

Softwaretechnikpraktikum

Git - Einführung



SOFTWARE
SYSTEME



UNIVERSITÄT
LEIPZIG

Prof. Dr.-Ing. Norbert Siegmund
Sebastian Simon
Software Systems

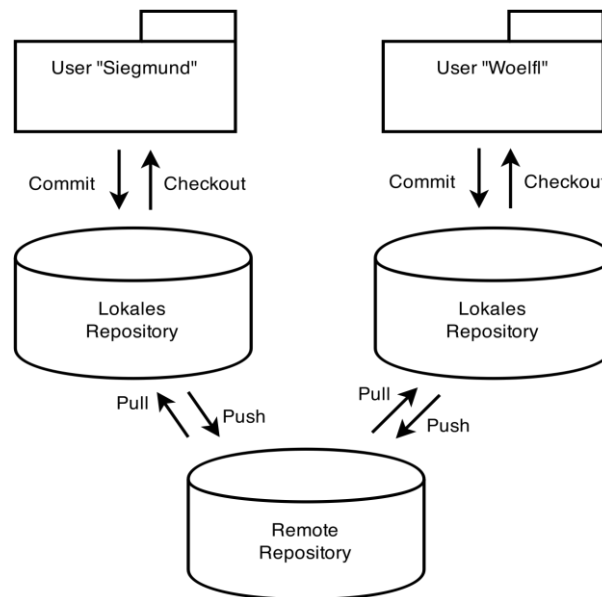
Versionskontrolle

- Versionskontrolle: Entwicklung eines Projektes abbilden und über die Zeit verfolgen
- VCS Werkzeug für die Versionkontrolle
- Funktionalität VCS:
 - alte Versionen wiederherstellbar
 - Metadaten: Wer, wann, warum?
 - Protokollieren von Änderungen
 - mehrerer Entwicklungszweige

Verteilte Versionskontrollsystem

➤ Git

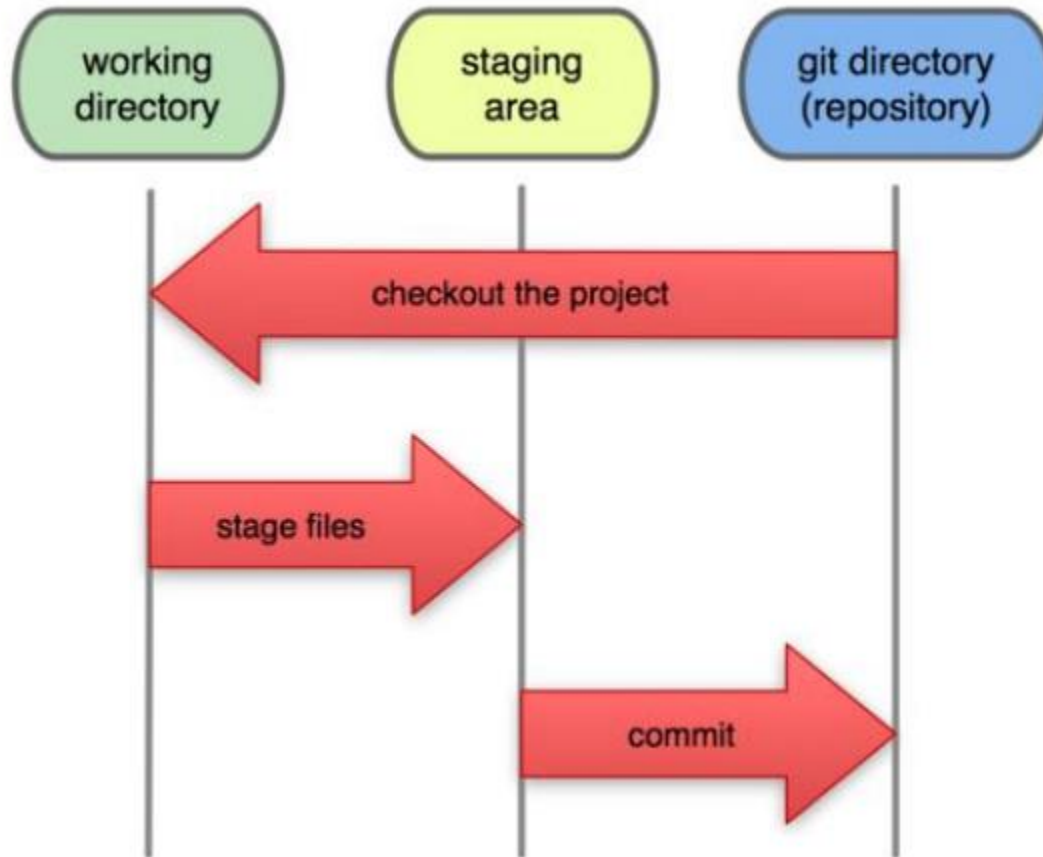
- verteilte Versionverwaltung
- jeder Nutzer hat eigenes lokales Repository
- braucht nicht zwingend Netzwerkanbindung
- Open Source + große Community



