Git II

Florian Engel

June 17, 2019

Wer hat was verändert

```
> git log --author="Florian Engel"
```

Zeigt alle commits an deren Autor Florian Engel ist

Nach bestimmten commit messages suchen

```
> git log --grep "monetdb"
```

Zeigt alle commits an in deren message "monetdb" vorkommt

Was wurde geändert

```
> git log --patch
```

Zeigt zusätzlich zum log die diffs der einzelnen commits an

Den log einer Datei anzeigen

```
> git log <datei>
```

- Zeigt den log von <datei> an
- kannn mit –patch verbunden werden

Dateien vergleichen mit diff

```
> git diff <commit>
> git diff <commit> <commit>
```

Zeigt die Änderungen zwischen 2 commits an

Demo

Änderen von letztem commit mit amend

```
> git commit --amend
```

Bearbeitet den letztem commit

Beispiel

```
> echo "The Room: 5 stars" > movie_rating.txt
> git add movie_rating.txt
> git commit -m "Added the room to rating"
> git commit --amend -m "Add the room to rating"
> # edit rating to 1 star
> git add movie_rating.txt
> git commit --amend --no-edit
```

Ändern von mehrenen commits

> git rebase –interactive HEAD~<Anzahl commits>

Optionen:

- p (pick): der commit soll so bleiben
- r (reword): Die commit-message soll verändert werden
- e (edit): amend auf diesen commit
- s (squash): Der commit und der davor sollen zu einem zusammengeführt werden
- f (fixup): Wie s aber ohne die commitmessage
- d (drop): commit soll entfernt werden

Wie funktioniert "^"

```
> git log HEAD^
> git log HEAD^^
> git log HEAD^<n>
```

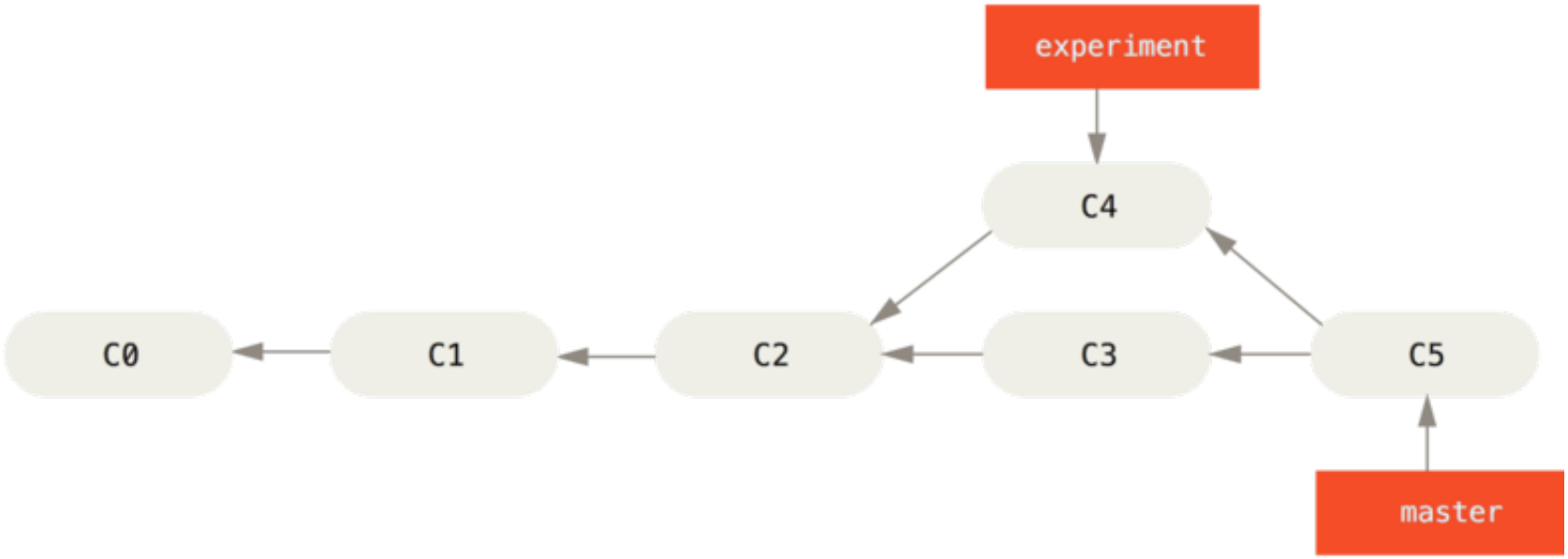
- Gibt den ersten, zweiten,... Parentcommit aus
- 'HEAD^n' gibt den Parentcommit vom Parentcommit aus

