

Git-Workflows im Alltag

Valentin Haenel

Freelance Consultant and Software Developer

<http://haenel.co>

@esc_

2015-03-21 @ CLT



Version: v0.3.0

<https://github.com/esc/clt-2015-git-workflows>

This work is licensed under the *Creative Commons Attribution-ShareAlike 4.0 License*.

- Git-Workflows?
- Folge von Kommandos
- Nutzung von Branches und Tags

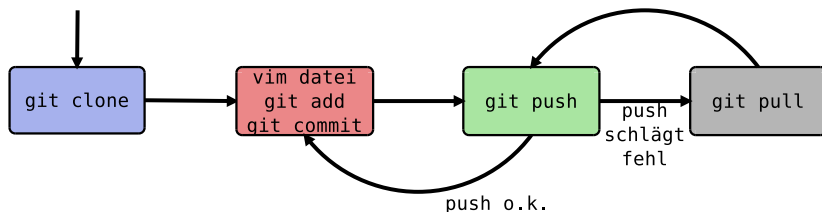
Übersicht

- 1 Intro
- 2 Push 'n' Pull
- 3 Github-Flow
- 4 Rebase 'n' Force-Merge
- 5 Gitflow
- 6 Andere
 - Gitworkflows
 - Gerrit Workflow
- 7 Outro

Übersicht

- 1 Intro
- 2 Push 'n' Pull
- 3 Github-Flow
- 4 Rebase 'n' Force-Merge
- 5 Gitflow
- 6 Andere
 - Gitworkflows
 - Gerrit Workflow
- 7 Outro

Push 'n' Pull Workflow



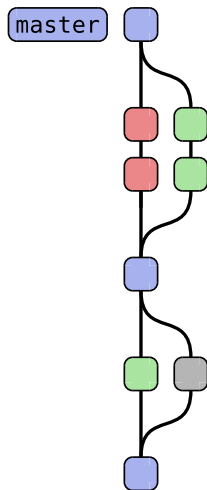
❶ Lokale Änderungen

- `vim datei`
- `git add datei`
- `git commit -m "msg"`

❷ Änderungen veröffentlichen

- `git push`
- Wenn push fehlschlägt:
- `git pull`, dann `git push`

Push 'n' Pull – Resultat



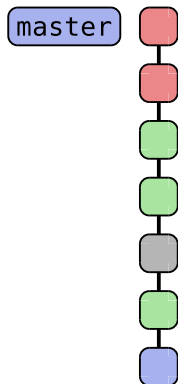
- Vorteile

- Leicht für Anfänger
- Nur weniger Kommandos

- Nachteile

- Es entstehen Merge-Commits
- »Aber wir arbeiten doch alle auf master?!«
- Rebase ist eine Option (für Anfänger?)

Push 'n' Pull – Resultat mit --rebase

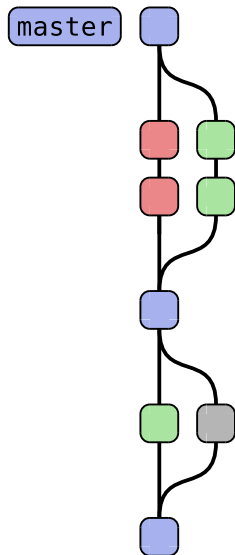


- Vorteile
 - Keine Merge-Commits
- Nachteile
 - »Sinnloses« Linearisieren
 - Feature-Commits in zufälliger Reihenfolge
 - Konflikte beim Rebase?

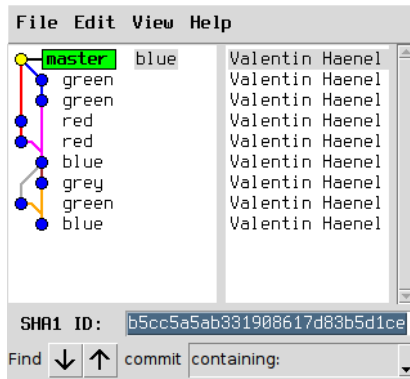
Interlude: Darstellung von Git-Graphen

- Bei mir:
 - Zeit läuft von unten nach oben
 - Wie bei Gitk, Eclipse, SourceTree etc...
- Aber:
 - Mehrere Commits auf einer Zeile
(vereinfacht die Darstellung)
 - Bei Gitk, Eclipse, SourceTree ist das nicht so..

