Git-Workflows im Alltag

Valentin Haenel

Freelance Consultant and Software Developer
http://haenel.co
@esc___

2015-03-21 @ CLT

Version: v0.2.0 https://github.com/esc/clt-2015-git-workflows

Intro

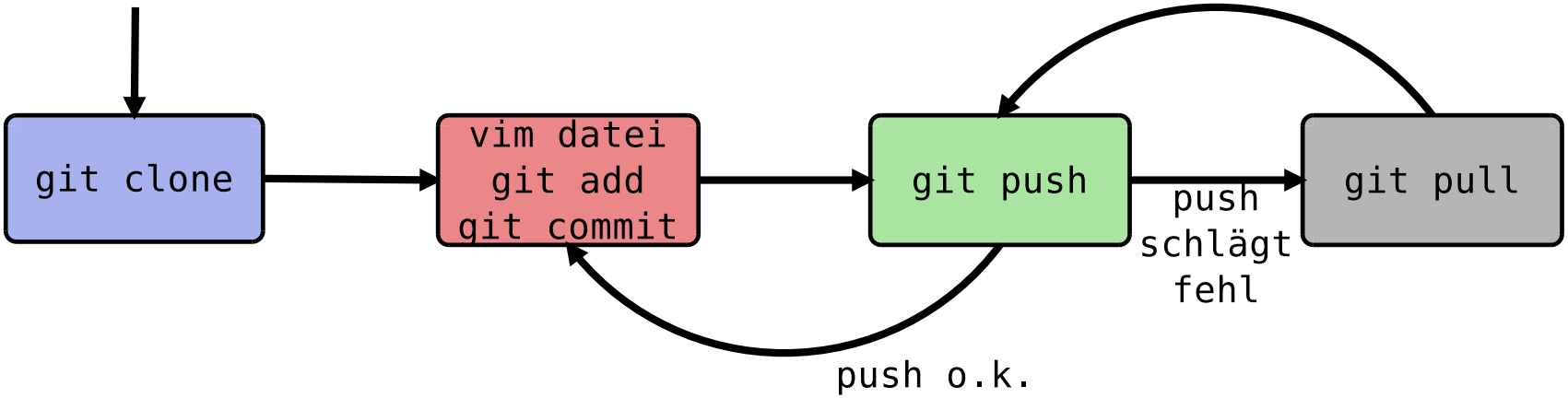
- Git-Workflows?

- Folge von Kommandos
- Nutzung von Branches und Tags

Übersicht

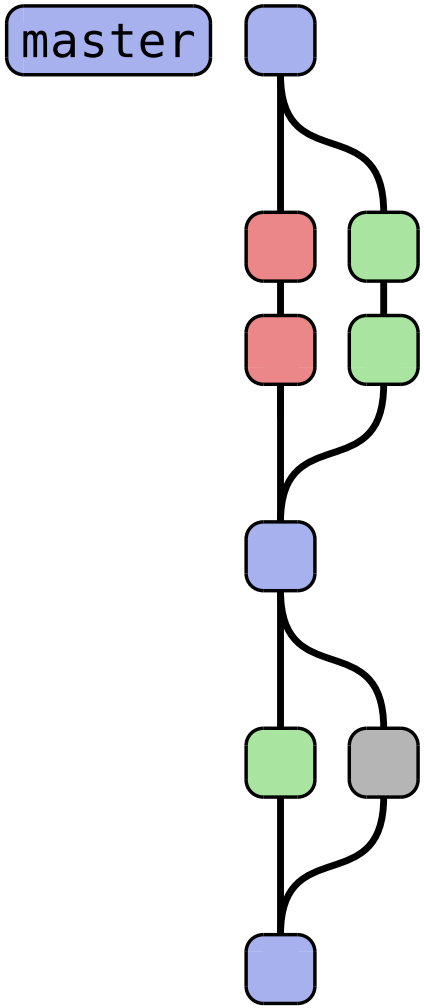
Übersicht

Push 'n' Pull Workflow



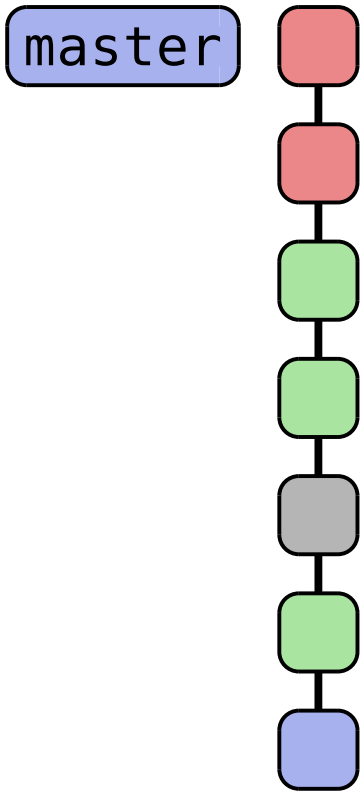
1. Lokale Änderungen
   - `vim` *datei*
   - `git add` *datei*
   - `git commit -m "`*msg*`"`
2. Änderungen veröffentlichen
   - `git push`
   - Wenn `push` fehlschlägt:
   - `git pull`, dann `git push`

