git for noobs

next generation distributed version control system(s)

Michael Rodler

2010-10-27

Vorstellung

Michael Rodler

- Kennt mich wer net?
- aka f0rk, f0rki, f0rkmaster, Gabel, etc.
- Student SIB09
- Bewohner Awesome WG

Skills? Experience?

- Lesen
- Leider keine Erfahrung mit git im “professionellen” Umfeld
- kaum Erfahrung mit Subversion

Hagenberger Kreis

Table of contents

1. Introduction
2. Einführung in Versionskontrollsysteme
   - History – 1st Generation
   - History – 2nd Generation
   - 2nd gen – What sucks?
   - Distributed, Fast, Awesome – 3rd Generation
3. How GIT works
   - Object Model
4. How to use GIT
   - Basic Usage
   - Branching
   - Repository
   - Nützliches
5. References

Hagenberger Kreis

Damals... im letzten Jahrtausend

1st gen

RCS (Revision Control System) und CVS (Concurrent Versions System)

- Versionskontrolle pro Datei
  - sehr unpraktisch für Software Developement
  - Abhängigkeit zwischen Dateien
- Netzwerk und Multiuser nur über Hacks
- Just a pain in the ass...

Hagenberger Kreis

Damals... vor 10 Jahren

2nd gen

Subversion (svn)

- Zentrales *Repository* – Ein Server der alle Versionen verwaltet
- User machen ein *checkout* einer bestimmten Version
  - User besitzen eine *"Working Copy"* einer Version
  - User nimmt Änderungen vor
  - User übergibt den Zustand der gesamten Working Copy dem VCS
    $\rightarrow$ ***commit***

Hagenberger Kreis

Subversion Workflow Example

- Alice checks out version 42
- Bob checks out version 42
- Bob changes line 1 in fu.txt
- Bob commits version 43
- Alice also changes line 1 in fu.txt
- Alice also commits version 43
- But there is already a version 43 on the Server
- CONFLICT
- Alice needs to *merge* her changes into version 43
  Either accepting her or Bobs line 1 in fu.txt
- Alice commits (the merged) version 44

Hagenberger Kreis

Branches

Mainline

- meist ein Hauptentwicklungszweig
  - Subversion: *trunk*
  - git: *master*
  - Mercurial: *default*
- Hauptzweig bleibt "stable"
  - Compilable/Runnable
  - Deployable
  - besteht Unit Tests

Branches

- experimentelle Features (kompilieren/laufen nicht)
- Nach Fertigstellung Merge in die Mainline
- Regelmäßig Änderungen aus Mainline importieren um Merge zu erleichtern

Klingt gut. Wieso git?

Nachteile

- Merge-Tracking erst seit kurzem in Subversion
  - Merge von Branch und Mainline
  - History der Mainline geht verloren
- Subversion tracks files
  - explizites rename/move tracking
- Repository Server ist Single Point of Failure
- Subversion is slow (compared to git ;)
  - Berechnungen/Operations am Server + Netzwerk Latency
- Subversion Repos sind rrrrrrießig
  - Mozilla project's repository [4]

CVS	SVN (fsfs)	git
3GB	12 GB	300 MB

- Linus Torvalds didn't like Subversion

3rd Generation

3rd Generation

- git
- mercurial
- bazaar
- etc.
  - Monotone
  - Darcs
  - SVK

Benchmarks unter [5]

3rd Generation

Distributed

- komplette History (auch) lokal
  - viele Aufgaben einfach schneller
  - keine Network Latency
- Arbeiten auch ohne Netzwerk Verbindung
  - commit, merge, branch, etc.
- Arbeiten komplett ohne Server (praktisch oder?)
  - Changes zwischen lokalen Repos hin und her schieben
- Server kein Single Point of Failure mehr

Hagenberger Kreis

Tracking Content

Trennung von Content und Meta-Informationen

- Implizites Tracking von Renames/Moves
- Keine redundanten Files
- History bezieht sich auf Content nicht auf Files

GIT Objects

File Format

- Keine Datenbank dahinter
- Objekte sind normale files/directories
- Objekte werden durch ihren SHA1-Hash indentifiziert

4 Arten von Objekten

- Blob
- Tree
- Commit
- Tag

Hagenberger Kreis

GIT Objects

Blob

"Binary Large OBject"

- komprimierter Content einer Datei (gzip)
- KEINE Metainfo
  - kein Dateiname
  - keine Permissions
  - Nur Content

Hagenberger Kreis

GIT Objects

Tree

- Jedes Directory wird durch einen Tree repräsentiert
- Speichert Referenzen auf andere Trees und Blobs
- Meta Informationen zu referenzierten Objekten

```
$ ls
doc  src
$ ls -a
.  ..  doc  .git  src
$ ls -l ./src
total 4
-rwxr--r-- 1 mikey mikey 277 2010-10-24 17:45 main.c
$ git ls-tree master
040000 tree b2b11bf4c8878b15757108545dff4b8619a453bf  doc
040000 tree 2f518093f0a1da8aae6582704fac8c49d014ecbe  src
$ git ls-tree 2f518093f0a1da8aae6582704fac8c49d014ecbe
100755 blob c3e5770622044cdad9b79a52493cad8321a6a1fc
    main.c
$ git show c3e5770622044cdad9b79a52493cad8321a6a1fc
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  if (argc != 1) {
[...]
```