Wie funktioniert "~"

```
> git log HEAD~
> git log HEAD~~
> git log HEAD~<n>
```

- Gibt den ersten, zweiten,... Parentcommit aus
- 'HEAD~n' gibt den 2. Parentcommit aus

Was ist der unterschie zwischen "~" und "^"



- Welchen commit gibt `git show HEAD~2`
- Welchen commit gibt `git show HEAD^2`

Wie sehe ich alle commits von topic die noch nicht in master sind

> git log master..topic

Wie sehe ich alle commits die noch nicht in master sind

> git log topic ^master

xours

```
> git merge -Xours <branch>
```

Wenn ein Konflikt auftritt benutze den aktuellen branch

xtheirs

```
> git merge -Xtheirs <branch>
```

Wenn ein Konflikt auftritt benutze den anderen branch

ours

```
> git merge -s ours <branch>
```

Nimm nur den aktuellen Branch

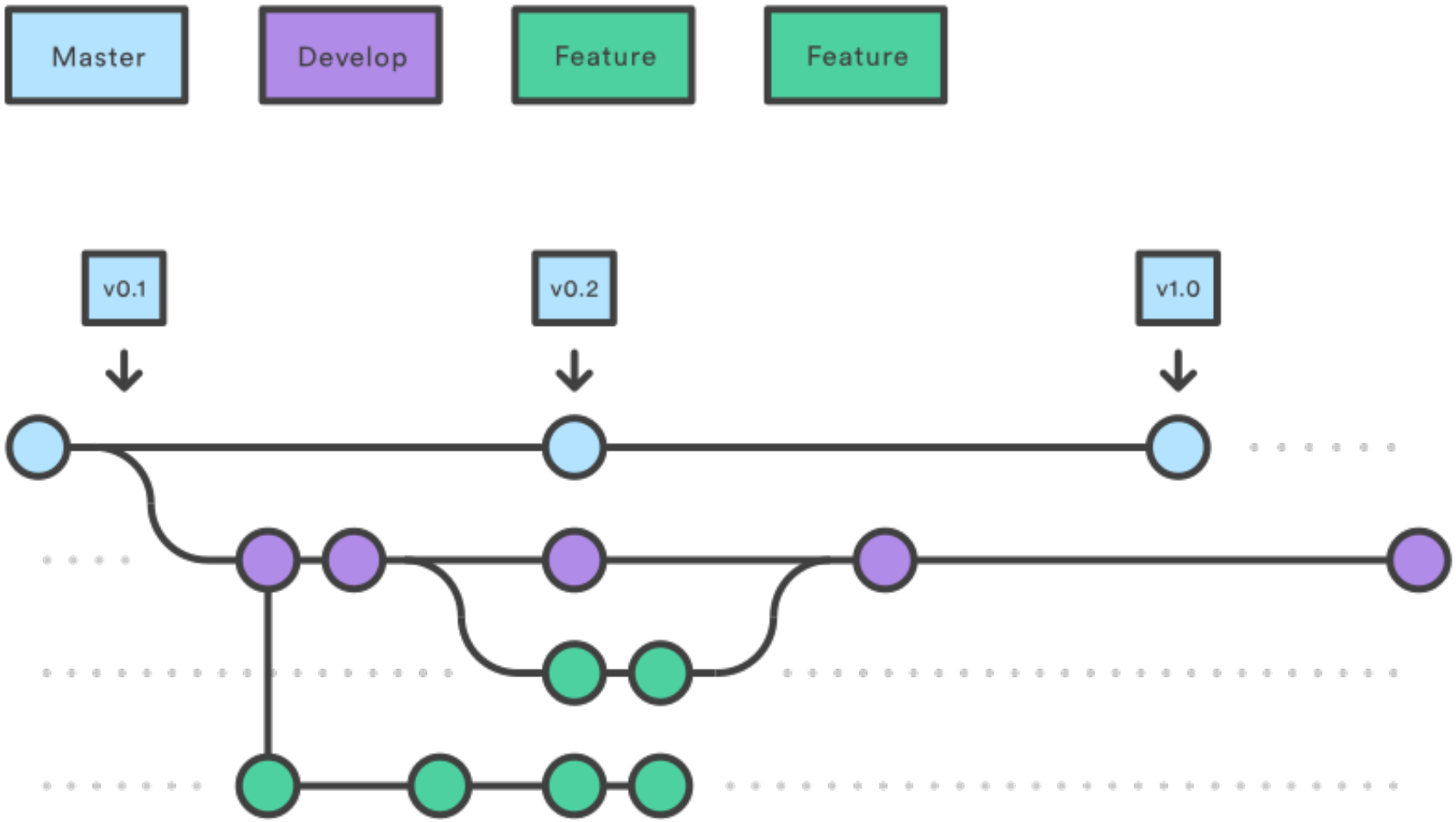
theirs

```
> git merge -s theirs <branch>
```

