Gitting started

Christian Neukirchen

05nov2010

Table of contents

Wieso Versionskontrolle?

Den Verlauf eines Projekts speichern

Zu alten Versionen zurückkehren

“Gestern ging das noch…”

Mehrere Versionen parallel pflegen

An einem Projekt zusammenarbeiten

Zur Geschichte

Klassisch: SCCS (1972), RCS (1982),
CVS (1989), SVN (2000)

Verteilt: GNU Arch (2001), Monotone (2003),
Darcs (2003), Bazaar (jan2005),
Git (7apr2005), Mercurial (19apr2005)

Wieso verteile Versionskontrolle?

Jedes Repo hat die *gesamte* History.

Alle *repositories* sind a priori gleich.

Ideal für *open source*…

… aber auch für die Arbeit offline.

Wieso Git?

Populär: Linux, X.org, GCC, Android, Wine, Exherbo, GNOME, Perl 5, Qt, Samba…

Schnell, Robust, Unixig

GPL

Jetzt gits los!

```
% packer -S git

% emerge -av dev-vcs/git

% port install git-core

% apt-get install git-core
```

whoami

Um Commitbeschreibungen sinnvoll auszufüllen, braucht Git den Namen und die Mail-Adresse des Users.

```
% git config --global user.name \
    "Christian Neukirchen"
% git config --global user.email \
    chneukirchen@gmail.com
```

Editor kontrollieren:

```
% echo $EDITOR
```

Erste Schritte: Workshop

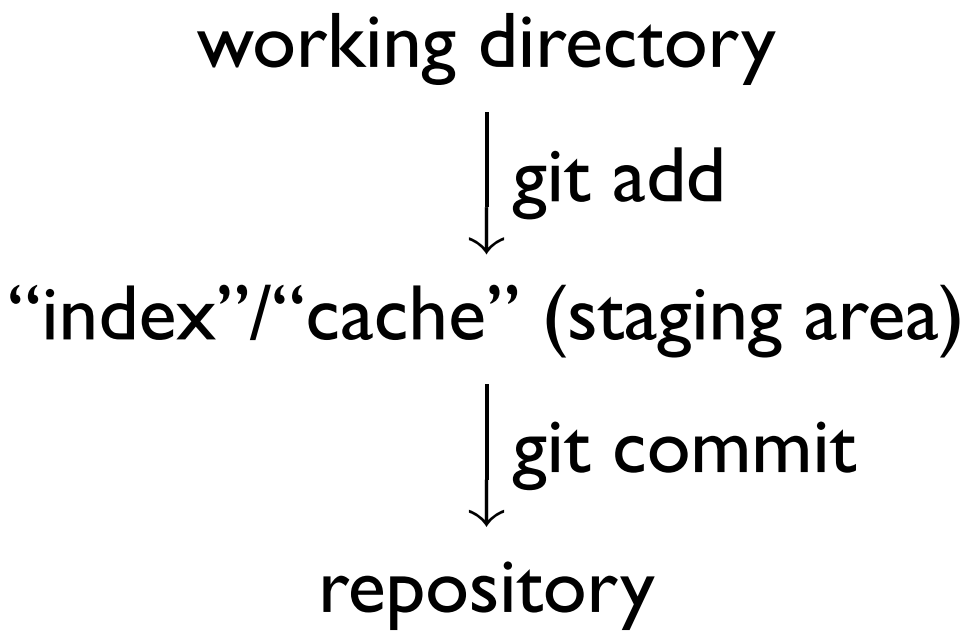
```
% git init myrepo
% cd myrepo

myrepo% vi hello-world.c
myrepo% git status
myrepo% git add hello-world.c
myrepo% git status
myrepo% git commit
```

Zweite Schritte: Workshop

```
myrepo% git log
myrepo% vi hello-world.c
myrepo% git status
myrepo% git diff
myrepo% git add hello-world.c  # !!!
myrepo% git diff --staged
myrepo% git commit
```

Der Index


working directory
git add
“index”/“cache” (staging area)
git commit
repository

Interaktiv committen: Workshop

```
myrepo% git add -p
myrepo% git commit
```

Schneller: Workshop

```
myrepo% vi hello-world.c
myrepo% git commit hello-world.c
```

Noch schneller: Workshop

```
myrepo% vi hello-world.c
myrepo% git commit -a
```

Mehrere Dateien: Workshop

```
% mv hello-world.c goodbye-world.c
% git add goodbye-world.c
% git rm hello-world.c
% git status
% git commit

% git mv hello-world.c goodbye-world.c
```

Git speichert das Umbenennen *nicht*, sondern erkennt es heuristisch. Funktioniert prima.