GIT Objects

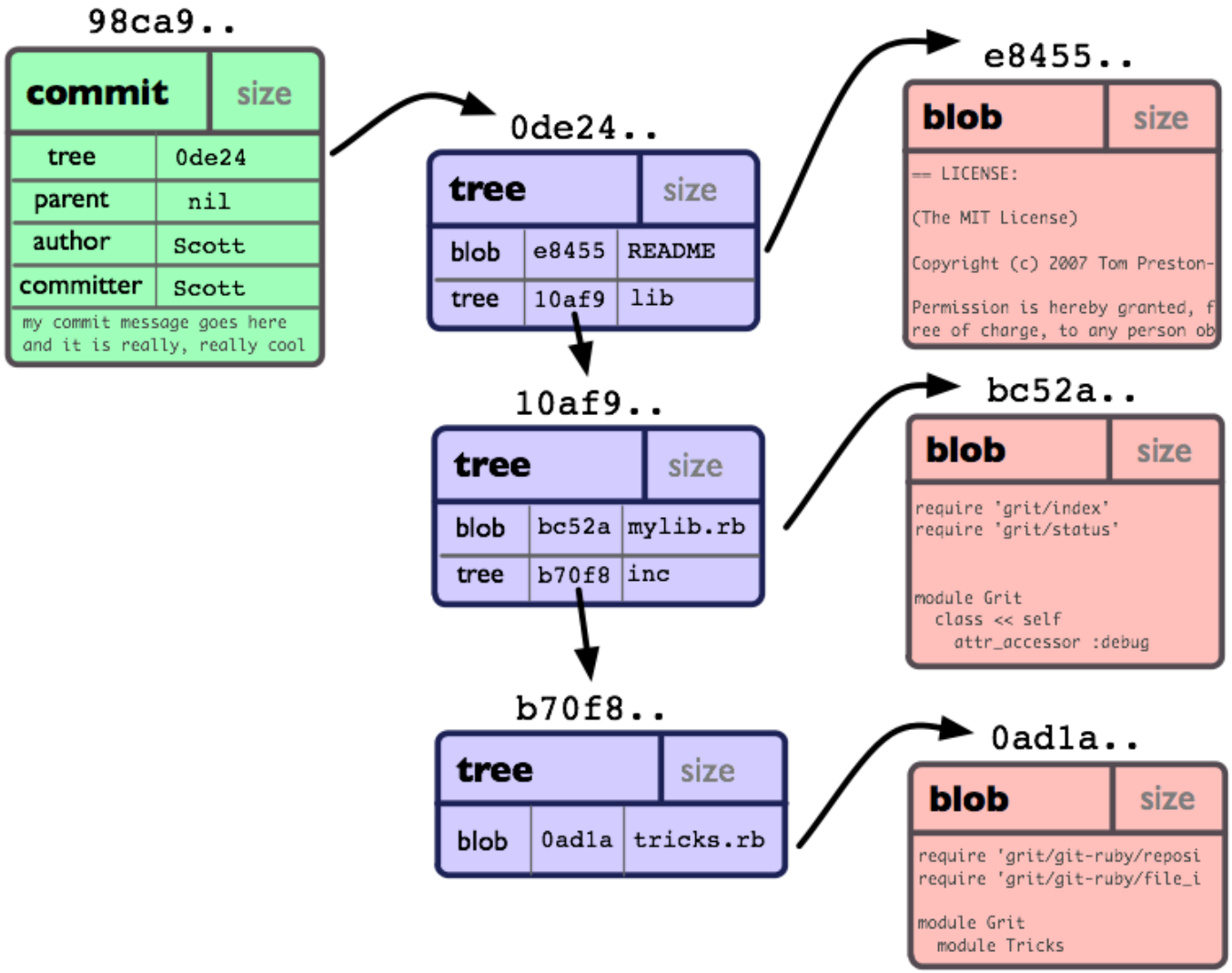
Commit

- Repräsentieren den Status des Repositories zu einem Zeitpunkt
- Referenz auf einen Tree
- Referenz auf einen oder mehrere Parent Trees
- Author
- Committer
- ein Kommentar

```
$ git log --pretty=raw
commit 92eb1f398f92d65d6cbef9f4e2f7111a5e33ec33
tree cd4c8db3688bd9f7e328d9ca70f727f58d8238d7
author Michael Rodler <michael@michaelrodler.at>
    1287935145 +0200
committer Michael Rodler <michael@michaelrodler.at>
    1287935145 +0200

    added readme and main.c
```

The Bigger Picture



[1]

GIT Objects

Tag

- Man kann Objekte “taggen” (d.h. mit einem Namen versehen)
- Referenz auf ein Objekt
- Typ des Objekts
- Name des Taggers
- Ein Kommentar
- Beispiele
  - Ein *Commit* wird als “v1.0” markiert
  - Ein *Tree* wird als “broken” markiert

Hagenberger Kreis

So einfach?

