Nicht alltägliche Git Funktionen

Sujeevan Vijayakumaran
zudʒi:væn̩ vɪdʒæjekumaræn

Twitter: @svijee
Blog: svij.org

19. August 2017

Alle fit in Git?

Git @ FrOSCon

- **Samstag, 17:45 - 18:45, Saal 4**: Nicht alltägliche Git Funktionen
- **Sonntag, 14:00 - 18:00, C116**: Git Workshop für Einsteiger

Über den Talk

- Nennung und Erläuterung (hoffentlich?) unbekannter Git Funktionen
- keine Funktion, die man zwangsläufig täglich braucht
- nicht (unbedingt) für Git-Anfänger
- gute Git-Kenntnisse vorausgesetzt
- altes Bekanntes mit nützlichen Verbinden
- ausschließlich Git für die Kommandozeile

Vorausgesetzte (alltägliche) Git Funktionen

- `git add` – Änderungen stagen
- `git commit` – Commits erzeugen
- `git checkout` – Branches auschecken
- `git push` – Commits & Branches pushen
- `git pull` – Änderungen herunterladen und mergen
- `git merge` – Branches mergen
- `git rebase` – Branches rebasen
- `git status` – Status-Abfrage
- `git log` – Log ansehen

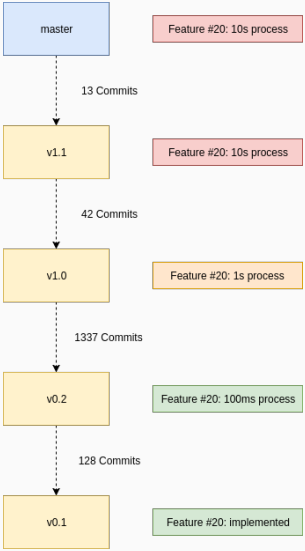
`git bisect`

Finde deine Fehler!

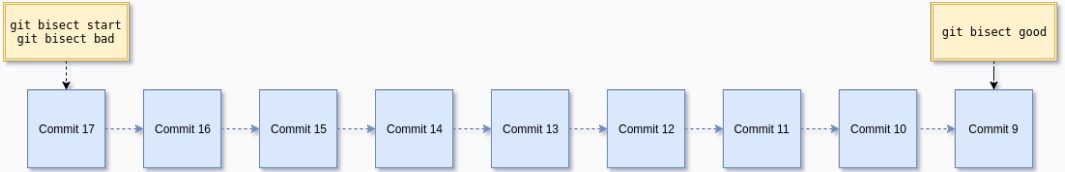
`git bisect` – Fehler finden

- Nicht debugbarer, aber reproduzierbarer Fehler zwischen zwei Versionen
- Binary-Search über Commits mittels `git bisect`

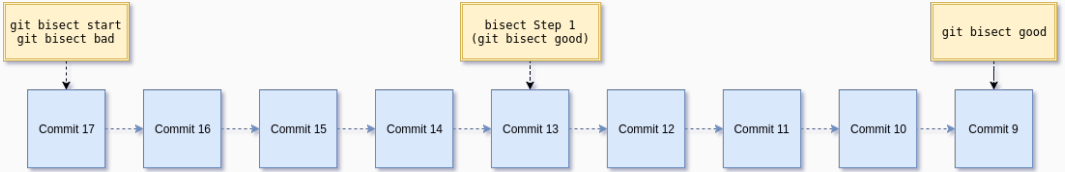
`git bisect` – Fehler finden



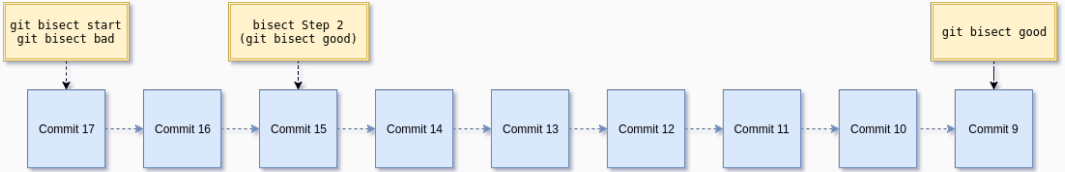
`git bisect` – Fehler finden



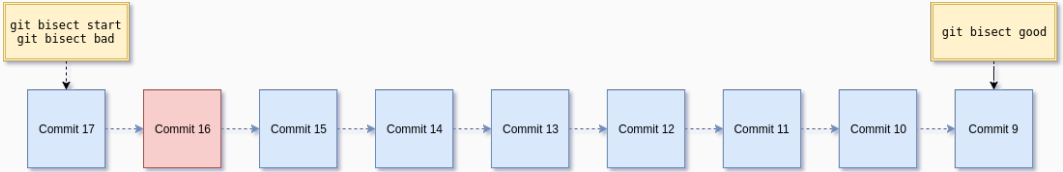
`git bisect` – Fehler finden



`git bisect` – Fehler finden



`git bisect` – Fehler finden



`git bisect` – Fehler finden

- Binäre Suche über die Commits
- Projekt **muss** compilierbar/ausführbar/testbar sein
- per Script automatisierbar