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`

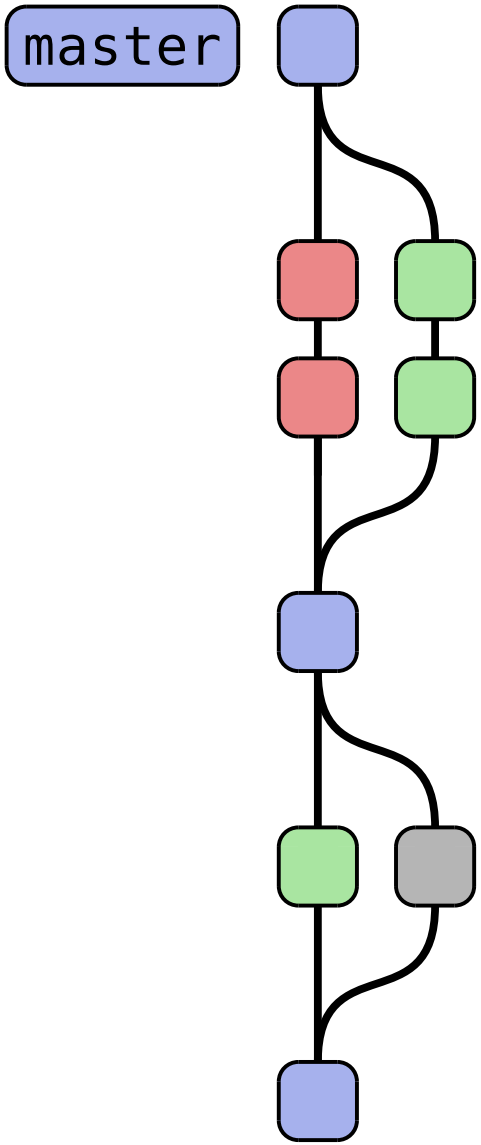


- Vorteile
  - Keine Merge-Commits
- Nachteile
  - »Sinnloses« Linearisieren
  - Feature-Commits in zufälliger Reihenfolge
  - Konflikte beim Rebase?

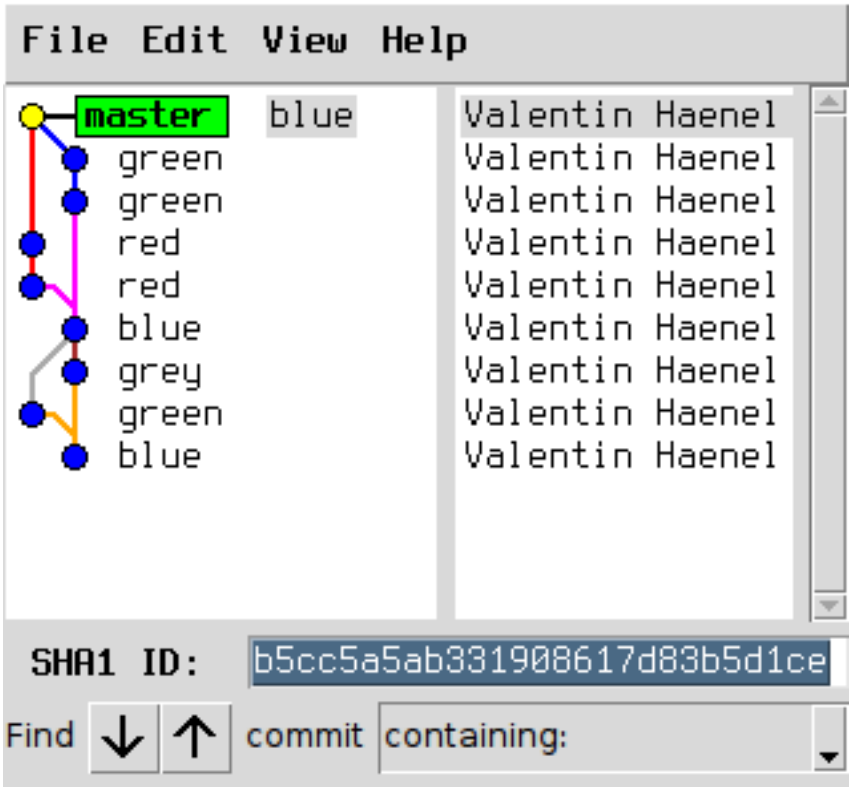
Interlude: Darstellung von Git-Graphen

- 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.

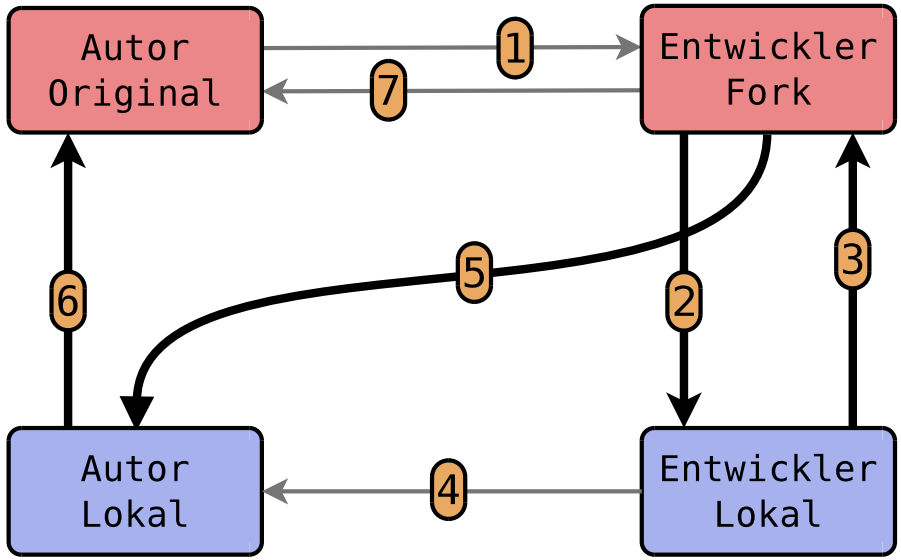


Übersicht

Github-Flow

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

Github-Flow



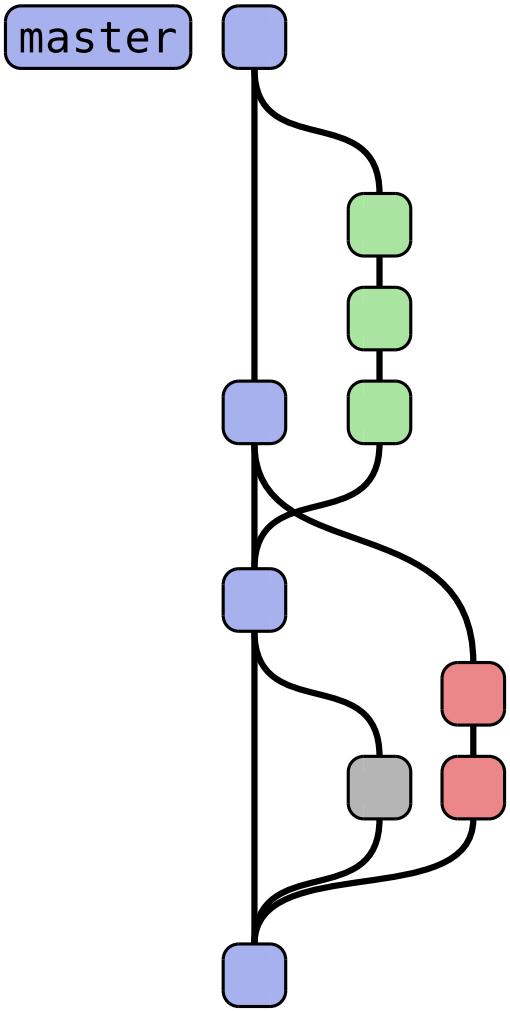
Entwickler:

