git Workshop

Matthias Beyer

Hochschule Furtwangen University

matthias.beyer@hs-furtwangen.de
mail@beyermatthias.de

19. November 2013

Overview

Overview

2. Einfache Kommandos
   - Konfigurieren und ein Repo erstellen
   - Den Status abfragen und eine Datei zu git hinzufügen
   - Historie und Änderungen ansehen
   - Wer hat was geschrieben?
   - Änderungen rückgängig machen und zurücksetzen
   - Fragen und Antworten

Overview

Overview

Overview

5. Fortgeschrittene Kommandos
   - Entwicklunszweige umpflanzen
   - Entwicklunszweige interaktiv umpflanzen
   - Automatisiertes Fehlersuchen

Section 1

Einführung in Git

Warum Versionskontrolle

Warum Versionskontrolle?

Warum Versionskontrolle

- Einfache Verwaltung von Änderungen
  - Rückgängig machen von Änderungen
  - Wer hat was geschrieben
- Verteiltes arbeiten
- Fehlersuche

Subsection 1

Geschichte

Die Anfänge (1)

- (Kernel wurde mit Patchfiles versioniert)
- Kernel wurde mit BitKeeper versioniert
- Lizensänderungen bei BitKeeper

Die Anfänge (2)

- Linus Torvalds schreibt eigene Versionsverwaltung (2005, April)
- Nach wenigen Tagen "fertig"

Anforderungen

Torvalds wünscht sich...

- Unterstützung verteilter, BitKeeper-ähnlicher Arbeitsabläufe
- Sehr hohe Sicherheit
- Hohe Effizienz

Anforderungen

Der Kernel heute:

- 16,6 Mio Zeilen Quellcode in 37.547 Files
- ings. 47.552 Files
- 11.196 Entwickler
- 411.203 Änderungen

Anforderungen

Ein "normales"git-projekt: typo3

- 899.622 Zeilen Quellcode in 4.237 Files
- insg. 5146 Files
- 166 Entwickler
- 21.766 Änderungen

Namensgebung

I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'Git'.

1

[1] Linus Torvalds

Subsection 2

Was ist Git...

... für Anwender

Git ist ein Verteiltes (Quellcode-)Versionskontrollsystem, dass ...

- ... einfach zu lernen ist
- ... einen kleinen "Fußabdruck"hat
- ... schnell ist
- ... einfaches Branch-Management mit sich bringt
- ... verschiedene Workflows unterstützt
- ... kryptografische Sicherheit für die Versionshistorie bietet

... für Anwender

Git ist ein Verteiltes (Quellcode-)Versionskontrollsystem, dass ...

- ... einfach zu lernen ist
- ... einen kleinen "Fußabdruck"hat
- ... schnell ist
- ... einfaches Branch-Management mit sich bringt
- ... verschiedene Workflows unterstützt
- ... kryptografische Sicherheit für die Versionshistorie bietet

... für Anwender

Git ist ein Verteiltes (Quellcode-)Versionskontrollsystem, dass ...

- ... einfach zu lernen ist
- ... einen kleinen "Fußabdruck"hat
- ... schnell ist
- ... einfaches Branch-Management mit sich bringt
- ... verschiedene Workflows unterstützt
- ... kryptografische Sicherheit für die Versionshistorie bietet

... für Anwender

Git ist ein Verteiltes (Quellcode-)Versionskontrollsystem, dass ...

- ... einfach zu lernen ist
- ... einen kleinen "Fußabdruck"hat
- ... schnell ist
- ... einfaches Branch-Management mit sich bringt
- ... verschiedene Workflows unterstützt
- ... kryptografische Sicherheit für die Versionshistorie bietet

... für Anwender

Git ist ein Verteiltes (Quellcode-)Versionskontrollsystem, dass ...

- ... einfach zu lernen ist
- ... einen kleinen "Fußabdruck"hat
- ... schnell ist
- ... einfaches Branch-Management mit sich bringt
- ... verschiedene Workflows unterstützt
- ... kryptografische Sicherheit für die Versionshistorie bietet

... für Anwender

Git ist ein Verteiltes (Quellcode-)Versionskontrollsystem, dass ...

- ... einfach zu lernen ist
- ... einen kleinen "Fußabdruck"hat
- ... schnell ist
- ... einfaches Branch-Management mit sich bringt
- ... verschiedene Workflows unterstützt
- ... kryptografische Sicherheit für die Versionshistorie bietet

Subsection 3

Was Git nicht ist...