`git bisect` – Fehler finden

- Binäre Suche über die Commits
- Projekt **muss** compilierbar/ausführbar/testbar sein
- per Script automatisierbar

Pro Tipp:

- `git bisect` hilft nur, wenn die Commits nachvollziehbar sind
- kleine überschaubare Commits wichtig (auch für das Review)
- Commits so klein wie möglich und so groß wie nötig

`git add -p`

Häppchen für Häppchen in ein Commit

`git add -p` – Änderungen häppchenweise stagen

- **Ziel**: Nicht alle Änderungen einer Datei stagen
- **Interaktives Stagen**: Mit `git add -p`
- **Grund**: Möglichst kleine, logisch getrennte Commits erzeugen
- **Terminologie**: Hunk (Deutsch: Stück, Brocken)

`git add -p` – Modi

- **y** - stage this hunk
- **n** - do not stage this hunk
- **q** - quit; do not stage this hunk or any of the remaining ones
- **a** - stage this hunk and all later hunks in the file
- **d** - do not stage this hunk or any of the later hunks in the file
- **g** - select a hunk to go to
- **/** - search for a hunk matching the given regex
- **j** - leave this hunk undecided, see next undecided hunk
- **J** - leave this hunk undecided, see next hunk
- **k** - leave this hunk undecided, see previous undecided hunk
- **K** - leave this hunk undecided, see previous hunk
- **s** - split the current hunk into smaller hunks
- **e** - manually edit the current hunk

`git alias`

Aliasse setzen und nutzen

`git alias` – Aliasse setzen und nutzen

Kein Hexenwerk

- **Problem**: Lange Befehle
- **Lösung**: Alias setzen
- **Befehl**: `git config --global alias.co checkout`

`git alias` – Aliasse setzen und nutzen

```
$ git config --global alias.stsh 'stash --keep-index'
$ git config --global alias.staash 'stash --include-untracked'
$ git config --global alias.staaash 'stash --all'
```

`git alias` – Aliasse setzen und nutzen

```
$ git log --pretty=oneline --grep #123
$ git config --global alias.loggrep "log --pretty=oneline --grep"
```

`git log`

Mehr aus dem Log herausholen

`git log` - Mehr aus dem Log herausholen

```
$ git log -3
$ git log --since=2017-01-01
$ git log --since=2weeks
$ git log --committer=svij --since=2weeks
```

```
git diff --word-diff
```

Wortweises diff mit word-diff

git diff --word-diff

```
$ git diff
$ git diff --word-diff
```

git diff --word-diff

```
$ git log [---pretty=online-]{+--pretty=oneline+} --grep #123
```

`git grep`

Datei-Inhalte suchen mit `git grep`

`git grep` - Datei-Inhalte suchen

```
$ git grep "<body>" HEAD~1
HEAD~1:index.html:
 <body>
```

git log -G

```
$ git log -G "<body>"
commit 3477f6ff2d62de097d76a818ca222b9dfa9a7d57
Author: Sujeevan Vijayakumaran <mail@svij.org>
Date:
 Thu Nov 12 19:38:34 2015 +0100
index.html hinzugefügt.
```

`git rebase -i`

Der interaktive Rebase

`git rebase -i` – Interaktives Rebasing

- **Problem**: Commit-Reihenfolge verschieben, Commit-MSG verändern …
- **Lösung**: `git rebase -i HEAD~2`
- **Zusatz**: Neuschreiben der Historie

`git rebase -i HEAD~2` – Interaktives Rebasing

Editor öffnet sich:

```
pick da62305f Unified Logging
pick 18691691 Removed "if not nocache" from Wiki PageManager.
```

`git rebase -i HEAD~2` – Interaktives Rebasing

```
p, pick = Commit verwenden
r, reword = Commit verwenden, aber Commit-Beschreibung bearbeiten
e, edit = Commit verwenden, aber zum Nachbessern anhalten
s, squash = Commit verwenden, aber mit vorherigem Commit vereinen
f, fixup = wie "squash", aber diese Commit-Beschreibung verwerfen
x, exec = Befehl (Rest der Zeile) mittels Shell ausführen
d, drop = Commit entfernen
```

`git rebase -i`

Zu beachten:

- Historie wird neu geschrieben!
- Force-Push wird benötigt!
- Nur auf Feature-Branches wo man allein drauf arbeitet!

`git reflog`

Das Medikament gegen „Ich hab nix getan!1elf!“

`git reflog` – Ausgeführte Aktionen anzeigen

- **1. Problem**: „Kaputtes“ Repository (etwa verlorene Commits)
- **2. Problem**: Der Mensch weiß nicht, was er getan hat
- **3. Problem**: Der supportende Mensch, weiß es auch nicht

`git reflog` - Ausgeführte Aktionen anzeigen

Was zeigt es an?