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

Github-Flow – Resultat



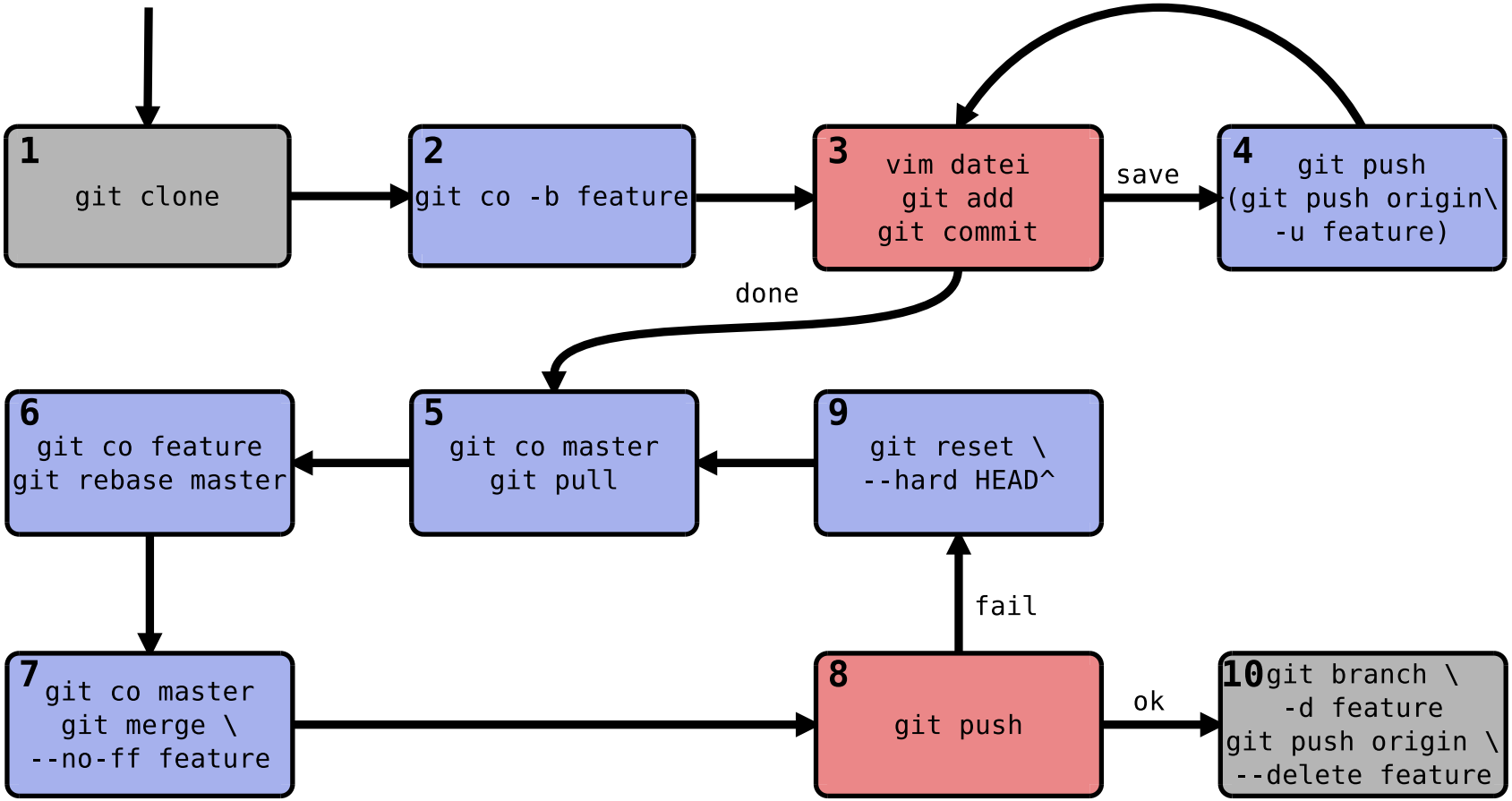
- Vorteile
  - Feature-Branches
- Nachteile
  - Feature-Branches basieren vor dem Merge evtl. nicht auf dem aktuellen Master
  - »Durcheinander«