Terminologie

- **Revision:** Begriff für eine **Version** der Software
- **Repository:** **Archiv**, in dem Revisionen verwaltet werden
- **Checkout:** Übertragung der aktuellsten Revision vom Repository zum Benutzer als **neue Arbeitskopie**
- **Commit:** Übertragung der Arbeitskopie eines Benutzers zum Repository. Pro Commit wird eine **neue Revision** erzeugt
- **Branch:** Abspaltung von einer Version zu einem **parallelen Entwicklungszweig**
- **Merge:** Zusammenfügen von parallelen Entwicklungszweigen

Arbeitsbereiche

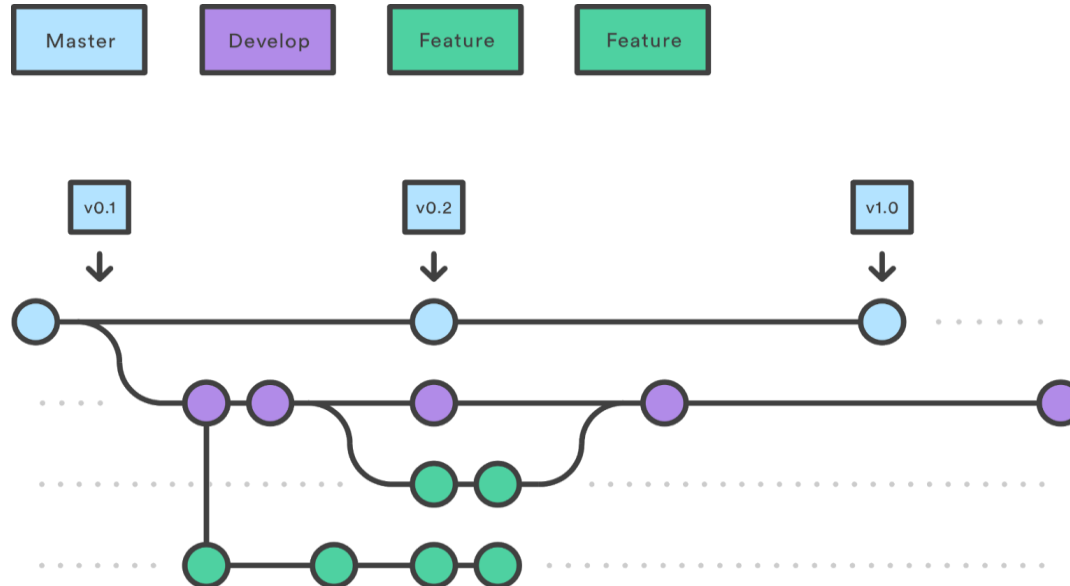


Arbeitsweise

- Wie benutze ich ein VCS richtig?
 - Es existieren mehrere Workflow-Modelle
 - Zentralisierter Workflow
 - Feature Branch Workflow
 - Gitflow Workflow
 - Forking Workflow
 - ...
- Gitflow: guter Mittelweg zwischen einfacher Benutzbarkeit und ausreichend Robustheit für große Projekte