Merge Konflikt auflösen

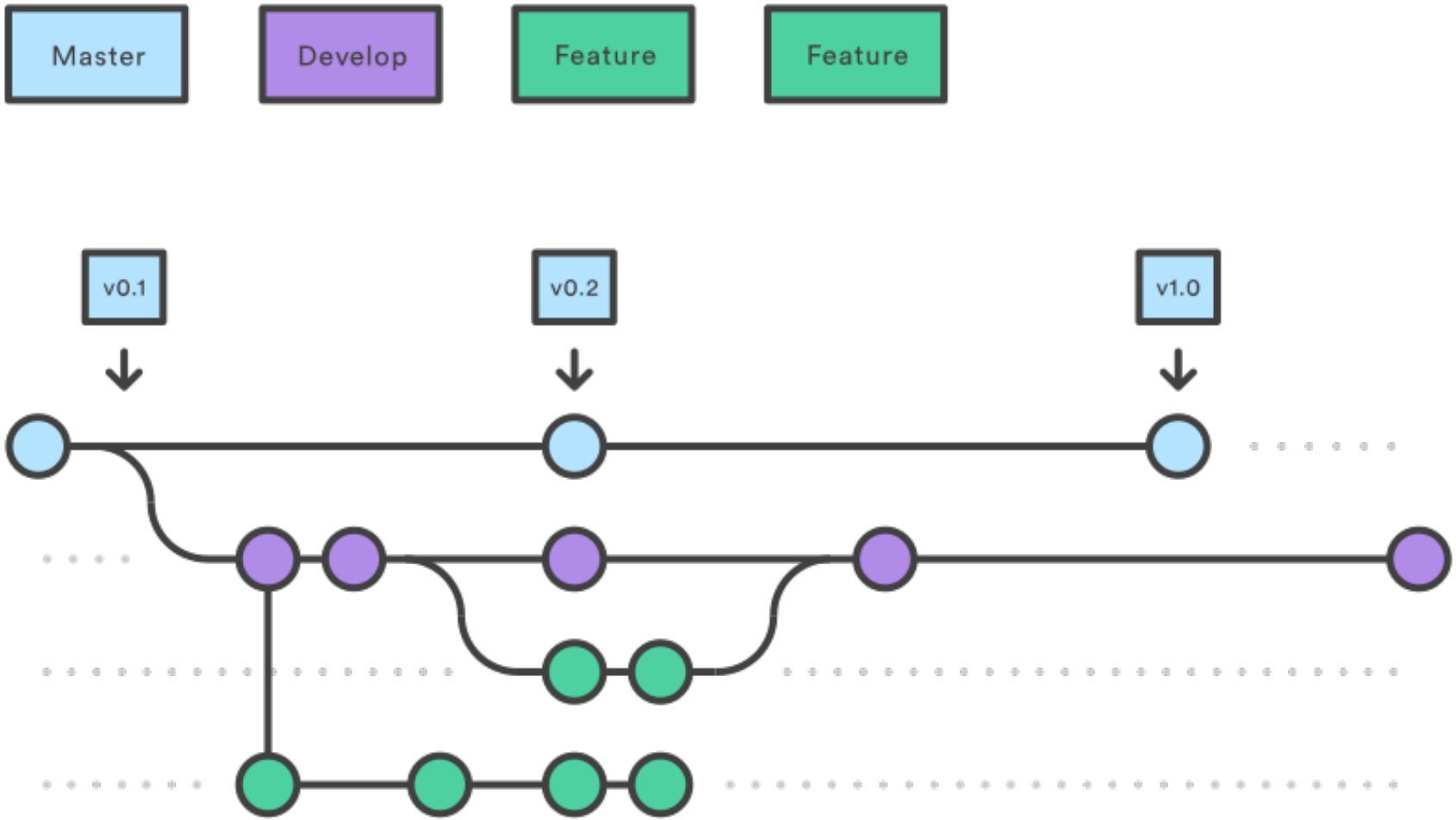
Demo

Workflow



- Bei projekten kommen bestimmte branches häufig vor
- master,stable,dev und feature branches