Github-Flow – Anmerkungen

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

Übersicht

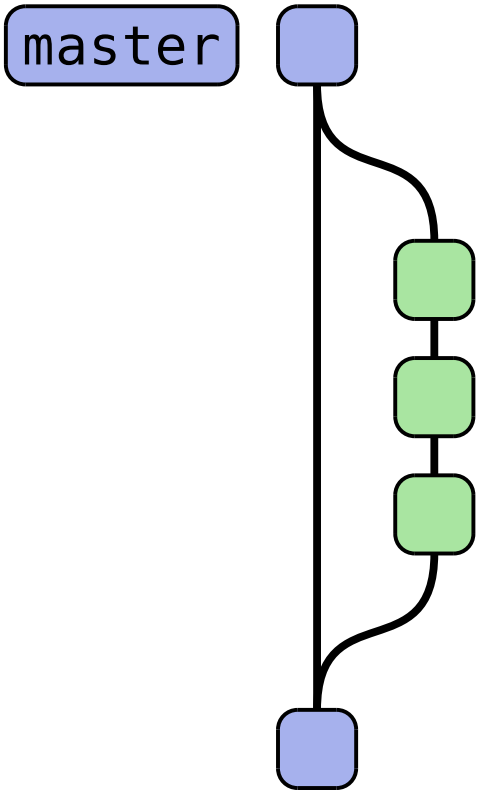
Rebase 'n' Force-Merge Workflow



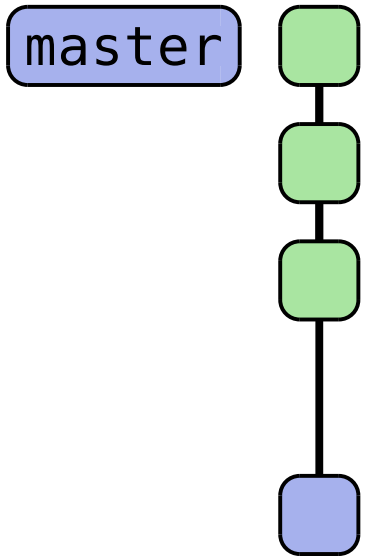
Beschreibung der Schritte

1. **Repository clonen:**
   ```
   git clone <url>
   ```
2. **Feature-Branch anlegen**
   ```
   git checkout -b feature
   ```
3. **Arbeiten:**
   ```
   vim datei
   git add datei
   git commit -m "<msg>"
   ```
4. **Getane Arbeit hochladen**
   ```
   # beim ersten Mal
   git push origin -u feature
   git push
   ```
5. **Lokalen `master` aktualisieren:**
   ```
   git checkout master
   git pull
   ```
6. **Rebase `feature` auf `master`:**
   ```
   git checkout feature
   git rebase master
   ```
7. **Forcierter Merge von `feature`:**
   ```
   git checkout master
   git merge --no-ff feature
   ```
8. **`master` Hochladen:**
   ```
   git push
   ```
9. **Fehlschlag: Merge rückgängig:**
   ```
   git reset --hard HEAD^
   ```
10. **Erfolg: `feature` löschen, lokal und remote:**
    ```
    git branch -d feature
    git push origin --delete
    feature
    ```

Rebase 'n' Force-Merge – Force-Merge



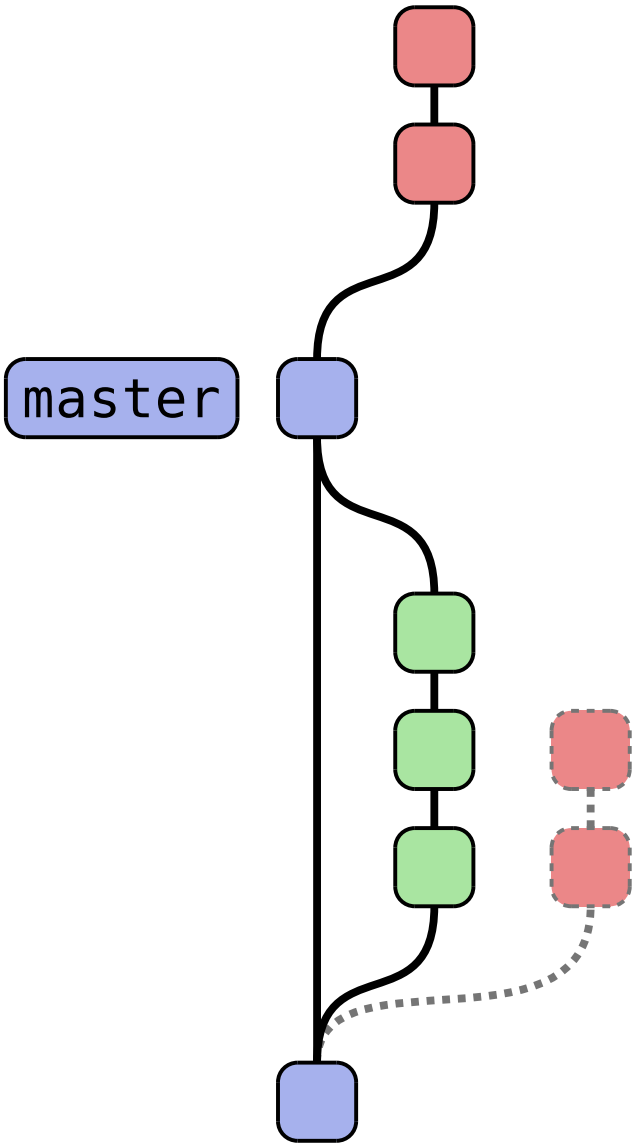
`git merge --no-ff`



`git merge`

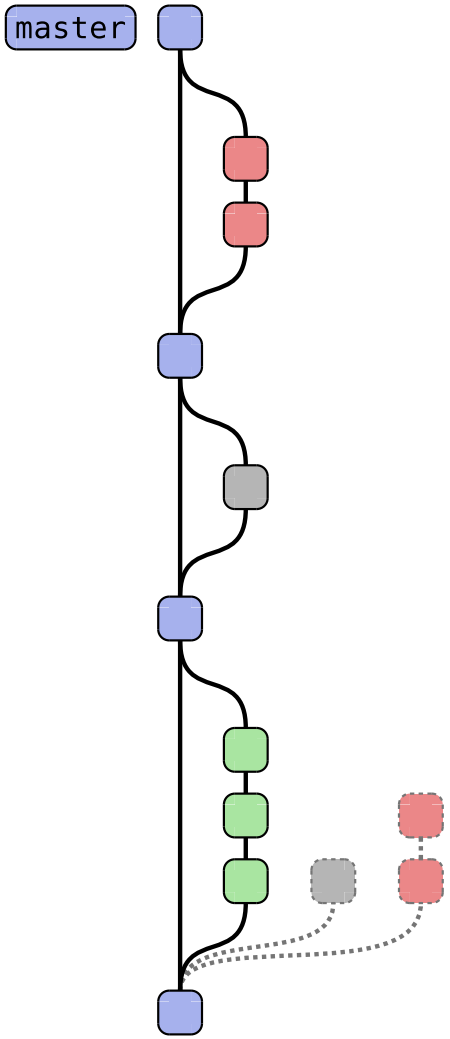
- Fast-Forward vermeiden
- Merge-Commit erzwingen
  - Nur so kann `git revert` einen Merge rückgängig machen
- Merge-Commit als »sinvolle« Markierung

Rebase 'n' Force-Merge – Rebase



- »Auf eine neue Basis stellen«
- Integration von Änderungen aus `master`

Rebase 'n' Force-Merge – Resultat



- Vorteile
  - Saubere history
  - Feature-Branches
  - Merges sinnvoll
- Nachteile
  - Verständnis von Git gebraucht
  - Mehrere Kommandos nötig