Was Git ist...

- kein Konkurrent zu Subversion
- kein Backup Tool
- kein Dropbox-Ersatz!
- nicht Netzwerktraffic-Intensiv
- nicht nur für die Kernel-Entwicklung
- nicht schwer aufzusetzen / zu benutzen
- nicht komplex.

Was Git ist...

- kein Konkurrent zu Subversion
- kein Backup Tool
- kein Dropbox-Ersatz!
- nicht Netzwerktraffic-Intensiv
- nicht nur für die Kernel-Entwicklung
- nicht schwer aufzusetzen / zu benutzen
- nicht komplex.

Was Git ist...

- kein Konkurrent zu Subversion
- kein Backup Tool
- kein Dropbox-Ersatz!
- nicht Netzwerktraffic-Intensiv
- nicht nur für die Kernel-Entwicklung
- nicht schwer aufzusetzen / zu benutzen
- nicht komplex.

Was Git ist...

- kein Konkurrent zu Subversion
- kein Backup Tool
- kein Dropbox-Ersatz!
- nicht Netzwerktraffic-Intensiv
- nicht nur für die Kernel-Entwicklung
- nicht schwer aufzusetzen / zu benutzen
- nicht komplex.

Was Git ist...

- kein Konkurrent zu Subversion
- kein Backup Tool
- kein Dropbox-Ersatz!
- nicht Netzwerktraffic-Intensiv
- nicht nur für die Kernel-Entwicklung
- nicht schwer aufzusetzen / zu benutzen
- nicht komplex.

Was Git ist...

- kein Konkurrent zu Subversion
- kein Backup Tool
- kein Dropbox-Ersatz!
- nicht Netzwerktraffic-Intensiv
- nicht nur für die Kernel-Entwicklung
- nicht schwer aufzusetzen / zu benutzen
- nicht komplex.

Was Git ist...

- kein Konkurrent zu Subversion
- kein Backup Tool
- kein Dropbox-Ersatz!
- nicht Netzwerktraffic-Intensiv
- nicht nur für die Kernel-Entwicklung
- nicht schwer aufzusetzen / zu benutzen
- nicht komplex.

Subsection 4

Des weiteren...

Fürs nächste mal...

Was nächstes mal benötigt wird:

- Git installiert (auf Windows mit entsprechender Kommandozeile)
- Einen Texteditor (Windows: Notepad, *nix: Vim, Vi, geany)
- cd, ls, mkdir, ...
- (evtl.) einen SSH-Key

Fragen?

Wer fragt, ist ein Narr für fünf Minuten. Wer nicht fragt, bleibt ein Narr für immer. [2]

[2] Aus China

Section 2

Einfache Kommandos

Subsection 1

Konfigurieren und ein Repo erstellen

Git Konfigurieren...

Es gilt: Nicht nur stur abtippen! Mitdenken!

Listing 1: git config

```
1 git config --global user.name Max Mustermann
  git config --global user.email mm@mustermail.tld
3 git config --global color.ui auto
```

... für Windows

```
1 git config --global core.editor notepad
  # bzw. kompletten Dateipfad
3 git config --global core.autocrlf true
```

... für den Pinguin und den Apfel

```
1 git config --global core.editor geany
  # oder vi, oder emacs, oder ...
3 git config --global core.autocrlf input
```

Ein repo erstellen

```
1 mkdir workshop
  cd workshop
3 git init
  ls
5 ls -a
```

Subsection 2

Den Status abfragen und eine Datei zu git hinzufügen

git status

```
git status
# Auf Branch master
#
# Initialer Commit
#
nichts zu committen (Erstellen/Kopieren Sie Dateien
und benutzen Sie git add zum Beobachten
```

git add

Mittels add fügt man eine Arbeitskopie dem “index” hinzu. Noch sind die Änderungen nicht im Repository!

```
1 echo 'Hallo Welt' >> erstedatei.txt
  git add erstedatei.txt
3 git status
```

git add

Git weiß jetzt, dass es die Datei beobachten soll, aber es hat noch keine Änderungen aufgenommen!

git commit

Eine Datei Committen heisst so viel wie die Änderungen kommentieren und somit im Repository verfügbar machen.

```
1 git commit erstedatei.txt
  git status
```