Wie funktioniert das alles?

```
git show-ref
git show --format=raw $COMMITID
git ls-tree $TREEID
git show $BLOBID
```

Exkurs: Revisionsnamen angeben

`HEAD`	Stamm
`b130ec682`	Eindeutiges Hash-Präfix
`refs/tags/$NAME`	Tags
`refs/heads/$NAME`	Andere Stämme
`refs/remotes/$NAME`	Andere Repos
`$REF^`	Erster Vater
`$REF~n`	n-ter Vorfahre
`$REF:$PATH`	`$PATH` im Tree `$REF`
`$REF1..$REF2`	Alle Eltern von `$REF2` ohne Eltern von `$REF1`

Branching

```
% git branch
% git checkout -b chris2

% git branch
% git log
% git log master
% git log master..chris2
% git diff master..chris2
```

Merging

```
% git checkout master
% ...
% git diff chris2
% git merge chris2
% git diff
% git log --graph --oneline
```

Tagging

```
% git tag $TAGNAME
% git show-ref
```

fetch, pull und push

```
% git clone gauss.gaf.fs.lmu.de:/gaf/self/git/hello
hello% git fetch
hello% git pull

hello% git remote add goodbye \
         gauss.gaf.fs.lmu.de:/gaf/self/git/goodbye
hello% git fetch goodbye
hello% git log master..goodbye/master
hello% git merge goodbye/master

hello% git pull goodbye master

hello% git push
```

blame

Wer hat das verbrochen?

```
% git blame hello-world.c
```

Patches erstellen

```
% git checkout catwell/master

% git diff master..

% git format-patch \
    --stdout origin/master

% git format-patch origin/master~3
```

Empfehlungen

Viele, kleine Commits

Spezifische Commits

Sinnvolle Commit-Messages

Feature-Branches

Nur einer (oder wenige) committed in den master.

Upps, was vergessen/vertippt

```
% git show-ref master
% git commit --amend
% git show-ref master
```

Die Operation ist *destruktiv*, “history is rewritten”.

Nur Privat verwenden!

Upps, zurück…

```
% git add foo
# Upps, nee...
% git reset

% git commit -a
% git log
# Upps, nee...
% git reset --hard HEAD^
% git log
```

Obacht, kann zu Datenverlust führen!

Wie werd ich diese tausend Dateien los?

```
% find .git/objects
% git gc
% find .git/objects
```

Eigentlich nicht nötig, von Hand aufgerufen zu werden.

fsck

```
% git fsck
```

Ist das “Git-Filesystem” konsistent?
Und wo ist nach einem Reset mein alter HEAD hin?

```
% git fsck --lost-found
% git reset --hard $COMMITID
```

Mal nicht mergen

Ausser mergen kann man auch rebasen:

```
% git rebase BRANCH
% git log --graph --oneline
```

oder einfach einzelne Commits “cherry-picken”:

```
% git cherry chris2
% git cherry-pick $COMMITID
```

Dateien ignorieren

```
% latex foo.tex
% git status
% echo "*.aux" >>.gitignore
% echo "*.toc" >>.gitignore
% git status
```

Am besten committed man die `.gitignore` auch ins Repo.

Stashing

```
*werkel... oh mist, ich muss pullen*
% git pull
Updating 3e4833b..85ca454
error: Your local changes to the
following files would be
 overwritten by merge:
        lib/rack/request.rb
% git stash
% git pull
% git stash list
% git stash apply
*weiterwerkel*
```

Git GUIs

gitk (Visualisierung der History, nützlich)

git-gui (Grafisches Commit-Tool)

Mac OS X: GitX, Gity

Zusammenarbeit mit SVN

```
% git svn init \
http://conque.googlecode.com/svn/trunk/
conque
% git svn fetch  # dauert erstmal
% ... commit ...
% git svn dcommit
```

Zum Wechseln von SVN gibts auch svn2git.

Hacks: homegit

Wie kann ich meine Dotfiles mit Git managen?

... ohne ein ~/.git (sicherer, sauberer)

```
% alias homegit="GIT_DIR=~/prj/dotfiles/.git git"
```

```
% homegit add .zshrc
% homegit commit
% homegit push github
% ...
```

Hacks: Webseiten aktualisieren

Git-Repos haben “hooks”, Skripte die bei bestimmten Aktionen ausgeführt werden:

```
% find .git/hooks
% cat .git/hooks/post-receive
#!/bin/sh
cat >/dev/null
cd ..
GIT_DIR=.git git checkout -f master
cd ..
ruby vuxi.rb
```

Hacks: Heroku

Die Idee konsequent weitergedacht: Heroku

```
$ git push heroku master
```

… und die Applikation wird automatisch deployt.

Ausblick

```
% ls -l /usr/lib/git-core | wc -l
153
```

Noch nicht diskutiert wurden:

am apply archimport archive bisect bundle cat-file check-attr checkout checkout-index check-ref-format citool clean commit-tree count-objects cvsexportcommit cvsimport cvsserver daemon describe diff-files diff-index difftool diff-tree fast-export fast-import fetch-pack filter-branch fmt-merge-msg for-each-ref get-tar-commit-id grep hash-object help http-backend http-fetch http-push imap-send index-pack init-db instaweb lost-found ls-files ls-remote mailinfo mailsplit merge-base merge-file merge-index merge-octopus merge-one-file merge-ours merge-recursive

Ausblick

merge-resolve merge-subtree mergetool merge-tree mktag mktree

name-rev notes pack-objects pack-redundant pack-refs parse-remote

patch-id peek-remote prune prune-packed quiltimport read-tree

receive-pack reflog relink remote-ftp remote-ftps remote-http

remote-https remote-testgit repack replace repo-config request-pull rerere

revert rev-list rev-parse send-email send-pack shell shortlog show-branch

show-index show-ref sh-setup stage stash status stripspace submodule svn

symbolic-ref tar-tree unpack-file unpack-objects update-index update-ref

update-server-info upload-archive upload-pack var verify-pack verify-tag

whatchanged write-tree