Gitflow

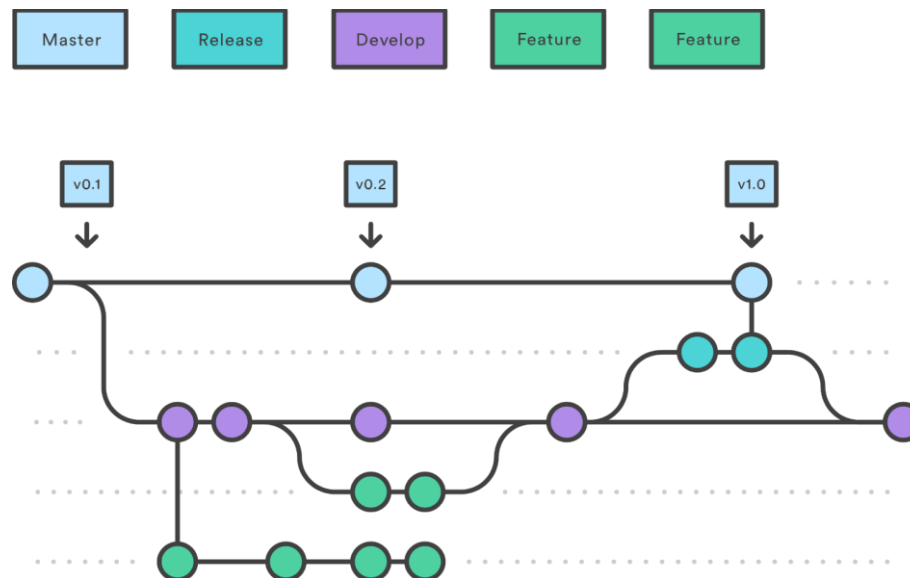
- Branching-Modell mit strikten Rollen für jeden Branch
- **Master:** Offizielle Release Historie
- **Develop:** Integrationzweig für neue Features
- **Feature:** jedes Feature soll in einem eigenen Branch entwickelt werden, Abspaltung und Zusammenführung immer mit Develop



Gitflow

➤ Release:

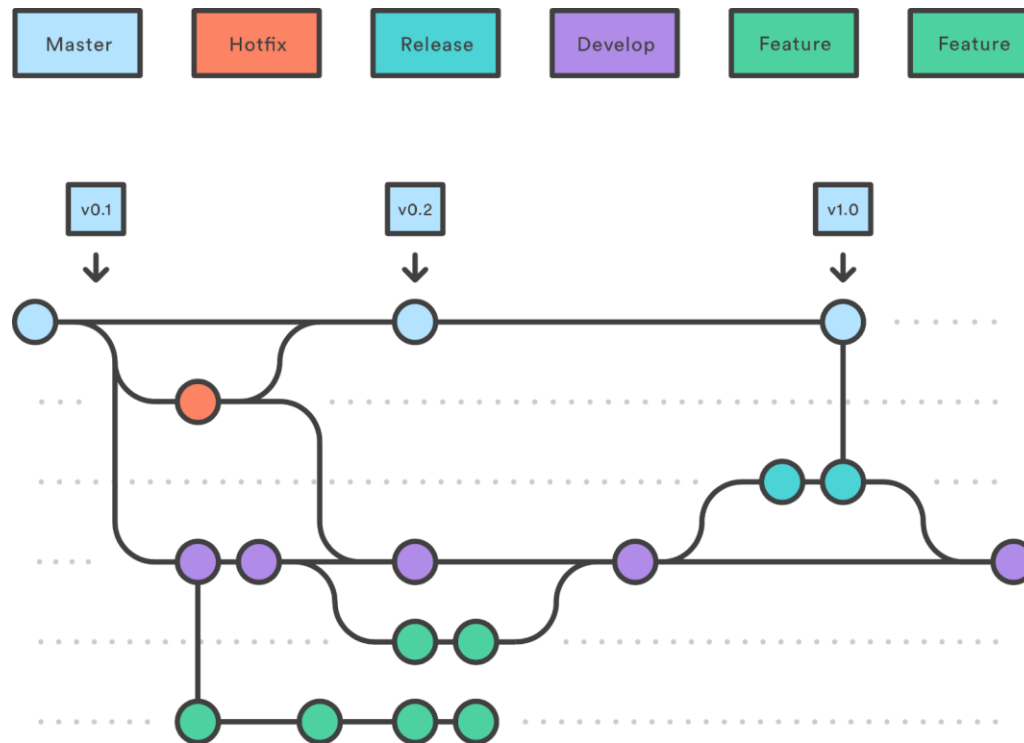
- ausreichend Features für ein Release in Develop, dann Abspaltung in Release (Feature Freeze)
- Jede Revision im Release stellt **Release-Candidate (RC)** dar
- nur noch Bugfixes oder Dokumentation einchecken
- Software fehlerfrei → Merge mit Master



Gitflow

➤ Hotfix:

- Schneller Patch der Software
- Hotfix-Branch wird vom Master abgespalten
- Nach Fertigstellung wieder zusammenführen



Wichtigste Befehle

- `git init`
- `git clone`
- `git add`
- `git status`
- `git commit`
- `git push`
- `git pull`
- `git checkout`
- `git branch`
- `git merge`

Live Demonstration



Neues Repository erstellen

```
$ git init [directory]
```

- Neues Repository im aktuellen Ordner
- Optionaler Parameter *directory*: Initialisiert Git-Repository in neuen Ordner *directory*
- Erstellt *.git*-Ordner mit allen Repository Informationen