Rebase 'n' Force-Merge – Anmerkungen

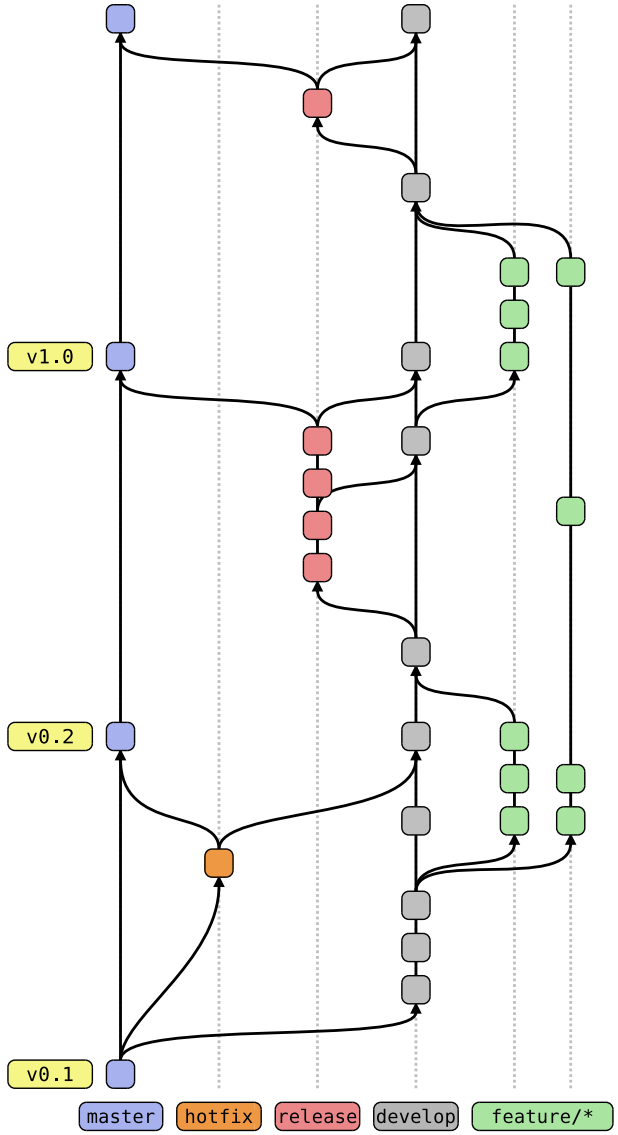
- Kann auch im Github-Flow verwendet werden
  - $\rightarrow$ Feature-Branches vor dem Merge des Pull-Request per Force-Push aktualisieren
  - $\rightarrow$ Beim Merge von Pull-Requests wird der Merge-Commit automatisch forciert