Interlude: Darstellung von Git-Graphen



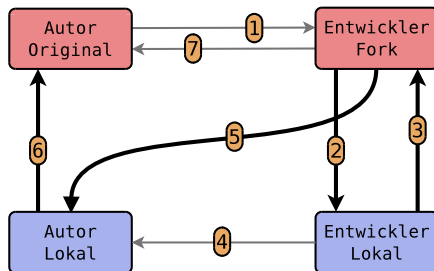
VS.



Übersicht

- 1 Intro
- 2 Push 'n' Pull
- 3 Github-Flow**
- 4 Rebase 'n' Force-Merge
- 5 Gitflow
- 6 Andere
 - Gitworkflows
 - Gerrit Workflow
- 7 Outro

- Geeignet für Github
- Webinterface oder 'github' Kommandozeilenerweiterung
- Nutzt Feature-Banches

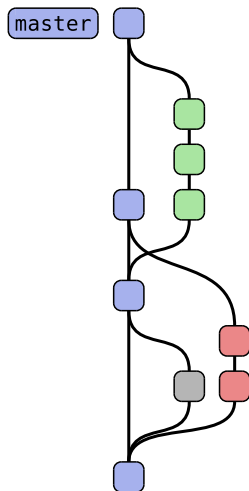


Entwickler:

- 1 Forkt ein Repository
- 2 Klont seinen Fork und macht Commits in einem Feature-Branch
- 3 Pusht den Feature-Branch in seinen Fork
- 4 Eröffnet einen Pull-Request

Autor:

- 5 Fetcht den Feature-Branch und mergt ihn
- 6 Pusht ins Original
- 7 **oder:** Mergt den Pull-Request über das Webinterface



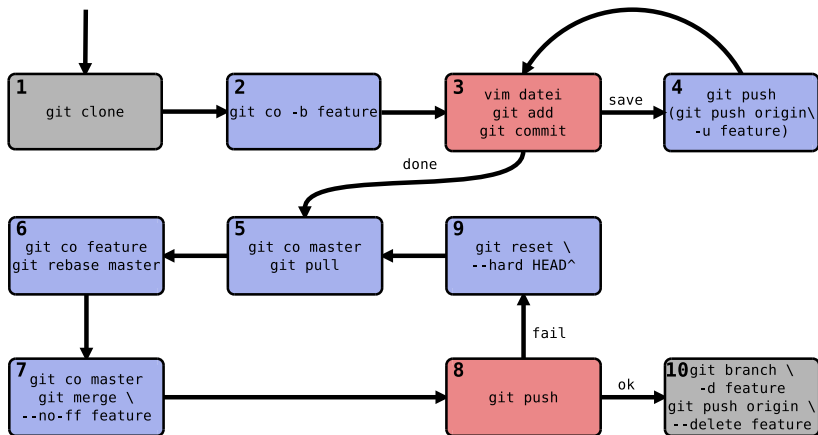
- Vorteile
 - Feature-Branches
- Nachteile
 - Feature-Branches basieren vor dem Merge evtl. nicht auf dem aktuellen Master
 - »Durcheinander«

- Beim Merge von Pull-Requests wird der Merge-Commit automatisch forciert
- Weitere Schritte vom Entwickler nötig
 - → Feature-Branches löschen
 - → Synchronisation vom `master` im Fork mit `master` aus Original

Übersicht

- 1 Intro
- 2 Push 'n' Pull
- 3 Github-Flow
- 4 Rebase 'n' Force-Merge
- 5 Gitflow
- 6 Andere
 - Gitworkflows
 - Gerrit Workflow
- 7 Outro

Rebase 'n' Force-Merge Workflow



Beschreibung der Schritte

❶ Repository clonen:

```
git clone <url>
```

❷ Feature-Branch anlegen

```
git checkout -b feature
```

❸ Arbeiten:

```
vim datei
```

```
git add datei
```

```
git commit -m "<msg>"
```

❹ Getane Arbeit hochladen

```
# beim ersten Mal
```

```
git push origin -u feature
```

```
git push
```

❺ Lokalen master aktualisieren:

```
git checkout master
```

```
git pull
```

❻ Rebase feature auf master:

```
git checkout feature
```

```
git rebase master
```

❼ Forcierter Merge von feature:

```
git checkout master
```

```
git merge --no-ff feature
```

❽ master Hochladen:

```
git push
```

❾ Fehlschlag: Merge rückgängig:

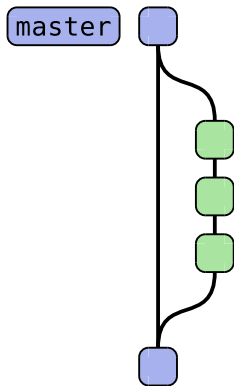
```
git reset --hard HEAD^
```

❿ Erfolg: feature löschen, lokal und remote:

