Verteilte Versionskontrolle mit Git

Versionierung für OpenSourcler

Sebastian „tokkee“ Harl
<tokkee@debian.org>

OpenRheinRuhr 2010
14. November 2010

OPEN RHEIN RUHR
Ein Pott voll Software

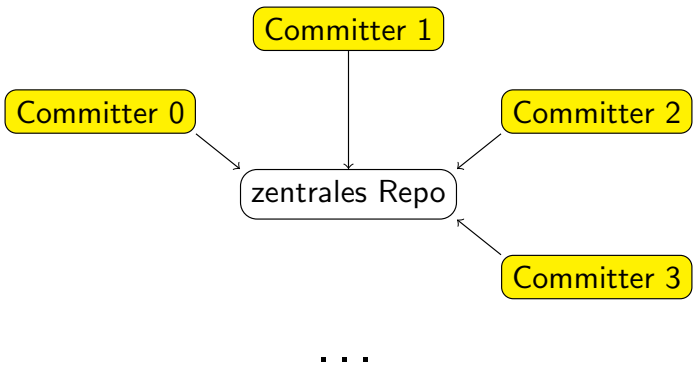
Statistiken: Wer seid ihr?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
  - . . . mit mehr als 1 Entwickler?
  - . . . mit mehr als 10 Entwicklern?
  - . . . mit mehr als 100 Entwicklern?
  - . . . mit mehr als 1000 Entwicklern?
- Wer hat schon ein Versionsverwaltungssystem (VCS) verwendet?
- Wer hat schon ein zentrales VCS (CVS, SVN, . . .) verwendet?
- Wer hat schon ein dezentrales VCS (Git, bzr, Mercurial, . . .) verwendet?
- Wer hat schon mit Git gearbeitet?

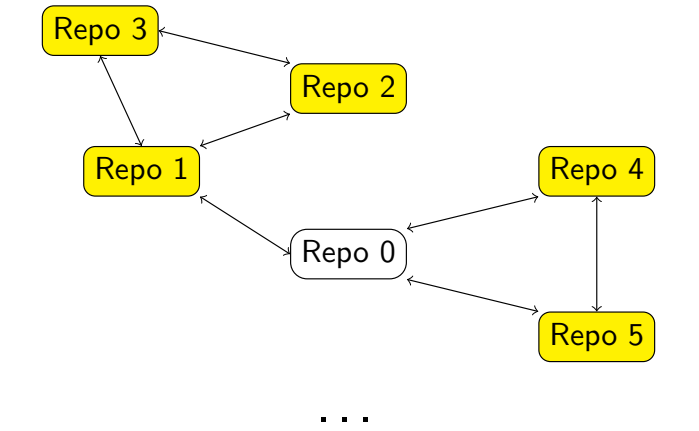
Grundlagen: Was ist Versionskontrolle?

- technisch gesehen: ein Haufen Dateien mit Meta-Informationen und irgendwelchen Beziehungen untereinander ☺
- Protokollieren von Änderungen an (Quell-)text
- Archivierung mit „Rücksetz-Operation“
- koordinierter Zugriff
- parallele Entwicklungszweige (neue Features, alte Releases)

Grundlagen: Typen von VCSen



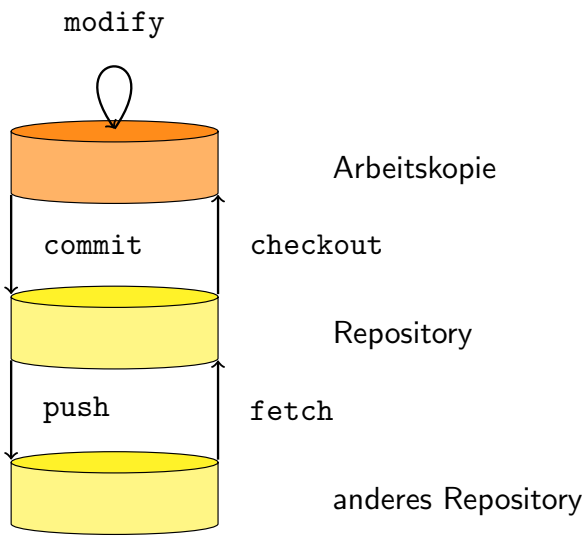
Workflow in OpenSource Projekten



Grundlagen von dezentralen VCSen

- „Peer-to-Peer“ Ansatz
- jede Arbeitskopie bringt ein komplettes Repository mit (Klon)
- gearbeitet wird auf lokalem Repository
  ⇒ kein Netzwerk-Zugriff nötig
  ⇒ Operationen schnell
  ⇒ Offline-Arbeit möglich
- automatisches „Backup“ durch Repository-Klons
- Zusammenführen meist auf Basis eines „Web-of-Trust“

Arbeitsweise



Git: Übersicht

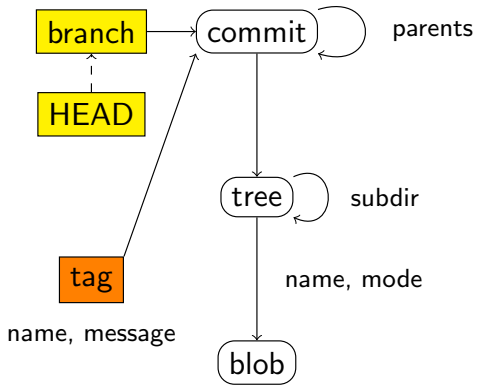
- `http://www.git.or.cz/`
- VCS (Version Control System)
- 2005 von Linus Torvalds initiiert
  (aktueller Maintainer: Junio C. Hamano)
- dezentral
- schnell und effizient
- kryptographisch gesichert
- „Toolkit design“
- OpenSource (GPLv2)
- weit verbreitet im Einsatz (z.B. Linux Kernel, Ruby on Rails, Perl, WINE, X.org, GNOME, Qt, Debian, . . .)