Übersicht

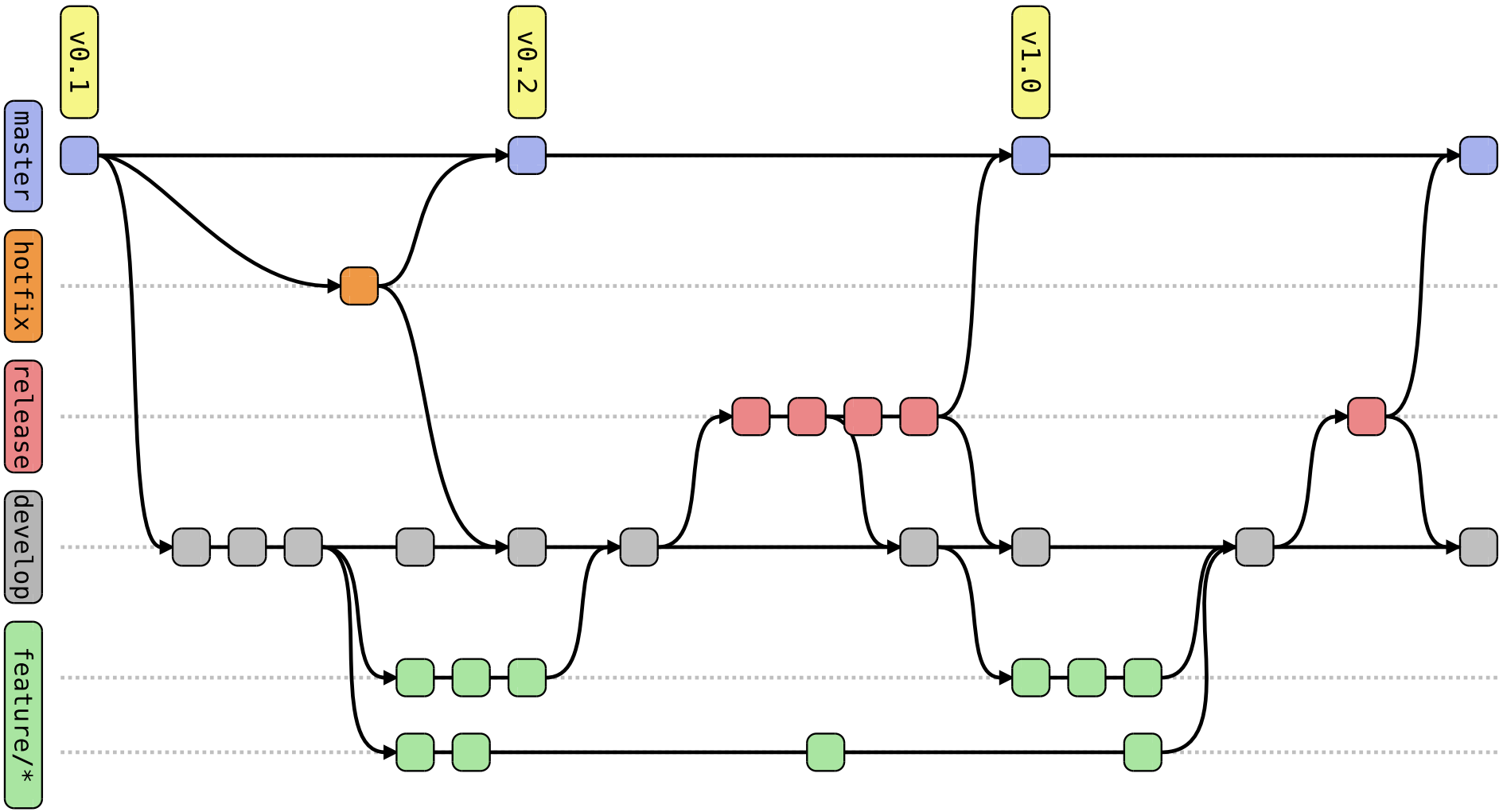
Gitflow

- 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/

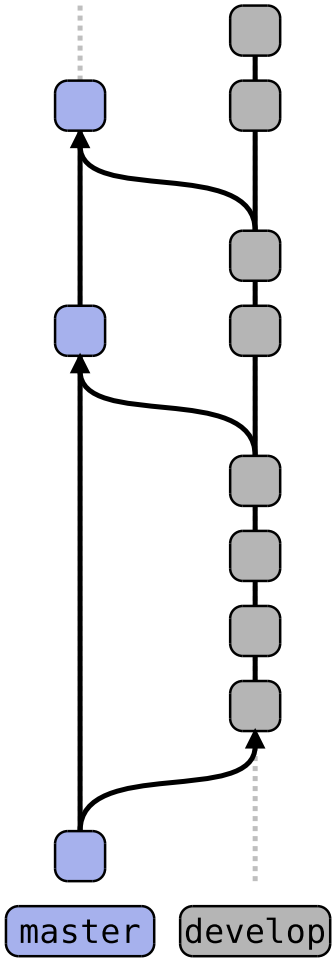
Gitflow



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

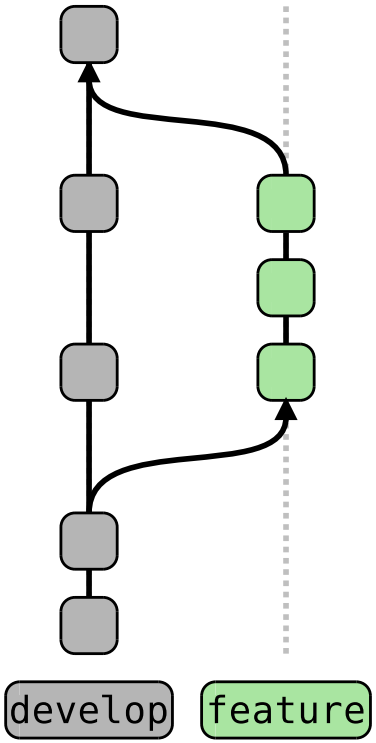


Gitflow – `master` und `develop`



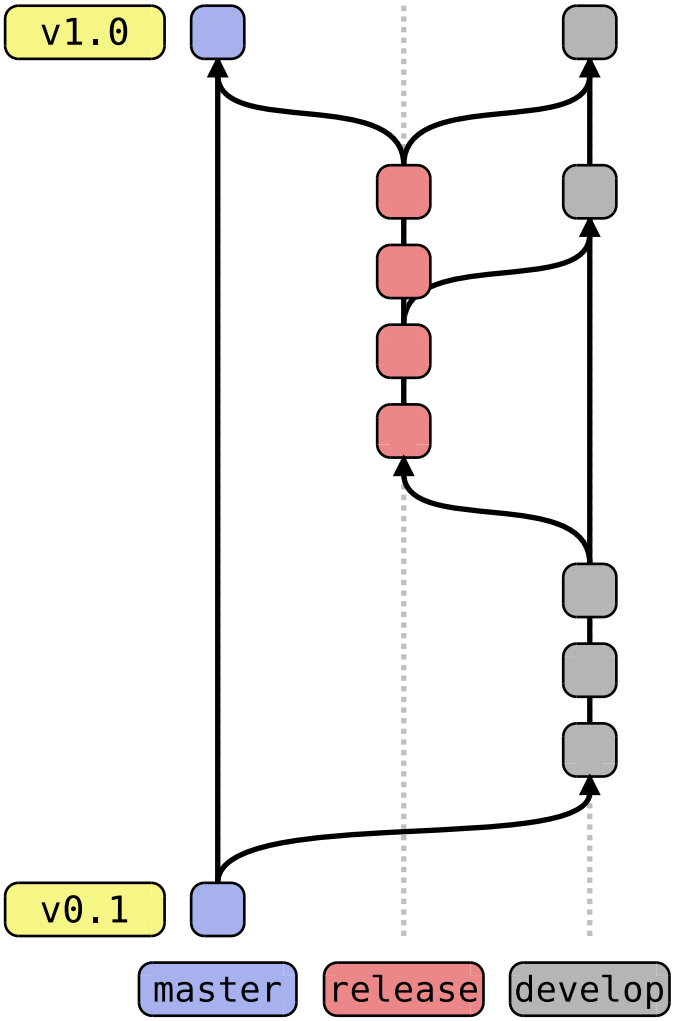
- Zwei langlebige Branches
  - `master` → Stabil/Produktion
  - `develop` → Instabil/Vorbereitung

Gitflow – Feature-Branches



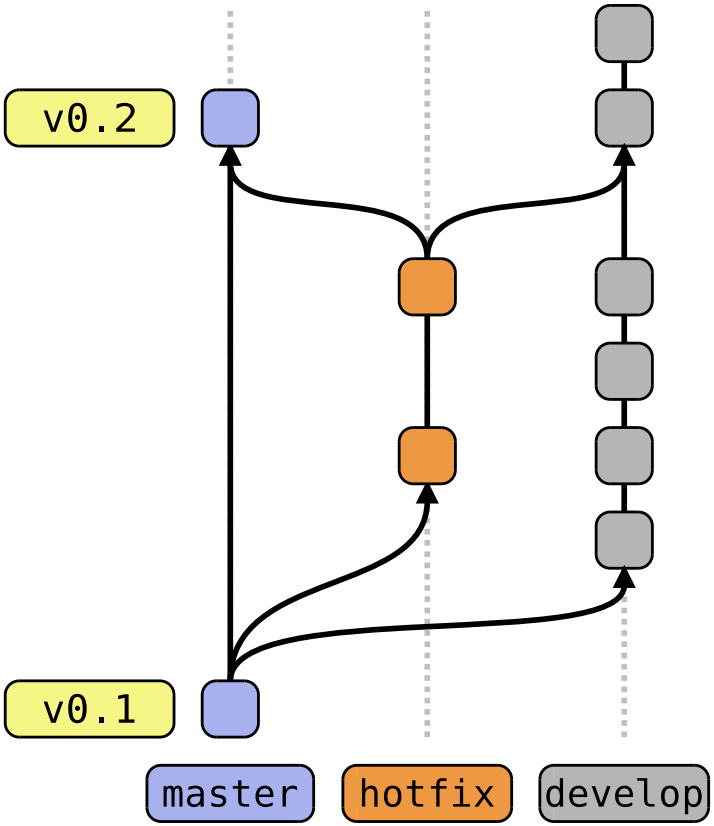
- Neue Features werden hier entwickelt
- Zweigen ab von `develop`
- Werded gemert nach `develop`
- Wenn fertig $\rightarrow$ löschen nicht vergessen!

