Git - Grundlagen und Anwendungen

Simon Fromme

9. Februar 2017

Tralios IT GmbH

Inhaltsverzeichnis

Git Interna

- Ein Git-Repository “from Scratch”

Fortgeschrittene Git Befehle

- git blame
- git cherry pick
- git reset
- git rebase
- git reflog
- squashing Commits

Git Workflows

- Centralized Workflow
- Feature Brance Workflow
- Gitflow Workflow
- Forking Workflow

Diverses

- Wie schreiben wir eine Commit-Message?
- 7 Regeln für eine gute Commit-Nachricht

Bildquellen

Git Interna

Git Interna

Ein Git-Repository “from Scratch”

siehe Vorführung...

Git Objekte (Blob)

Speichere einen Text-String in der Git Datenbank

```
$ echo -n 'Tralios IT GmbH' | git hash-object -w --stdin
  bb030efe0cb9955b1caf199f2476cb8094cad015
```

Speichere den Inhalt einer Datei in der Git Datenbank

```
$ echo "This is some file content" > test.txt
$ git hash-object -w text.ttx
  482d9ac465d65447f03634287103 2ad6add000af
```

```
$ find .git/objects -type f
  .git/objects/bb/030efe0cb9955b1caf199f2476cb8094cad015
  .git/objects/48/2d9ac465d65447f03634287103 2ad6add000af
```

Git erzeugt eine neue Datei, deren Name der SHA1-Hash des Strings plus eines Headers ist und deren Inhalt Zlib komprimiert ist.

All das speichert nur Inhalte und keine Dateinamen!

Git Objekte (Tree)

Erzeuge ein Tree-Objekt, das auf die zuvor erstellten Blobs `bb030efe0...` verweist:

```
$ git update-index --add --cacheinfo 100644 \
  bb030efe0cb9955b1caf199f2476cb8094cad015 tralios.txt
$ git update-index --add --cacheinfo 100644 \
  482d9ac465d65447f036342871032ad6add000af test.txt

$ git write-tree
  b5a509c6d89995ac4c4f3f3bbcf42c12706b9cc4
```

SHA1-Hashes müssen nur solange spezifiziert werden, bis die eindeutig sind

```
$ git cat-file -p b5a509c6d
  100644 blob 482d9ac465d65447f036342871032ad6add000af test.txt
  100644 blob bb030efe0cb9955b1caf199f2476cb8094cad015 tralios.txt
```

Git Objekte (Comitt)

```
$ echo "first commit" | git commit-tree b5a509c6d
  a830fd3d438e47226d4577bc280bdca804a3d06a
```

Berechnung des Object-Hashes

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-
3
4   import hashlib
5   import zlib
6   import os
7
8   content = 'Tralios IT GmbH'
9   header = "blob {}\0".format(len(content))
10  store = head + header
11
12  hashed = hashlib.sha1(store).hexdigest()
13  compressed = zlib.compress(store)
14
15  # path to store the blob object
16  path = os.path.join(".git/objects",
17                      hashed[:2],
18                      hashed[2:])
19
20  # create directory if necessary and write
21  # zlib compressed content to file
22  if not os.path.exists(os.path.dirname(path)):
23      os.makedirs(os.path.dirname(path))
24
25  with open(path, 'w') as f:
26      f.write(compressed)
```

Fortgeschrittene Git Befehle

Fortgeschrittene Git Befehle

git blame

Für jede Zeile in einer Datei, zeigt `git blame <file>` den letzten Commit an, der die Zeile bearbeitet hat.

`git show <commit>` zeigt den dazugehörigen Commit an.

Fortgeschrittene Git Befehle

git cherry pick

“Whoops.” Ausversehen zu master commited, obwohl auf den feature branch commited werden sollte.

```
git checkout feature; git cherry-pick <commit> ..
```

erzeugt einen neuen Commit auf dem feature Branch, der auf dem Original-Commit von master basiert.

Fortgeschrittene Git Befehle

git reset

Setzt den Branch Pointer zurück, der auf den gegenwärtigen Commit auf einem Branch zeigt.

```
$ git checkout master
$ git reset --hard
```

`HEAD`: gegenwärtiger Commit
`HEAD^`: vorheriger Commit
`HEAD~5`: fünf Commits vorher

Fortgeschrittene Git Befehle

git rebase

Z.B. master hat sich geändert, seit angefangen wurde, an Änderungen im feature Branch zu arbeiten. Am besten `rebase` statt `merge`.