Subsection 3

Historie und Änderungen ansehen

git log

Die Historie kann man mit

```
git log
```

ansehen.
Man kann dem Log sehr viele Parameter übergeben, ich benutze meist:

```
1 git log --all --oneline --graph --decorate
  #oder
3 git log --oneline ---graph --decorate <von>...<bis>
```

Subsection 4

Wer hat was geschrieben?

git blame

Manchmal möchte man herausfinden, welche Zeile in einer Datei von wem wann wieso geändert wurde.

```
1 git blame erstedatei.txt
```

Zeigt aktuell nur einen Autor:

```
1 git blame erstedatei.txt
^8af25a5 (Matthias Beyer 2013-09-22 12:31:11 +0200 1)
```

Mehr Änderungen

An diesem Punkt ist es notwendig noch mehr Änderungen an der Datei vorzunehmen. Schreibe ein paar Zeilen (bitte nicht einfach “asdfasdf”) und Commite diese. Möglichst in mehreren Commits!

git show

Der HEAD zeigt immer auf das letzte Commit (des aktuellen Branches)

```
git show HEAD
git show HEAD~1
git show <commit-hash>
```

Seh dir nun bitte das drittletzte Commit an!

git diff

Das diff Kommando zeigt den Unterschied zwischen zwei Commits an

```
1 git diff <commit-hash>...<commit-hash>
  git diff <commit-hash>...
3 git diff HEAD~1...HEAD
```

Zeige bitte den Unterschied zwischen dem drittletzten Commit und dem vorletzten Commit an!

Subsection 5

Änderungen rückgängig machen und zurücksetzen

Änderungen ändern

Es gibt zwei Arten, eine Änderung rückgängig zu machen

- git revert - Invertiert Commits
- git reset - Löscht Commits

Kann sich jemand denken, welche wann eingesetzt wird?

git revert

```
1 git revert <commit-hash>
```

Reverted bitte nun das vorletzte Commit!

git reset

```
1 git reset
```

Setzt nun bitte zurück auf das Commit, bevor wir das Revert angewendet haben!

Subsection 6

Fragen und Antworten

Fragen?

Wenn du eine weise Antwort verlangst, mußt du vernünftig fragen. [3]

[3] Goethe

Section 3

Weitere Basiskommandos

Branching und Merging

In diesem Kapitel:

- Was sind Branches
- Wie benutze ich Branches
- Wie binde ich Branches ineinander ein (mergen)

Subsection 1

Entwicklungszweige

Branches sind Zweige

→ Siehe Tafel

Subsection 2

Entwicklungszweige erstellen und benutzen

Branches ansehen/erstellen

```
1 git branch
```

```
1 git branch neuerbranch
```

→ Merke: Branches sind nur Markierungen für Commits!

Erstelle einen neuen Branch 'neuerbranch' auf dem aktuellen Commit.

git checkout

Noch sind wir nicht auf dem Branch!

```
1 git checkout neuerbranch
```

Mit git checkout lassen sich übrigens auch Commits auschecken!
→ Erstelle einen Branch 'andererbranch' auf dem vorletzten Commit!
(Erinnerung: git log) Gehe danach zurück auf den Branch, den du auf dem master-branch erstellt hast!

Subsubsection 1

Entwicklungszweige zusammenführen

Mergen ohne Konflikte (1)

Erstelle nun ein paar Commits auf deinem neuen Branch.

Mergen ohne Konflikte (2)

```
1 git checkout master
  git merge neuerbranch
```

Mergen ohne Konflikte (3)

Erkläre, was passiert ist. Du kannst dir gerne den Log ansehen:

```
git log --all --decorate --graph --oneline
```

Kannst du dir denken, warum das so schnell ging?

Mergen mit Konflikten (1)

Was passiert, wenn wir Zeilen ändern, die auf einem anderen Branch schon geändert wurden - nur anders?
$\rightarrow$ Konflikte entstehen!

