Versionsverwaltung mit Git

Linux-Cafe - 2013-11-11

Referent: Bernd Strößenreuther

mailto:linux-cafe@stroessenreuther.net

Lizenz

- Sie dürfen dieses Dokument verwenden unter den Bedingungen der Creative Commons Lizenz: http://creativecommons.org/licenses/by-nc-sa/3.0/de/

- Alle Grafiken und Icons von OpenClipArt.org ”released to the public domain”.

Agenda

- Teil 1 – Einsteiger-Level
- Optional: Teil 2 – Intermediate Level
- Ausblick: Teil 3 – Expert Level (abgefahrenes Zeug)

Teil 1 – Einsteiger-Level

- Alles, was man erst mal wissen muß

Was ist Git?

- Ein Versionsverwaltungssystem
- Englisch: Version Control System (VCS)

Was macht ein Versionsverwaltungssystem?

- Mehrere Versionen von einer oder mehreren Dateien werden verwaltet
- Nachvollziehbarkeit: Wer hat wann was geändert?
- Alle älteren Versionen jeder Datei sind noch verfügbar
- Ein gemeinsamer Versionsstand über alle Dateien kann gebildet werden: Tag
  - z. B. ein Software-Release

Arten von VCS

- Lokale Versionsverwaltung
  - z. B. rcs
- Zentrale Versionsverwaltung
  - Ein zentraler Server hält das Repository
  - Anwender hat auf seinem System lediglich eine Arbeitskopie
  - z. B. Subversion (SVN) oder
    Concurrent Versions System (CVS)
- Verteilte Versionsverwaltung
  - Jeder hat eine komplette Version des Repository
  - z. B. Git, Mercurial (hg) oder Bazaar

Warum Git?

Vorteile des Verteilten Ansatzes:

- Vollständig arbeiten ist auch offline möglich

Vorteile von Git:

- Die am häufigsten genutzen Funktionalitäten sind sehr einfach zu bedienen
- Weit verbreitet / Gut getestet
- OpenSource
- Skaliert extrem weit nach oben

Wo kommt Git her?

- Linus Torvalds war 2005 der Meinung, dass kein aktuell verfügbares freies Versionsverwaltungssystem geeignet wäre für ein so großes und so verteiltes Entwicklungsmodell, wie es beim Linux-Kernel der Fall ist
- Daher hat er sich innerhalb von ein paar Tagen mal schnell ein VCS geschrieben: Git war geboren
- Heute: Extrem weit verbreitet für die unterschiedlichsten Einsatzzwecke

Wo kommt das Wort ”Git” her?

- Englisch für ”Depp”

Installation

- `aptitude install git`
- Auch für Windows verfügbar:
  separat herunterladen und installieren
- Web: http://git-scm.com

Minimal-Konfiguration von Git

- Name / eMail-Adresse festlegen:

```
git config --global user.name "Hans Wurst"
git config --global user.email "hans.wurst@example.com"
```

- Optional: Farbe aktivieren:

```
git config --global color.ui "auto"
```

- Konfigurationsdatei:

```
~/.gitconfig
```

- Konfiguration anzeigen:

```
git config --list
```

Ein lokales Repository anlegen

```
mkdir meinRepo
cd meinRepo
git init
```

Eine Datei hinzufügen

- Datei anlegen:
```
echo test > test.txt
```
- Git status anzeigen lassen:
```
git status
```
- Datei für den nächsten Commit vormerken:
```
git add test.txt
git status
```
- Eine (von git verwaltete) Version erzeugen (Commit):
```
git commit -m "erste Version"
git status
```

Zentrales Repository: Einrichten am Server

- Als root:

```
mkdir /opt/meinRepo.git
chgrp users /opt/meinRepo.git
chmod g+ws /opt/meinRepo.git
```

- als (unprivilegierter) User:

```
cd /opt/meinRepo.git/
git init --bare
```

Zentrales Repository: Clone (am Client)

```
git config --global push.default simple
mkdir ~/git && cd ~/git/
git clone ssh://meinServer/opt/meinRepo.git
cd meinRepo/
git status
echo test > test.txt
git add test.txt
git commit -m "erste Version"
git push
```

Verteiltes Arbeiten

- Vor der Änderung: Aktuellen Stand abholen
  ```
  git pull
  ```
- Änderung durchführen
  ```
  vim test.txt
  ```
- Nochmal aktuellen Stand abholen
  ```
  git pull
  ```
- Lokale Änderungen werden dabei **nicht** überschrieben
- Commit (lokal)
  ```
  git commit -am "noch eine neue Zeile dazu"
  ```
- Übertragen der Änderungen ins zentrale Repo
  ```
  git push
  ```