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`

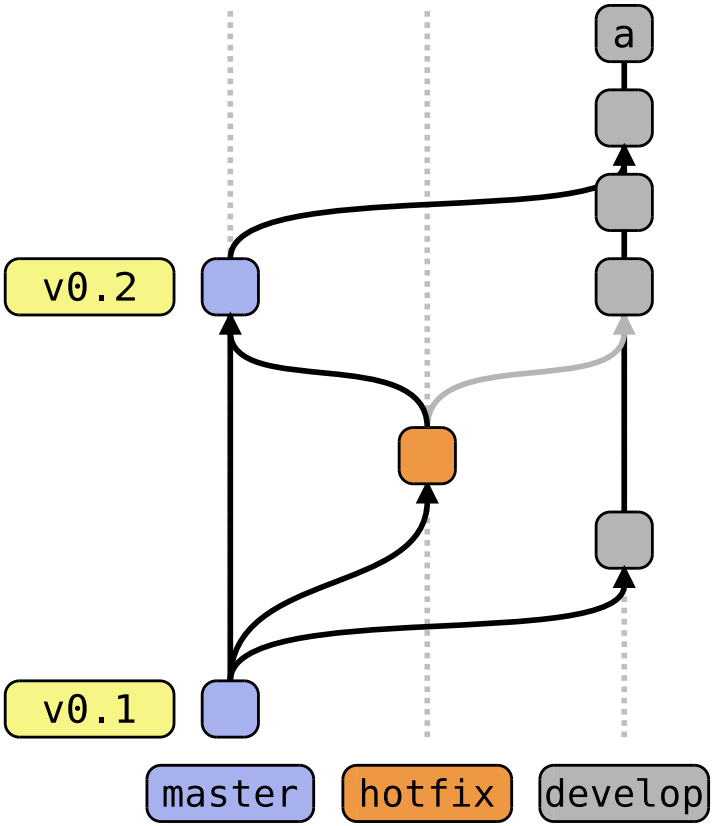


- Security-kritische Fixes → Produktion
- Mergt auch → `develop`

Gitflow – Anmerkungen

- 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`

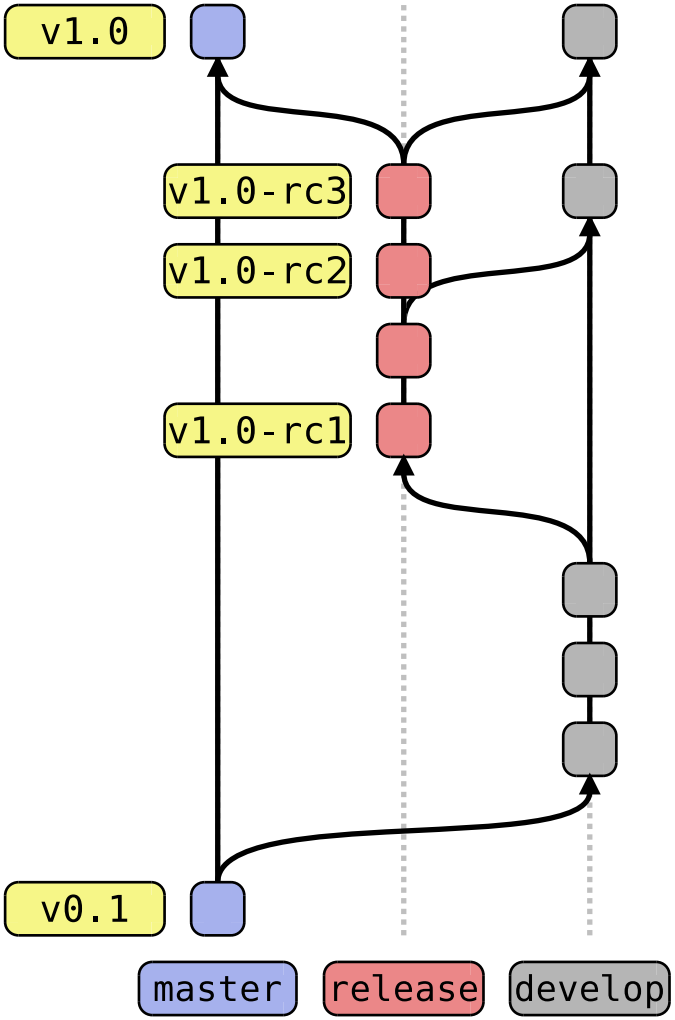


- Unterschiedliche »Erreichbarkeit«
- `git describe`

- Alt/Grau → `v0.1-5-g<sha1>`
- Neu/Schwarz → `v0.2-2-g<sha1>`

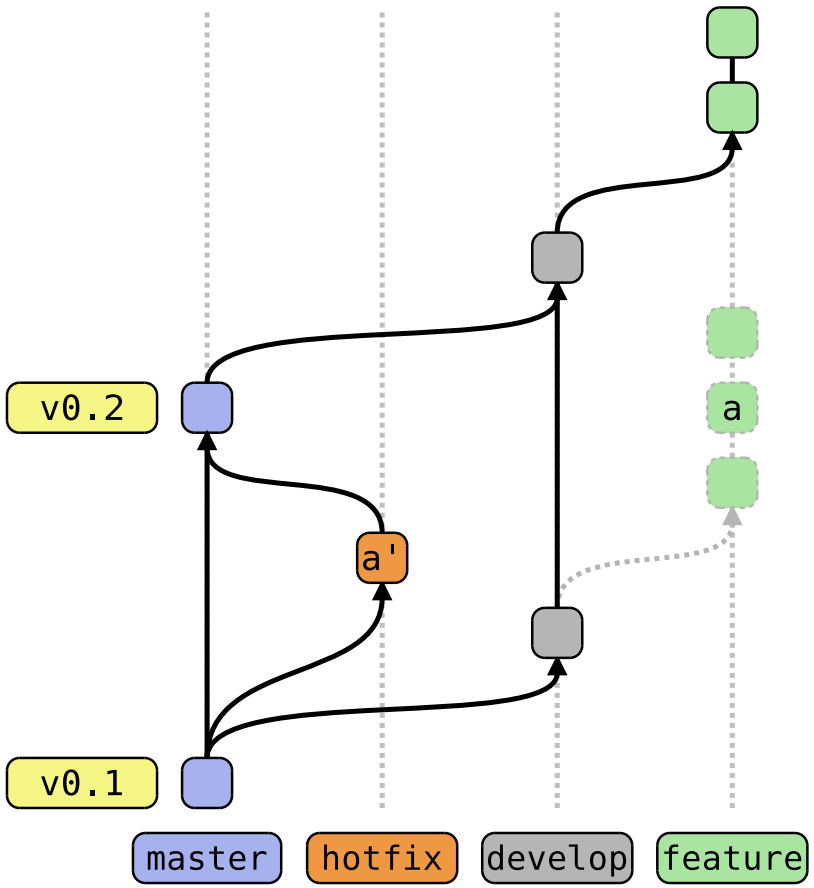
- Gilt auch für `release`

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

Manpage: Gitworkflows