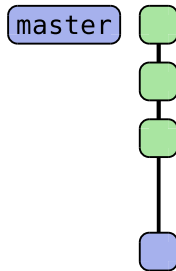
```
git branch -d feature
```

```
git push origin --delete  
feature
```

Rebase 'n' Force-Merge – Force-Merge



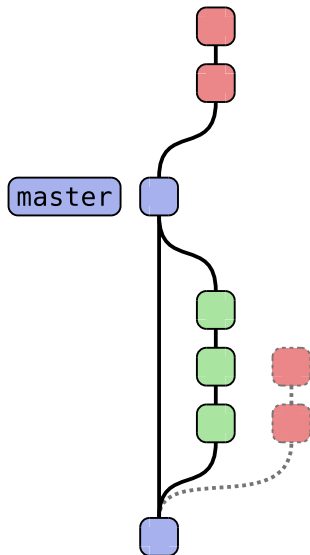
`git merge --no-ff`



`git merge`

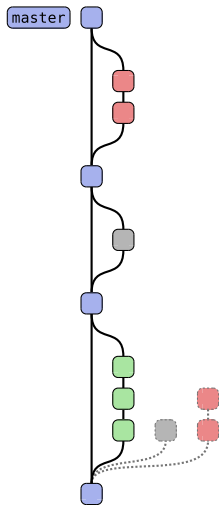
- Fast-Forward vermeiden
- Merge-Commit erzwingen
 - Nur so kann git revert einen Merge rückgängig machen
- Merge-Commit als »sinvolle« Markierung

Rebase 'n' Force-Merge – Rebase



- »Auf eine neue Basis stellen«
- Integration von Änderungen aus master
- Vorbereitung für den Merge

Rebase 'n' Force-Merge – Resultat



- Vorteile

- Saubere history
- Feature-Branches
- Merges sinnvoll

- Nachteile

- Verständnis von Git gebraucht
- Mehrere Kommandos nötig

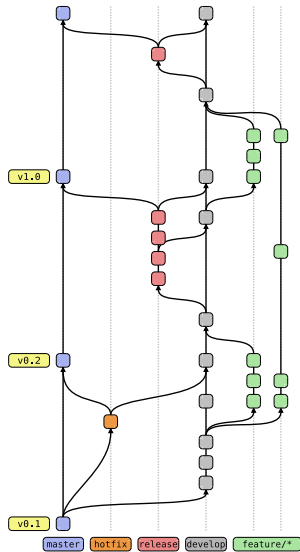
- Kann auch im Github-Flow verwendet werden
 - → Feature-Banches vor dem Merge des Pull-Request per Force-Push aktualisieren
 - → Beim Merge von Pull-Requests wird der Merge-Commit automatisch forciert

Übersicht

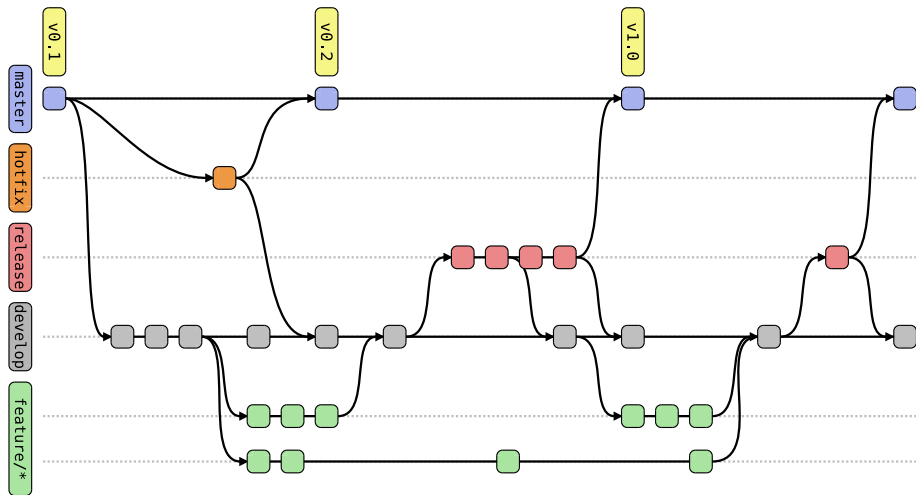
- 1 Intro
- 2 Push 'n' Pull
- 3 Github-Flow
- 4 Rebase 'n' Force-Merge
- 5 Gitflow**
- 6 Andere
 - Gitworkflows
 - Gerrit Workflow
- 7 Outro