Arbeiten mit Git: Grundlagen

- >> 100 einzelne Befehle
- „Porcelains“ und „Plumbing“
- Dokumentation als Manpages - `git(7)`
- `git help`, `git <command> -h`
- Benutzer Handbuch: `http://www.kernel.org/pub/software/scm/git/docs/user-manual.html`

Datenhaltung: Git Objektdatenbank

- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe



Git konfigurieren

Sich Git vorstellen

- `git config --global user.name <Dein Name>`
- `git config --global user.email <du@deine-domain.tld>`
- → Benutzerinformationen für Commit-Metadaten

Bunt und in Farbe

- `git config --global color.ui auto`
- → farbige `branch`, `diff`, `grep`, `status` Ausgaben

Repositories erstellen

Neues, leeres Repository

```
$ mkdir project
$ cd project
$ git init
Initialized empty Git
repository in .../.git/
```

Bestehendes Repository „klonen“

```
$ git clone <rep>
...
```


Änderungen vornehmen

Ändern

```
$ vim foo bar
$ git add foo bar
```

- `add`, `rm`, `mv`

Geschichte fortführen/ändern

```
$ git commit
```

```
$ git reset --hard HEAD^
```

- `reset`, `revert`, `checkout`

Exkurs: Commit Meldungen schreiben

- Einzeilige, kurze (< 80, optimal < 50 Zeichen) Zusammenfassung
- Leerzeile
- Detaillierte Beschreibung/Erklärung
- nicht vorgeschrieben, aber „common practice“ und von vielen Tools erwartet

Exkurs: Commit Meldungen schreiben

```
git-remote: do not use user input in a printf format string

'git remote show' substituted the remote name into a string that
was later used as a printf format string. If a remote name contains
a printf format specifier like this:

   $ git remote add foo%sbar .

then the command

   $ git remote show foo%sbar

would print garbage (if you are lucky) or crash. This fixes it.
```


Aus der Geschichte lernen

Status der Arbeitskopie

```
$ git status
$ git diff
```

Historie betrachten

```
$ git log
$ tig
```

Objekte betrachten

```
$ git show
$ git show HEAD:foo
```

- Commits, Trees, Blobs, Tags

Tags

```
$ git tag -m ‘‘<Beschreibung>’’ <Name> <Commit>
$ git tag -l
```

- „Zeiger“ auf einen Commit
  optional mit Metadaten (“annotated tag“)
- Kennzeichnung von bestimmten Entwicklungsständen (insb. Releases)
- „annotated tag“: Autor, Datum, Beschreibung, optional GnuPG Signatur

Branching und Merging

- Branch: „automatischer“ Zeiger auf eine Reihe von Commits
- `HEAD`: Zeiger auf den aktuellen Branch
- `master`: „Standard“-Branch
- Merge: Zusammenführen von zwei Entwicklungssträngen

Branch erzeugen

```
$ git checkout -b <Name>

$ git branch
  master
* new-branch
```


Branching und Merging

Branches zusammenführen

```
$ git merge master
$ git rebase master
```

Konflikte auflösen

- Konflikte entstehen, wenn die gleiche Stelle unterschiedlich geändert wurde
  ⇒ manuelles Eingreifen nötig
- Commit-Erzeugung wird unterbrochen
- Konfliktanzeiger in den betroffenen Dateien
- manuelle Entscheidung, wie beide Änderungen zusammengeführt werden

Arbeiten mit anderen Repositories

Repository klonen

```
$ git clone <rep>
```

Austauschen von Änderungen

```
$ git pull
$ git push
```

Alternativ:

- `git format-patch`, `git send-mail`

Arbeiten mit anderen Repositories

- „remote“: Repository, dessen Änderungen verfolgt werden
- „remote branch“: Branch, welcher der Zustand in einem anderen Repository wiederspiegelt
- technisch: Branch in einem anderen Namensraum mit anderer Semantik

Arbeiten mit „remotes“

```
$ git remote add <Name> URL
$ git remote update <Name>
```

Repository URLs

- lokal: `/path/to/repository/`
- http: `http://domain.tld/repository.git`
- git: `git://domain.tld/repository.git`
- ssh: `domain.tld:path/to/repository/`

Frontends

- `tig` (ncurses)
- `gitk` (Tk, read-only)
- `qgit` (Qt)
- `magit` (emacs)

Goodies

- `git reflog`
- `git rebase -i`
- `git commit --amend`
- `git add -p`
- `git stash`
- `git bisect`
- `git cherry` / `git-wtf`
- `git diff color-words`
- `git svn`

Versionierung für OpenSourcler

Vielen Dank für die Aufmerksamkeit!

Gibt es Fragen?

Kontakt:
Sebastian „tokkee“ Harl
<tokkee@debian.org>