Nützliche Commands

- Lokale Änderungen verwerfen
  ```
  git checkout -- test.txt
  ```
- Alle Commits auflisten, ggf. nur von einer Datei:
  ```
  git log
  git log test.txt
  ```
- Einen Commit anzeigen (enthaltene Änderungen)
  ```
  git show 6e46237b
  ```
- Eine alte Version einer Datei zurückholen
  ```
  git checkout 6e46237b -- test.txt
  ```
- Doch wieder die neueste Version
  ```
  git reset HEAD test.txt
  git checkout -- test.txt
  ```

Ein öffentliches Git

- Üblich bei OpenSource-Projekten
- Eine begrenzte Anzahl von Personen darf committen (per SSH)
- Jeder darf lesen oder das Repository clonen (per HTTP)

Hosting eines öffentlichen Git

- Bei einem Anbieter im Internet, z. B. https://github.com/
- Selbst hosten, z. B. auf einem Root-Server beim Provider
  z. B. mit Gitorious – https://gitorious.org/gitorious/
  oder mit gitweb – als Paket in den meisten Distributionen enthalten

Optional: Teil 2 – Intermediate Level

- Tags
- Branches
- Merge
- gitk
- Pull Requests
- Patches

Tags

- Ein bestimmter Stand bekommt einen (sprechenden) Namen
- Erzeugen:
  ```
  git tag  -m "Version 1.0" V1.0 2fb76db4
  ```
- Später zurückholen:
  ```
  git checkout V1.0
  ```
- Und wieder zur aktuellsten Version:
  ```
  git checkout master
  ```
- Welche Tags gibt es im aktuellen Repository?
  ```
  git tag
  ```

Branches

- Verschiedene Entwicklungszweige existieren parallel nebeneinander
- Ggf. werden an mehreren davon Änderungen vorgenommen
- z. B. Entwicklungsversion (neuen Features werden implementiert) und stabile Version (nur Bugfixes werden eingepflegt)

Arbeiten mit Branches (1/2)

- Erzeugen:
  ```
  git branch Entwicklungszweig
  ```
- Branches anzeigen und wo stehen wir gerade?
  ```
  git branch
  ```
- Wechseln in einen anderen Branch:
  ```
  git checkout Entwicklungszweig
  git branch
  ```
- Änderung durchführen und committen:
  ```
  vim test.txt
  git commit -am "neues Feature eingebaut"
  ```

Arbeiten mit Branches (2/2)

- Auch im master-Branch eine Änderung:
```
git checkout master
vim test.txt
git commit -am "Bug fixed"
```
- Branches wieder zusammenführen:
```
git merge Entwicklungszweig
```
- Nicht mehr benötigten Branch löschen:
```
git branch -d Entwicklungszweig
```

Branches versus Tags

- Die Konzepte sind sehr ähnlich, wann nimmt man was?
- Genereller Vorschlag:
  - Ein Branch lebt, d. h. verändert sich im Lauf seines Lebens
  - Ggf. wird er auch wieder gelöscht
  - Tag wird einmalig gesetzt und ändert sich danach nie wieder
  - Je nach Projekt kann man auch definieren: Ein Tag darf grundsätzlich nie gelöscht werden

gitk

- Grafisches Tool, um Branches und Commits zu visualisieren
- Installieren und benutzen:

```
sudo aptitude install gitk
cd meinRepo/
gitk
```

Zusammenarbeit über Projektgrenzen hinweg

- Jemand ohne Commit-Rechte am Projekt-Repository möchte Änderungen beisteuern
- Pull Requests
  - Ein Committer wird gebeten sich Änderungen aus dem öffentlichen Repository des Contributors zu ziehen
- Patches
  - Contributor erstellt einen Diff (=strukturierte Beschreibung der Änderungen) zwischen seiner Version und der orginal Version und schickt diesen z. B. auf die Entwickler-Mailingliste des Projektes
- Für beides bietet git Unterstützung

Ausblick: Teil 3 – Expert Level

abgefahrenes Zeug

- Rebase
- Mehrere Remotes anbinden

Noch Fragen?

- Jetzt und hier
- Im Anschluß beim Bier
- Bei (fast) jedem Linux-Cafe Gluga-Stammtisch, Ubuntuusers-Stammtisch, ...
- Jederzeit auf der Gluga Users Mailingliste, siehe http://mailing.gluga.de/