Mergen mit Konflikten (2)

Erzeuge einen Konflikt:

```
1 git checkout master
  <Modifiziere deine Dateien>
3 git commit <datei>
  git checkout andererbranch
5 <Modifiziere deine Dateien>
  git commit <datei>
```

Mergen mit Konflikten (3)

Merge nun diesen Branch ('andererbranch') in den Master:

```
git checkout master
git merge andererbranch
# automatischer Merge von hallowelt.txt
# KONFLIKT (Inhalt): Merge-Konflikt in hallowelt.txt
# Automatischer Merge fehlgeschlagen; beheben Sie die
# dann das Ergebnis.
```

→ Es sollte beim Mergen nun ein Fehler geworfen werden! Der Merge ist jetzt noch nicht fertig, und Konflikte müssen von Hand gelöst werden!

Mergen mit Konflikten (4)

Jetzt muss der Merge Konflikt aufgelöst werden.

Git hat jetzt die entsprechende Datei bearbeitet:

```
<<<<<<< HEAD
Hallo Kurs!
=======
Hallo Welt!
>>>>>>> andererbranch
```

Mergen mit Konflikten (5)

Wir müssen also etwas in Ordnung bringen:

```
1 Hallo Kurs!
```

Mergen mit Konflikten (6)

Und das Ergebnis hinzufügen:

```
1 git add hallowelt.txt
  git commit
```

Git schlägt uns jetzt eine Commit Message vor, die wir so übernehmen können.

Mergen mit Konflikten (7)

Wenn wir jetzt den Log ansehen, sehen wir dass die Commits gemerged wurden:

```
git log --oneline --graph --decorate --all
```

Subsubsection 2

Git die arbeit machen lassen

Automatisches Mergen (1)

Git kann verschiedene Merge-Strategien anwenden:

- –strategy recursive -X theirs
- –strategy recursive -X ours
- –strategy octopus

Automatisches Mergen (2)

Ausserdem kann git verschiedene Optionen zum Mergen anwenden:

- –ff-only
- –no-ff
- –edit (eigentlich eine Option zum committen)

Fragen?

Es gibt keine dummen Fragen, nur dumme Antworten! [4]

[4] Irgendjemand

Section 4

Weitere Basiskommandos

Mit einem Remote arbeiten, Tagging, Rebasing und interaktives Rebasing

- Remotes anlegen und abholen
- Remotes updaten
- Releases taggen
- Branches umpflanzen
- Interaktives rebasing
- Bisecting

Subsection 1

Mit (einem) Server arbeiten

Einen Remote Server anlegen (1)

→ github.com
→ git.informatik.hs-furtwangen.de
→ Eigene git-server (später mehr)

Einen Remote Server anlegen (2)

```
1 git remote add <name> <url>
```

```
1 git remote add meinremote
      git@github.com:matthiasbeyer/hfu-gitworkshop.git
```

Einen Remote Server anlegen (3)

- fetch
- pull
- push

Einen Remote Server anlegen (4) - Fetch

Mit

```
git fetch meinremote
```

holt man Commits vom Server ab.

Einen Remote Server anlegen (5) - Pull

```
1 git pull meinremote <branch>
```

Einen Remote Server anlegen (6) - Push

```
1 git push meinremote meinbranch
```

→ Natürlich nur, wenn man Schreibrechte hat!

Einen Remote Server anlegen (7)

Gut zu wissen:

- Remote gibt es auch Branches
- Diese sind jetzt lokal verfügbar $\rightarrow$ behandelbar wie normale Branches

Section 5

Fortgeschrittene Kommandos

git rebase und git bisect

Diese Kapitel behandelt fortgeschrittenere Kommandos:

- git rebase
- git bisect

Subsection 1

Entwicklunszweige umpflanzen

Branches umpflanzen (1)

Anwendungsfall:
$\rightarrow$ Features auf einer neuen Version basieren lassen.

Branches umpflanzen (2)

```
1 git rebase <worauf>
```

Branches umpflanzen (3)

Aufgabe:

Erstelle einen neuen Branch auf einem älteren Commit. Schreibe Mindestens zwei Commits und pflanze diesen Branch auf den neusten Master um.