Git Repositorys

```
$ ls .git  
HEAD          info/  
Config        objects  
Description   refs/  
Hooks/
```

- Wichtigste Datei zum manuellen Editieren: config

Git konfigurieren

Name und Mailadresse müssen gesetzt werden:

```
$ git config --global user.name "Sebastian Simon"
```

```
$ git config --global user.email "ssimon@informatik.uni-Leipzig.de"
```

- Mit `--global` wird Konfiguration in `~/.gitconfig` gespeichert

Git Status

```
$ git status
```

- Zeigt Status der Dateien im Repository an
- Gibt Hilfe welche Aktionen verfügbar sind

Dateien zum Repository hinzufügen

```
$ git add <file>
```

- Fügt Datei im nächsten Commit zum Repository hinzu (Staging Area)
- Alle Änderungen an der Datei bis zu diesem Zeitpunkt
 - Gibt es weitere Änderungen muss git add erneut ausgeführt werden

Commit erstellen

```
$ git commit [-m message]
```

- Erstellt einen Commit mit allen Änderungen von *git add*
- Jeder Commit braucht eine Commit Message
- Ohne *-m* wird Übersicht der Arbeitskopie und Staging Area angezeigt
- Staging Area wird danach zurückgesetzt

Commit History

```
$ git log
```

- Zeigt die Commit History an

Git History

```
$ git show [commit]
```

- Zeigt Details eines Commits an
- Commits werden über ihren Hash angegeben
- Ohne Argument wird der letzte Commit angezeigt

Git Checkout

```
$ git checkout <hash>
```

- Zeigt Projekt zum Zeitpunkt von Commit <hash>

```
$ git checkout master
```

- Geht zum aktuellsten Commit zurück
- Wichtig bevor neue Commits gemacht werden

Git Hosting

- Bisher haben wir nur offline gearbeitet
- Beispiele für Git Hosting Services:



