 dÃ¼rfte tags setzen. Nur git-chef hÃ¤tte die Berechtigung in sÃ¤mtliche Branches zu schreiben.

Hinweis: Als Jokerzeichen mÃ¼ssen in allowed-users regulÃ¤re AusdrÃ¼cke verwendet werden.

Die Ã„nderungen werden dann von den Clients mit git push auf den Server hochgeladen.

Weblinks

- git Dokumentation (http://www.kernel.org/pub/software/scm/git/docs/)
- git Homepage (http://git.or.cz/)
- Everyday git (http://www.kernel.org/pub/software/scm/git/docs/everyday.html)
- GIT Ã¼ber WebDAV (http://www.kernel.org/pub/software/scm/git/docs/howto/setup-git-server-o
ver-http.txt)