- Großflächiges Branch-Modell
- Branches
 - `master` → Stabil/Produktion
 - `develop` → Instabil/Vorbereitung
 - `feature` → Feature-Entwicklung
 - `release` → Letzte Releasearbeiten
 - `hotfix` → Hotfixes
- <http://nvie.com/posts/a-successful-git-branching-model/>

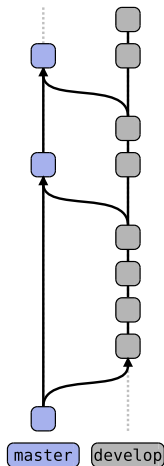
Gitflow



Gitflow – pdftk gitflow.pdf cat 1E output gitflow-rotated.pdf

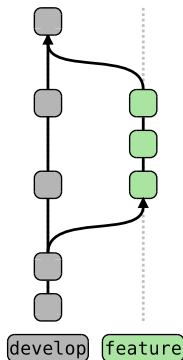


Gitflow – master und develop



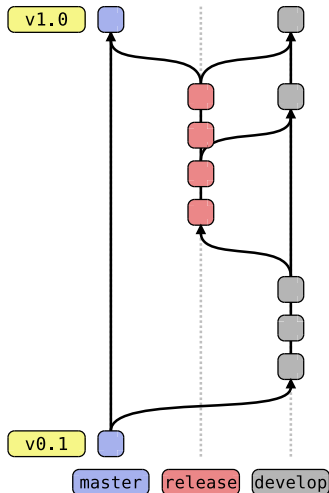
- Zwei langlebige Branches
 - master → Stabil/Produktion
 - develop → Instabil/Vorbereitung

Gitflow – Feature-Branches



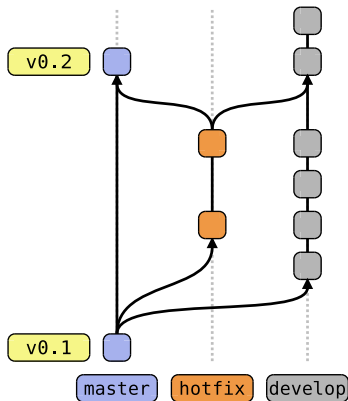
- Neue Features werden hier entwickelt
- Zweigen ab von develop
- Werden gemert nach develop
- Wenn fertig → löschen nicht vergessen!

Gitflow – release



- Release-Vorbereitung
- Letzte Typo-Fixes
- Changelog, Version-Bump etc. . .
- Wichtige Änderungen → develop
- Wenn fertig → master und develop
- Tags werden auf master gesetzt

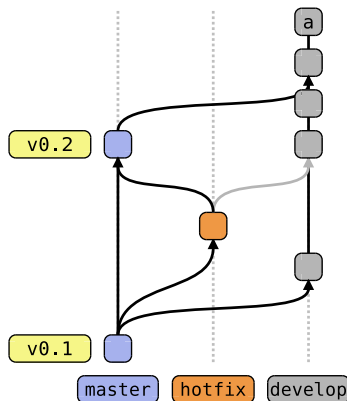
Gitflow – hotfix



- Security-kritische Fixes → Produktion
- Mergt auch → develop

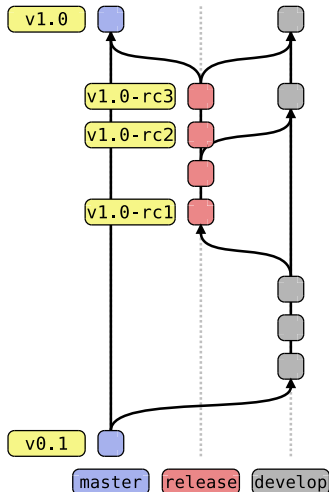
- Alle Merges sollten mit `--no-ff` passieren
- Triviale Commits auf `develop`?
- Rebase 'n' Force-Merge auch hier möglich
- Namensräume für Feature-Branches