GitHub



GitLab

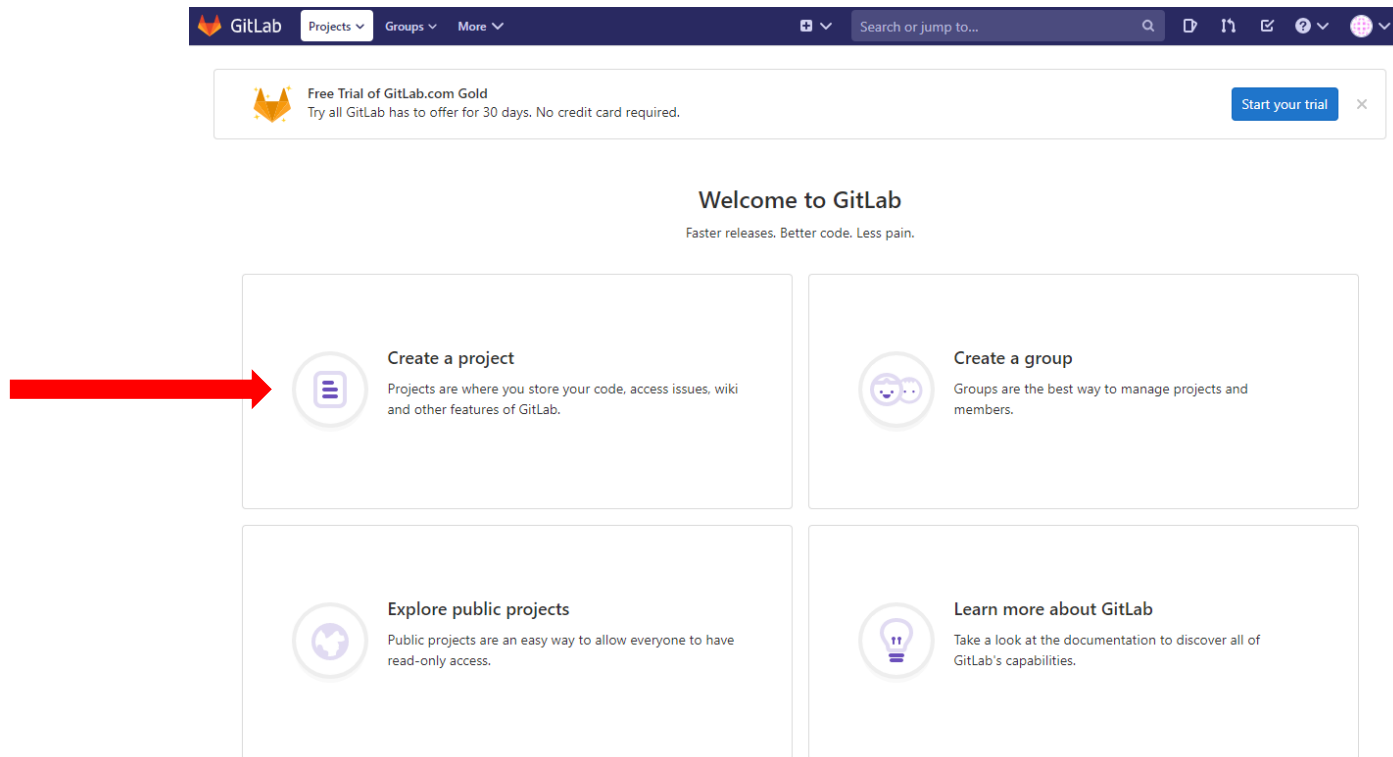


Gitea



Bitbucket


Repository auf GitLab anlegen




Repository auf GitLab anlegen




Create new project






Create blank project

Create a blank project to house your files, plan your work, and collaborate on code, among other things.




Create from template

Create a project pre-populated with the necessary files to get you started quickly.



Import project

Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.

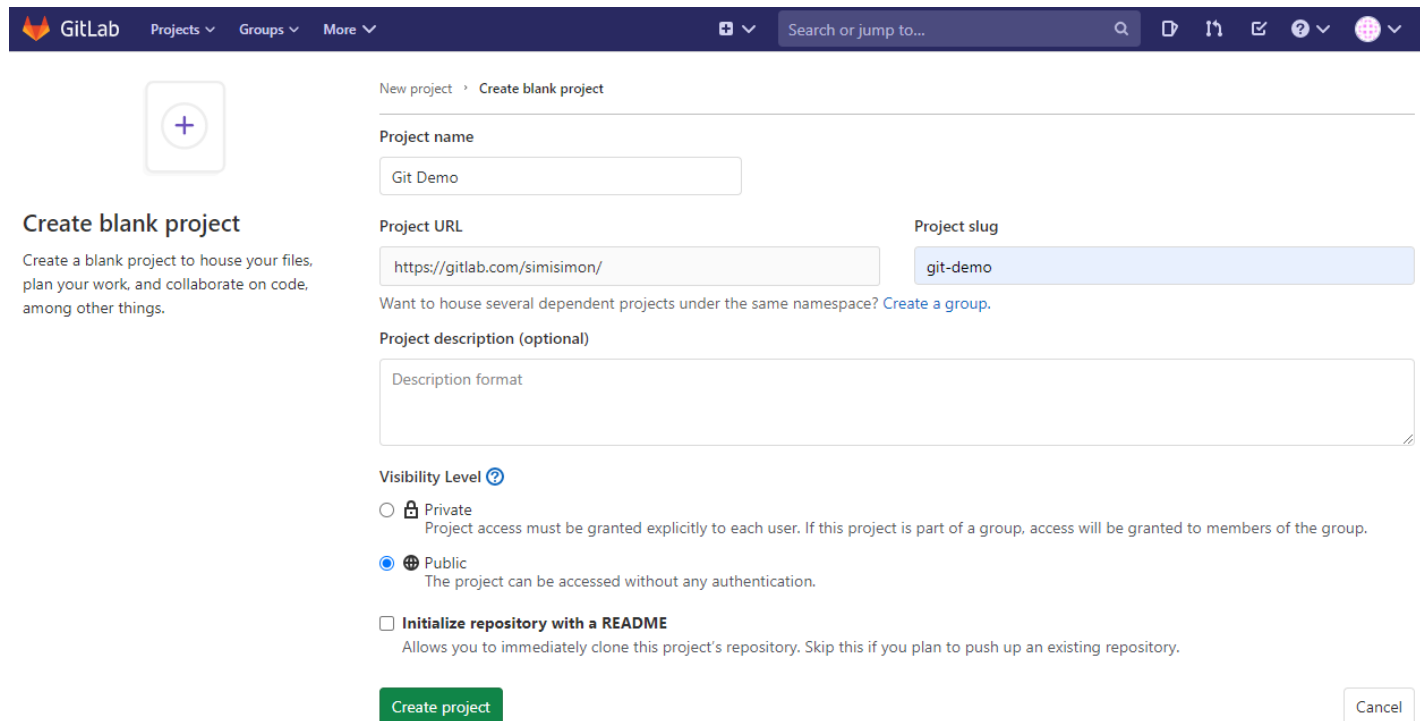


Run CI/CD for external repository

Connect your external repository to GitLab CI/CD.