master

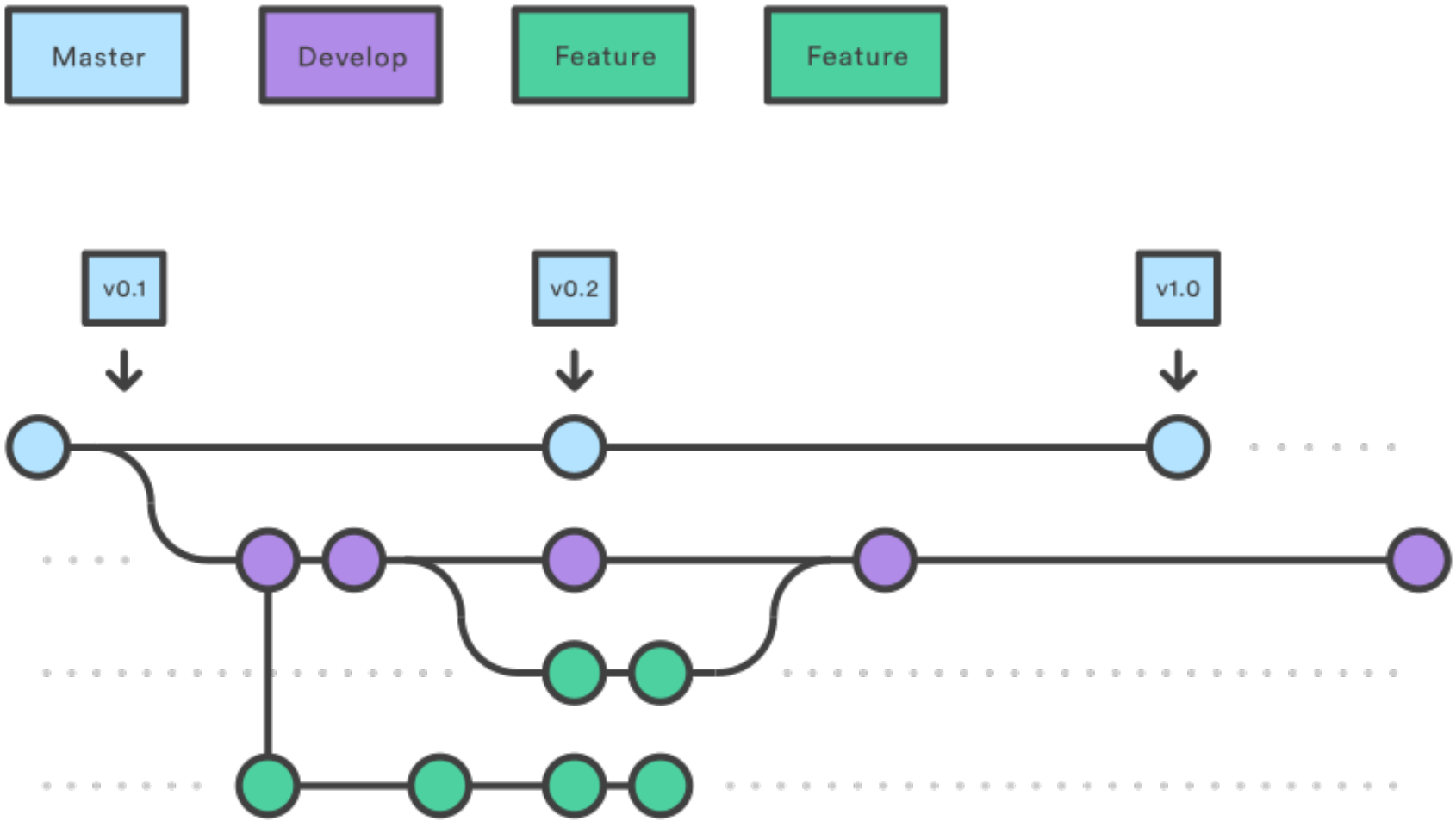


- Das aktuell laufende Projekt
- Enthält commits die an den Endnutzer weitergegeben werden können