- Checkouts
- Commits
- Push
- Pull
- Rebase
- ... „alles“

`git reflog` - Ausgeführte Aktionen anzeigen

```
2c1978 (HEAD -> master, origin/master) HEAD@{0}: checkout: …
807b45a (origin/fix-refs, fix-refs) HEAD@{1}: commit: Fix refs
a2c1978 (HEAD -> master, origin/master) HEAD@{2}: checkout: …
a2c1978 (HEAD -> master, origin/master) HEAD@{3}: pull: Fast-forward
017d22f HEAD@{4}: checkout: moving from PeceptronIsAFF to master
089240c (origin/PeceptronIsAFF, PeceptronIsAFF) HEAD@{5}: rebase finished
089240c (origin/PeceptronIsAFF, PeceptronIsAFF) HEAD@{6}: rebase: …
```

`git gc`

Unerreichbares aufräumen

`git gc` – Garbage Collection

- **1. Problem**: „dangling“ Commits
- **2. Problem**: Speicherplatzverschwendung
- **3. Problem**: unerreichbare Commits könnten noch gebraucht werden

`git gc` – Garbage Collection

- **1. Problem**: „dangling“ Commits
- **2. Problem**: Speicherplatzverschwendung
- **3. Problem**: unerreichbare Commits könnten noch gebraucht werden
- **Lösung**: Garbage Collection räumt regelmäßig auf.

`git gc` – Garbace Collection

- Komprimierung von Revisionen
- Entfernung von unerreichbaren Objekten
- Option: Ab bestimmtes Datum
- **Ziel**: Performance-Optimierung und Speicherplatz-Optimierung

`git blame`

„Ich war das nicht!1!elf! … Achso doch, das war ich.“

`git blame` – den Schuldigen finden

- **Problem**: Feature funktioniert nicht
- **Fragen**:
  - Wer?
  - Wann?
  - Warum?
  - Was hängt da zusammen?

`git blame` – den Schuldigen finden

Git Blame zeigt folgendes an:

- Commit-ID
- Autor
- Datum + Uhrzeit
- Zeilennummer
- Zeileninhalt

`git filter-branch`

Neuschreiben der kompletten Historie

`git filter-branch` – Komplette Historie neu schreiben

- **1. Problem**: Zugangsdaten im Quellcode
- **2. Problem**: Zugangsdaten sind in jeder Revision drin
- **Lösung**: Komplette Historie neuschreiben

`git filter-branch` – Komplette Historie neu schreiben

- `--tree-filter`: Im Datei-Baum ein Kommando ausführen
- `--index-filter`: Wie `tree-filter`, checkt nur nicht den Baum aus
- `--msg-filter`: Verändern der Commit-Messages
- `--subdirectory-filter`: Historie eines Unterverzeichnisses extrahieren
- … und noch einige mehr

`git filter-branch` – Komplette Historie neu schreiben

- **Wichtig**: Historie wird neu geschrieben
- nicht auf öffentlichen Repositorys durchführen
- mit Mitarbeitern kommunizieren
- **Alternatives Tool**: BFG Repo-Cleaner

`git worktree`

Mehrere Arbeitsverzeichnisse mit einem Repository

`git worktree` – Ein Repo - mehrere Worktrees

- **Problem**: Mehrere gleichzeitige Arbeiten an verschiedenen Branches
- **Mögliche Lösung**: Repository mehrfach klonen
- **Bessere Lösung**: `git worktree` nutzen

`git worktree` – Ein Repo - mehrere Worktrees

- ein Repository mit mehreren Arbeitsverzeichnissen
- ein Arbeitsverzeichnis (Worktree) entspricht einem Branch
- nur eine Repository Einstellung wird benötigt

`git diff`

Diff von Binärdateien

`git diff` – Diesmal mit Binärdateien

- **Problem**: Git zeigt das Diff nur für Text-Dateien an
- **Lösung**: Externe Tools für ein Diff von Binärdateien einklinken

`git diff` – Diesmal mit Binärdateien

- **odt2txt**: Konvertierung von odt Dokumenten
- **pdftotext**: Konvertierung von pdf Dokumenten
- **usw...**

Eintrag in `.gitattributes` und `.gitconfig` notwendig.

`.gitattributes`

Attribute für dein Repository

`.gitattributes`

… behandelt die Dateien im Repository anders

- Definierung von Binärdateien
- Definierung der Zeilenenden (CRLF vs. LF)
- **Wo?**: `.gitattributes`

Was fehlt?

Verlosung: Git-Buch

- Verlag: mitp, 2016, (Preis: 29,99€ (Buch), 25,99€ (Ebook))
- Buch basiert auf Tutorial auf meinem Blog **svij.org**

Fragen?

Vielen Dank für die Aufmerksamkeit!

Fragen, Feedback und Fehler gerne per E-Mail an mail@svij.org.

Blog: svij.org

Twitter: @svijee

42