You can also create a project from the command line. [Show command](#)

Repository auf GitLab anlegen



GitLab Projects Groups More

Search or jump to...

New project > Create blank project

Create blank project

Create a blank project to house your files, plan your work, and collaborate on code, among other things.

Project name

Git Demo

Project URL

https://gitlab.com/simisimon/

Project slug

git-demo

Want to house several dependent projects under the same namespace? [Create a group.](#)

Project description (optional)

Description format

Visibility Level ?

☐ Private
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.

☒ Public
The project can be accessed without any authentication.

☐ **Initialize repository with a README**
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project Cancel

Git Remotes

```
$ git remote add origin ....
```

```
$ git push -u origin master
```

```
$ git remote add <name> <url>
```

- Fügt ein Remote Repository als *name* hinzu
- Remotes können beliebige Orte sein (andere Rechner, Server, ...)
- Remotes haben jeweils das komplette Repository
- In der Praxis häufig ein Hauptrepository (Best practice: origin)
- *git remote [-v]* zeigt alle Remotes an

Git Push

```
$ git push -u origin master
```

- Schickt Commits zu einem anderen Repository
- -u (--upstream) setzt origin als Default Remote

Git Pull

```
$ git pull [<remote>]
```

- Holt Commits von einem anderen Repository und updated das lokale Repository
- Optional: Remote

Git Fetch

```
$ git fetch
```

- Lädt Commits runter
- Ändert keine lokalen Dateien

Git Merge

```
$ git merge <remote>/master
```

- Updated das lokale Repository mit den Commits eines anderen Repository

Git Clone

```
$ git clone <url>
```

- Erstellt lokale Kopie eines Remote-Repositorys
- Lokale Kopie hat einen Remote origin, der auf die URL des Remote-Repositorys zeigt
- Prozess wird auch als *Forking* bezeichnet

Git Branch

```
$ git branch <branchname>
```

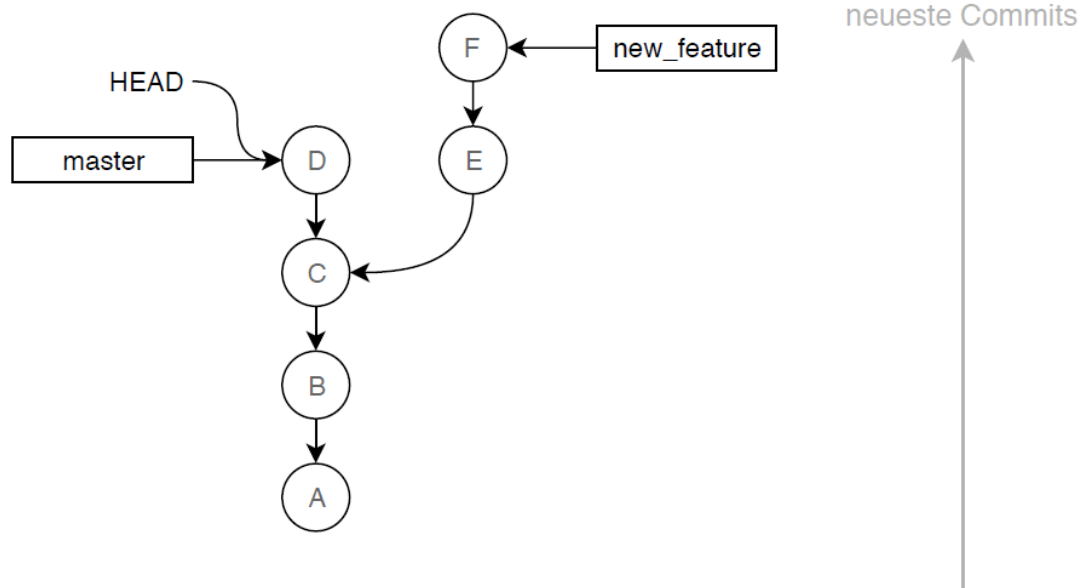
- *git branch* zeigt an, welche Branches im Repo existieren und in welchem wir gerade sind
- *git branch <branchname>* erstellt neuen Branch, der auf den aktuellen Commit Zeigt
- *git branch -d <branchname>* löscht einen Branch aus dem lokalen Repository
- *git checkout -b <branchname>*
= *git branch <branchname>* + *git checkout <branchname>*