```
man gitworkflows
```

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

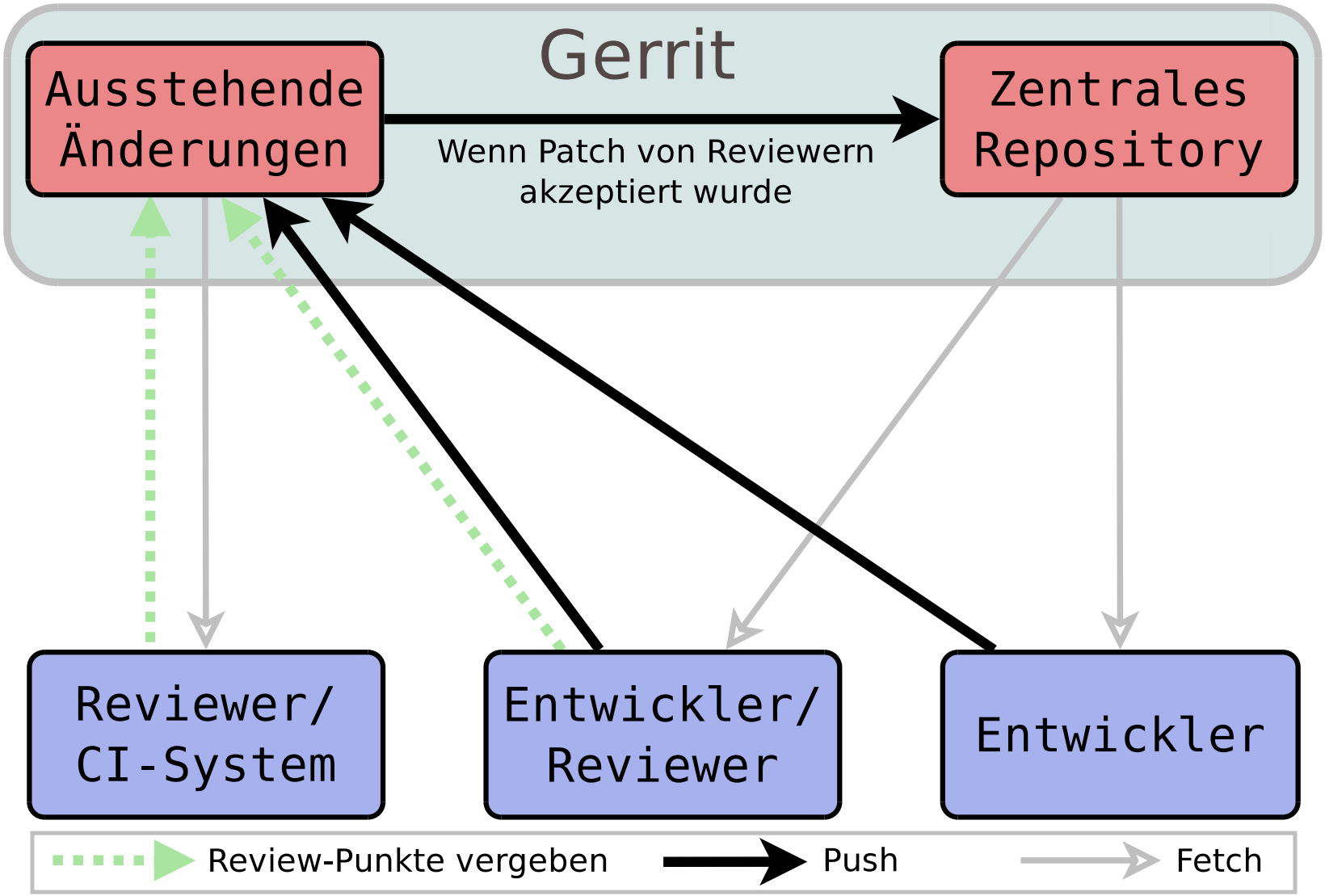
- Keine Bilder :(

Übersicht

Gerrit Workflow

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

Gerrit Workflow



Übersicht

Fazit

- In der Praxis macht sowieso jeder was er will
  - $\rightarrow$ 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
  - LaTeXbeamer
  - Dia
  - ccBeamer
  - `make`, `vim`, `zsh`, `git` usw...
- Git-Schulungen in München (z.B. 04.–05.05.2015)
- Unser Git-Buch: http://gitbu.ch