Gitflow – Erweiterung Nr. 1: hotfix über master



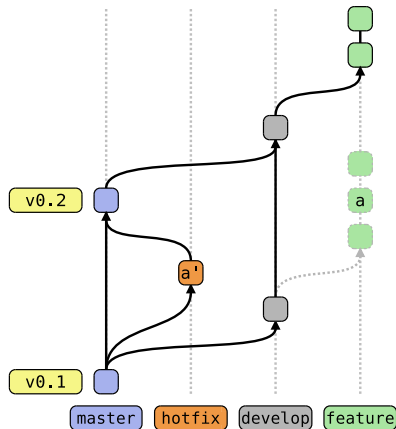
- Unterschiedliche »Erreichbarkeit«
- `git describe`
- Alt/Grau → `v0.1-5-g<sha1>`
- Neu/Schwarz → `v0.2-2-g<sha1>`
- Gilt auch für release

Gitflow – Erweiterung Nr. 2: Release-Candidate-Tags auf release



- Release-Candidate-Tags auf release
- Suffix: -rc*N*

Gitflow – Erweiterung Nr. 3: Rebase gegen hotfix



- Hotfixes per rebase in Feature-Branches integrieren
- Hier mit der alternativen Merge-Route
- (Evtl. wurde der Hotfix per Cherry-Pick übertragen)

Übersicht

- 1 Intro
- 2 Push 'n' Pull
- 3 Github-Flow
- 4 Rebase 'n' Force-Merge
- 5 Gitflow
- 6 Andere**
 - Gitworkflows
 - Gerrit Workflow
- 7 Outro

Manpage: Gitworkflows

```
man gitworkflows
```

- Älteres Model
- Info über `git.git`
- Sammlung von Best-Practices
- Vorschläge und Konzepte, e.g. »Throw-Away-Integration«

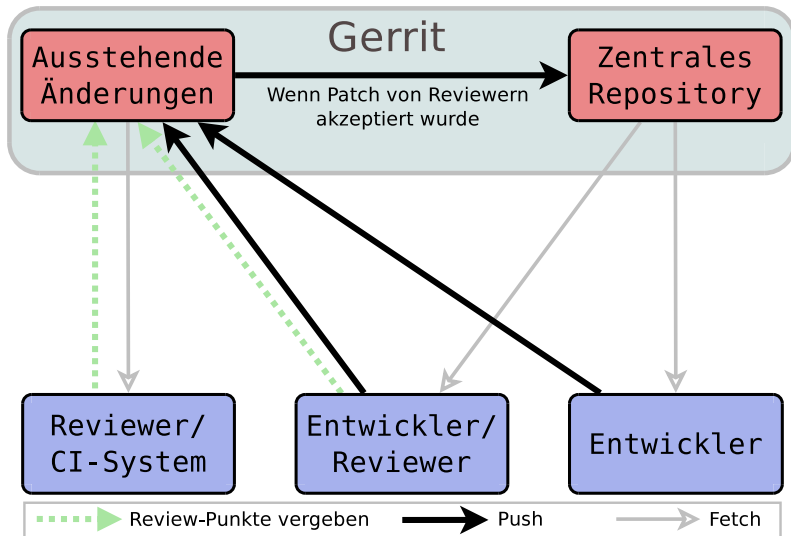
- Keine Bilder :(

Übersicht

- 1 Intro
- 2 Push 'n' Pull
- 3 Github-Flow
- 4 Rebase 'n' Force-Merge
- 5 Gitflow
- 6 Andere**
 - Gitworkflows
 - **Gerrit Workflow**
- 7 Outro

- Hosting, Webinterface für Code-Review
- Konzept: *Pending Changes*
- Eigener Workflow mit Feature-Banches und Automerging
- Genutzt von Wikimedia, OpenStack und Android

Gerrit Workflow



Übersicht

- 1 Intro
- 2 Push 'n' Pull
- 3 Github-Flow
- 4 Rebase 'n' Force-Merge
- 5 Gitflow
- 6 Andere
 - Gitworkflows
 - Gerrit Workflow
- 7 **Outro**

- In der Praxis macht sowieso jeder was er will
 - → nicht ärgern ;-)
- Nutzt Feature-Branches!!!
- Projektgröße?
 - Für diese Folien reichte ein einzelner master (KISS)
- Eigene Arbeitsweise Schritt für Schritt erweitern

Fragen?

- Quellen:
 - <https://github.com/esc/clt-2015-git-workflows>
- Danke:
 - Julius Plenz für Anregungen, Ideen und Patches
 - Jens Kubieziel für das erste Patch
- Alles mit Open-Source gemacht!
 - Wiki2beamer
 - L^AT_EXbeamer
 - Dia
 - ccBeamer
 - make, vim, zsh, git usw...
- Git-Schulungen in München (z.B. 04.–05.05.2015)
- Unser Git-Buch: <http://gitbu.ch>