Git Cherry Pick

```
$ git cherry-pick <ref>
```

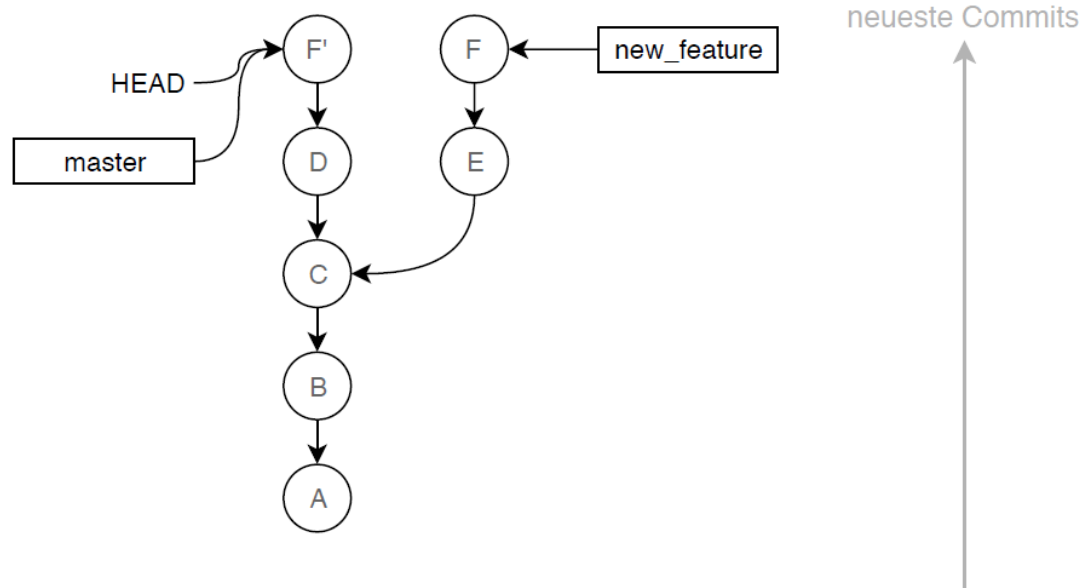
- Erstellt einen neuen Commit der eine exakte Kopie eines existierenden Commits ist

Git Cherry Pick



```
$ git cherry-pick F
```

Git Cherry Pick



```
$ git cherry-pick F
```

Git Reset

```
$ git reset <commit>
```

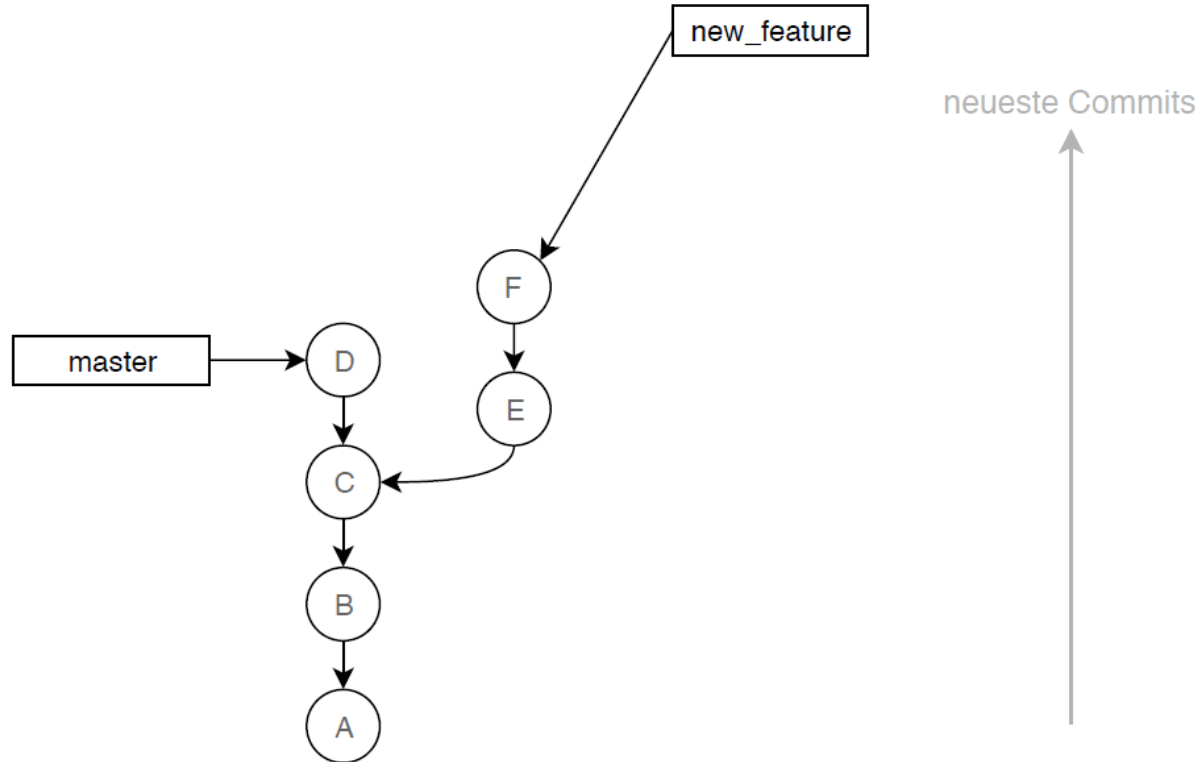
- Verschiebt den aktuellen Branch-Pointer auf *commit*
- Dateien werden dabei nicht verändert
- *git reset --hard commit* verändert die Dateien