```
$ git checkout feature
$ git rebase master
```

Am besten: History nur für Commits ändern, die noch nicht gepushed wurden.

Fortgeschrittene Git Befehle

git reflog

Enthält Commits, die zuletzt referenziert wurden.

1. Finde den gesuchten Commit
2. `checkout` den Commit und stelle sicher, dass alle stimmt.
3. mintinlinebashreset den Branch-Pointer auf den Commit

```
$ git checkout 1f732a
$ git reset --hard 1f732a1
```

Am besten: History nur für Commits ändern, die noch nicht gepushed wurden.

Fortgeschrittene Git Befehle

squashing Commits

“Verbessere” den letzten Commit.

```
$ git add missing-file.txt
$ git commit --amend
```

Ersetzt den letzten Commit durch den “verbesserten” Commit.
Auch mit

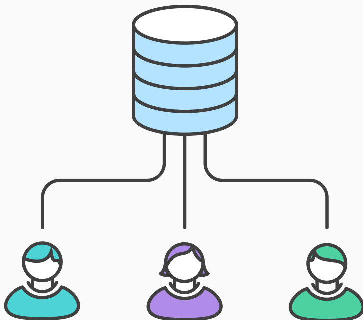
```
$ git rebase --interactive
```

möglich.

Git Workflows

Git Workflows

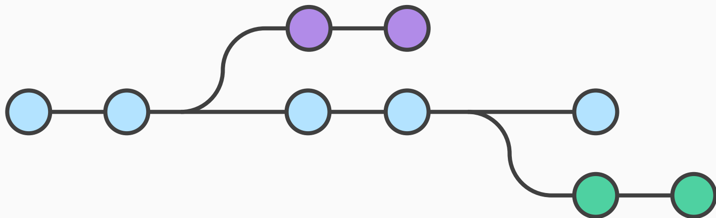
Centralized Workflow



- ein einziger master Branch
- Bearbeite Dateien und Commite Änderungen lokal
- Um Änderungen zum offiziellen Repository hinzuzufügen, verwende `push`
- Bei Konflikten: `fetch` von neuen Commits der zentralen Repos und `rebase` der lokalen Änderungen darauf

Git Workflows

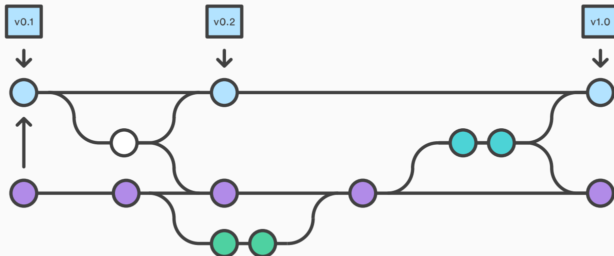
Feature Brance Workflow



- Alle Features bekommen ihren eigenen Branch (nicht master Branch)
  → master enthält nie “kaputten” Code
- merge-requests um Diskussionen über Änderungen/fertige Features zu starten und um feature in master zu mergen.
  → Code-Review durch Andere
  → erleichtert durch Gitlab-Interface
- feature Branches sollten auch zum zentralen Repository gepushed werden

Git Workflows

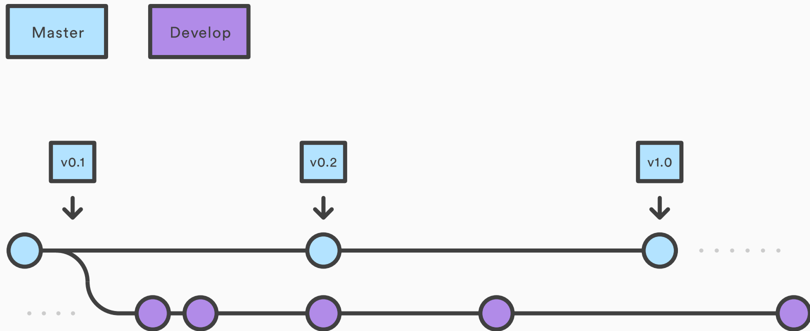
Gitflow Workflow



striktes Branching-Modell um Produkt-Release

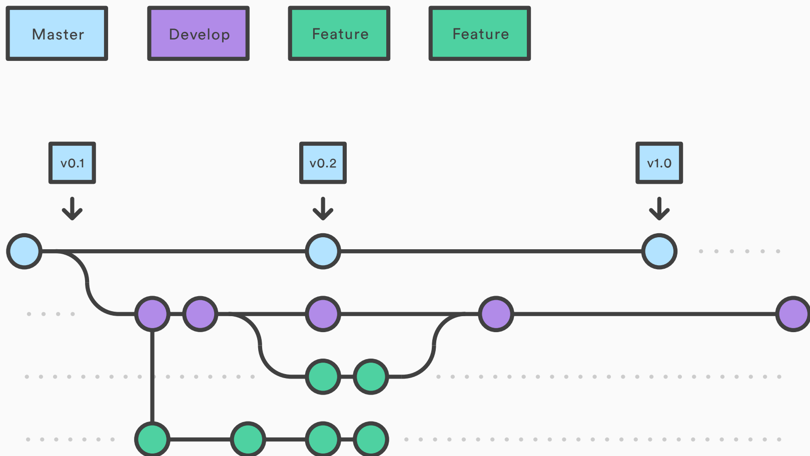
- keine neuen Konzepte im Vergleich zu "Feature-Branch-Workflow", aber spezielle Rollen für einzelne Branches und Definition, wie diese interagieren.
- master Branch: Offizielle Release History des Projektes
- deveop Branch: Integrations-Branch für alle Features

Gitflow



- jedes Feature auf eigenem Branch
- Aber: Branching von develop als “Eltern-Branch”
- Features sollen nicht direkt mit master interagieren