stable

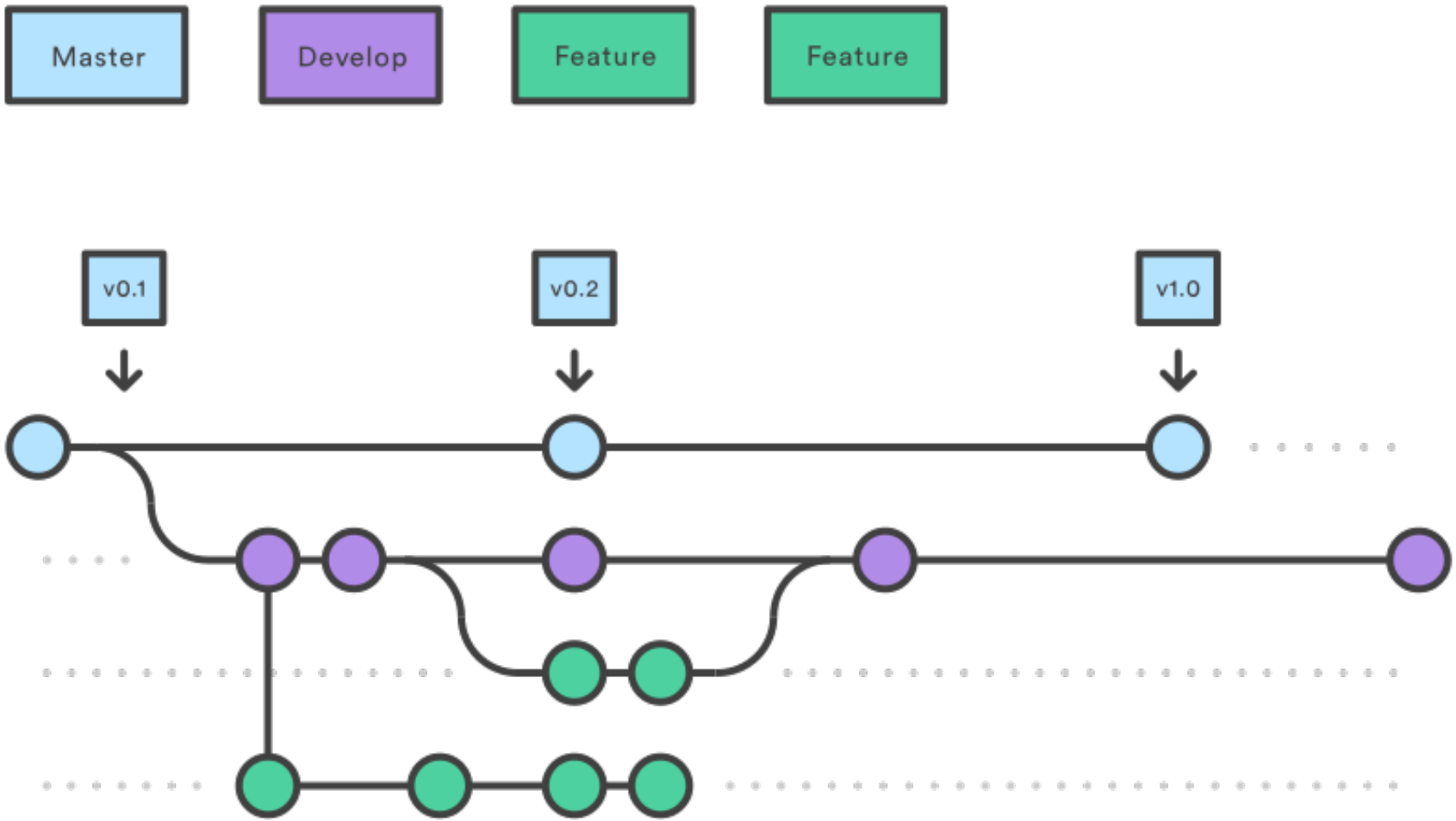
- Alle gut getesteten commits
- Das Projekt in diesem Branch sollte wenig Bugs haben

dev



- Der aktuelle stand des Projekts
- Entwickler wollen diesen noch nicht an Endnutzer weiter geben
- Enthält eventuell noch viele Bugs

feature branches



- Für jedes Feature einen branch
- Wenn das Feature fertig ist, wird es in dev gemerged