Subsection 2

Entwicklunszweige interaktiv umpflanzen

Interaktives umpflanzen (1)

Interaktives umpflanzen heisst:

- Commits neu anordnen
- Commits splitten
- Commits zusammenführen
- Commits löschen
- Commits ändern

Interaktives umpflanzen (1)

Interaktives umpflanzen heisst:

- Commits neu anordnen
- Commits splitten
- Commits zusammenführen
- Commits löschen
- Commits ändern

Interaktives umpflanzen (1)

Interaktives umpflanzen heisst:

- Commits neu anordnen
- Commits splitten
- Commits zusammenführen
- Commits löschen
- Commits ändern

Interaktives umpflanzen (1)

Interaktives umpflanzen heisst:

- Commits neu anordnen
- Commits splitten
- Commits zusammenführen
- Commits löschen
- Commits ändern

Interaktives umpflanzen (1)

Interaktives umpflanzen heisst:

- Commits neu anordnen
- Commits splitten
- Commits zusammenführen
- Commits löschen
- Commits ändern

Interaktives umpflanzen (2)

```
1 git rebase -i <worauf>
```

Kann auch sein:

```
1 git rebase -i HEAD~5
```

Interaktives umpflanzen (2)

Optionen für interaktives rebasing:

- edit
- fixup
- squash
- reword

Interaktives umpflanzen (3)

Aufgabe:

Schreibe einen Branch mit mindestens drei Commits. Squashe davon zwei ineinander und schreibe die Commit-Message vom dritten um.

Subsection 3

Automatisiertes Fehlersuchen

Fehler suchen mit git bisect (1)

Mittels git bisect kann man:

- Commits mit Binärsuche durchlaufen
- Fehler finden

Fehler suchen mit git bisect (2)

```
1 git checkout HEAD~5
  git bisect start
3 git bisect good
  git checkout master
5 git bisect bad
```

Fragen?

Über Fragen, die ich nicht beantworten kann, zerbreche ich mir nicht den Kopf. [5]

[5] Konrad v. Zuse

Section 6

Wissenswertes

Wissenswertes über git

- Bash, Vim, GUIs
- Workflows
- Sonstiges

Subsection 1

Git in die Umgebung integrieren

Editoren

Plugins für...

- Vim
- Eclipse
- Kate
- Sublime Text 2
- NetBeans
- Emacs
- ...

Oberflächen

- gitk oder gitx
- tig
- giggle
- git-cola
- TortoiseGit
- ...

Programmiersprachen

Interfaces zu git sind verfügbar für:

- C
- Perl
- Python
- Ruby
- Java
- PHP
- Haskell
- ...

Subsection 2

Workflows

No-Gos

Absolute No-Gos:

- Tags verschieben
- Veröffentlichte Commits ändern:
  - rebase (normal und interaktiv)
  - branches löschen
  - force push

No-Gos

Absolute No-Gos:

- Tags verschieben
- Veröffentlichte Commits ändern:
  - rebase (normal und interaktiv)
  - branches löschen
  - force push

No-Gos

Absolute No-Gos:

- Tags verschieben
- Veröffentlichte Commits ändern:
  - rebase (normal und interaktiv)
  - branches löschen
  - force push

No-Gos

Absolute No-Gos:

- Tags verschieben
- Veröffentlichte Commits ändern:
  - rebase (normal und interaktiv)
  - branches löschen
  - force push

Workflows

Es gibt verschiedene Workflows:

- Zentralisiert
- Feature Branching
- Gitflow
- Forking

Workflows - Zentralisiert

- Gleich wie Subversion
- Ein remote Repository

Workflows - Feature Branching

- Jedes Feature in einem eigenen Branch
- Jeder Fix in einem eigenen Branch

Workflows - Gitflow

- Ein Release-Branch
- Ein Develop-Branch
- Einfach kombinierbar mit Feature Branching

Workflows - Forking

- Jeder Entwickler hat ein remote Repository
- Jeder forkt von jedem, wenn gewollt

Subsection 3

Sonstiges

Was es sonst noch so gibt

- Commits per Email versenden
- Commits mit PGP/GPG unterschreiben
- Repositories teilen/zusammenführen
- git server
  - gitphp
  - cgit
  - gitorious
  - gitlab
  - gitweb

Nachschlagewerke

Besonders zu Empfehlen:

- book.git-scm.com
- de.gitready.com

Und natürlich:

- manpages
- google

Fragen?

Ob ein Mensch klug ist, erkennt man an seinen Antworten. Ob ein Mensch weise ist, erkennt man an seinen Fragen. [6]

[6] Nagib Machfus

Bei späteren Fragen

→ github.com/matthiasbeyer

→ mail@beyermatthias.de

→ Anquatschen

The End

Wenn es keine Fragen mehr gibt...

ist dass das Ende.