- fertiges Features wird in develop gemerged

Gitflow



- Ein Release wird auf einem eigenen $\mathrm{release}$ Branch vorbereitet
- Dieser wird anschließend auch in $\mathrm{develop}$ gemerged
- Release bekommt Tag

Gitflow

Master

Release

Develop

Feature

Feature

v0.1

v0.2

v1.0

- Maintenance oder Hotfix Branch um schnell Bug zu beheben
- Wird von master gebrancht
- Zurück in master und develop gemerged
- Neue Versionsnummer für Master

Gitflow

v0.1

v0.2

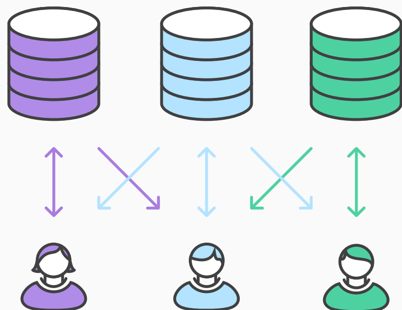
v1.0

Git Workflows

Forking Workflow

Forking Workflow

- Kein alleiniges Server-Side Repository als "zentrale Codebase"
- **Jeder** Entwickler bekommt ein server-side Repository
  → zwei Repositories pro User (*server-side* und *local*)



- Nur der Maintainer kann zum *offiziellen* Repository pushen
- Andere Entwickler schicken diesem **Pull-Requests**

Diverses

Diverses

Wie schreiben wir eine Commit-Message?

- **Style**: Sprache, Groß-/Kleinschreibung, Grammatik, Markup-Syntax, Zeichensetzung (mit/ohne Punkt)...
- **Inhalt**: Welche Informationen soll eine Commit-Message zusätzlich zur ersten Zeile beinhalten? Was soll sie *nicht* beinhalten?
- **Metadaten**: Sollen Issue-Tracing IDs, Pull-Request Nummern, ... mit aufgenommen werden?

Diverses

7 Regeln für eine gute Commit-Nachricht

1. Trennung von Titel und Body der Commit-Nachricht durch eine Leerzeile
2. Titel darf max. 50 Zeichen haben
3. Großschreibung des Titels
4. Titel wird *nicht* durch einen Punkt beendet
5. Imperativer Commit-Titel
6. Wrapping des Textkörpers bei 72 Zeichen
7. Benutze den Textkörper, um zu erklären *was* and *warum* und nicht *wie*

Bildquellen

Das git-Logo auf der Titelseite ist der Webseite https://git-scm.com/downloads/logos entnommen und steht unter der *Creative Commons Attribution 3.0 Unported License*. Veränderungen an der Grafik wurden nicht vorgenommen.

Die Grafiken auf den Seiten 15, 17, 19, 21, 23, 25, 27 und 28 sind der Webseite https://www.atlassian.com/git/tutorials/comparing-workflows entnommen und stehen unter unter der *Creative Commons Attribution 2.5 Australia Lizence*. Veränderungen an den Grafiken wurden nicht vorgenommen.