Loose Object

- Das was ich gerade erzählt habe
- Ineffizient – jede Version ein Blob

Packfiles

- Packs files
- More compression
- Heuristic delta magic
- `git gc`
- Mehr Info in [1] und [2]

Hagenberger Kreis

Cloning, Pulling/Fetching, Pushing

clone

```
git clone git://git.kernel.org/pub/scm/git/git.git
```

- Erstellt lokale Kopie des Repos
- Verschiedene Protokolle
  - `http(s)://host:port/path/to/repo`
  - `ftp(s)://host:port/path/to/repo`
  - `git://user@host:port/path/to/repo`
  - `ssh://user@host:port/path/to/repo`
  - local path `/path/to/repo` oder `file:///path/to/repo`

Hagenberger Kreis

Cloning, Pulling/Fetching, Pushing

pull

`git pull`

- Änderungen aus dem remote Repo

fetch

`git fetch`

- Änderungen aus dem remote Repo ohne diese ins lokale Repo zu mergen

push

`git push`

- Schiebt Änderungen in das remote Repo

Hagenberger Kreis

Adding, Commiting, Checking out

add

`git add file`

- Neues file hinzufügen
- "staging" von geänderten Files

commit

`git commit`

- commit staged files

`git commit -a`

- stage all changed files and commit

Adding, Commiting, Checking out

checkout

`git checkout file`

- Files/Dirs in Zustand des letzten Commits versetzen

`git checkout branchname`

- branch wechseln

`git checkout master^`

- Spezifische Versionen auschecken
- Mehr in [1] Seite 64

Hagenberger Kreis

Status of the Repo

status

`git status`

- Zeigt Status der Files in der Working Copy

diff

`git diff --cached`

- Diff von “Staging Area” und letztem Commit (HEAD)

Status of the Repo

log

`git log`

- zeigt vergangene commits an
- `git log -p` für log mit diffs
- `git log --graph` oder
  `git log --pretty=format:'%h : %s' --graph`

Änderungen Rückgängig machen

Ucommited

- Einzelne Files/Directories
  `git checkout filename`
- Ganze Working Copy `git reset --hard HEAD`

Commited

- Zu einer früheren Version zurückkehren
  `git revert HEAD^`
- Unveröffentlichten Commit löschen
  `git reset --hard HEAD^`

Branching

Erstellen

```
git branch branchname
```

- sehr schnell
- kaum Overhead
- TU ES!

Wechseln

```
git checkout branchname
```

Branching

merge

`git merge branchname`

- Resultiert in “merge commit”
- Konflikte müssen aufgelöst werden
- Fast-Forward Merge
  - Alle Commits der Branch befinden sich auch in der Anderen
  - Kein merge commit

Hagenberger Kreis

Branching

Delete Branches

- Branch löschen, es wird sichergestellt dass die Änderungen gemerged wurden
  `git branch -d branchname`
- Branch immer löschen
  `git branch -D branchname`

GIT Repository

Repository erstellen

- Lokales Repo erstellen
  `git init`
- Remote Repo, ohne Working Copy erstellen
  `git init --bare`

Maintenance

- Garbage Collecting, Repacking von Files
  `git gc`
- Kosistenz Check (dangling objects können ignoriert werden)
  `git fsck`

Hagenberger Kreis

GIT Repository

Remote Repository

- Ein Remote Repository hinzufügen
  `git remote add bob url`
- Man kann nun von “bob” pullen, pushen, mergen, etc.
- Eine lokale Branch soll eine Branch von Bob “tracken”
  `git branch --track bob-experimental bob/experimental`
- Remote Repository “origin” ist das Repository von dem geklont wurde

Nützliche Kommandos

grep

```
git grep regex
```

- wie grep, nur durchsucht auch alte Versionen

blame

```
git blame filename
```

- gibt alle Zeilen aus + Commiter + Hash + Date

Stashing

Stash

- Working copy “stashen”
  `git stash`
  - Working Copy wird gespeichert
  - Checkout auf den letzten commit
- was ist gestashed?
  `git stash list`
- Working Copy auf Stash zurücksetzen
  `git stash apply`
  - Automatischer Merge

Hagenberger Kreis

Referenzen

- http://book.git-scm.com/
- http://progit.org/
- Chaos Radio Express 130 – Verteilte Versionskontrollsysteme
  http://chaosradio.ccc.de/cre130.html
- https://git.wiki.kernel.org/index.php/GitSvnComparsion
- http://ldn.linuxfoundation.org/article/dvcs-round-one-system-rule-them-all-part-3

Hagenberger Kreis