Git Rebase

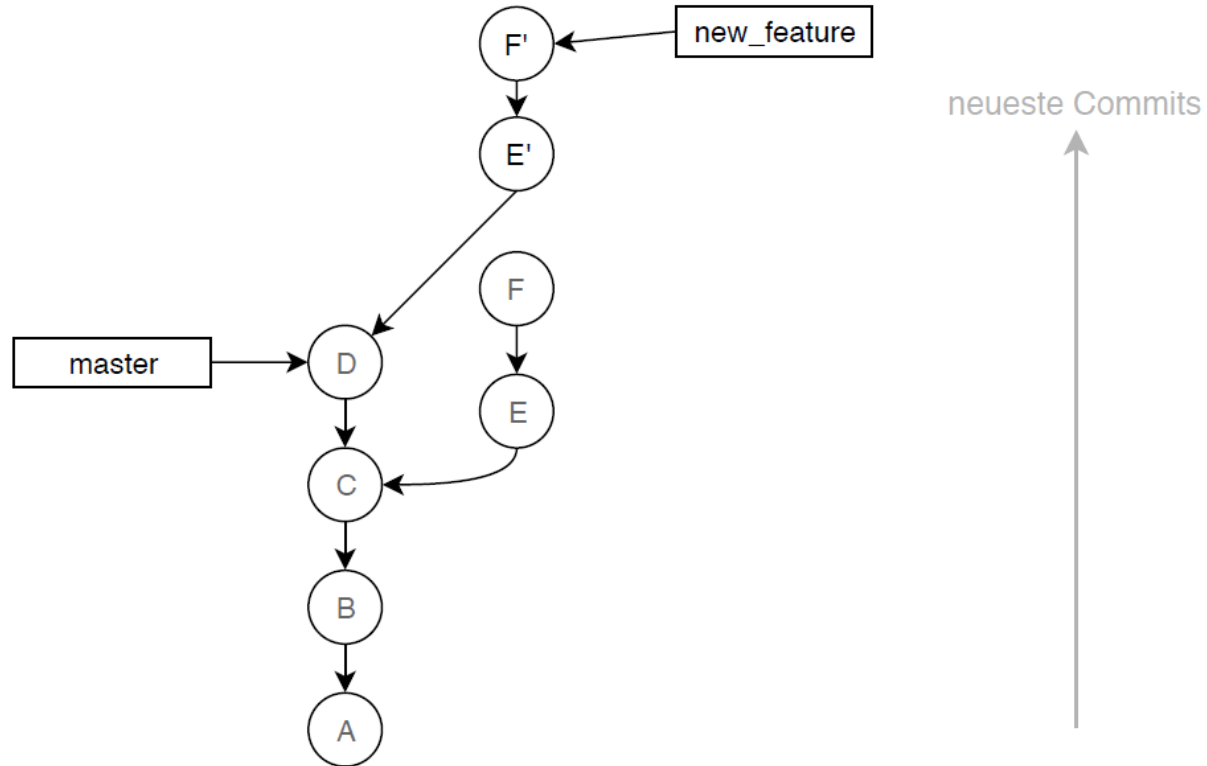
```
$ git rebase <ref>
```

- Mehrere cherry-picks um einen Branch von einem neuen Commit starten zu lassen
- Die alten Commits werden nicht gelöscht
- Rebase verändert History

Git Rebase



Git Rebase



Weitere nützliche Befehle

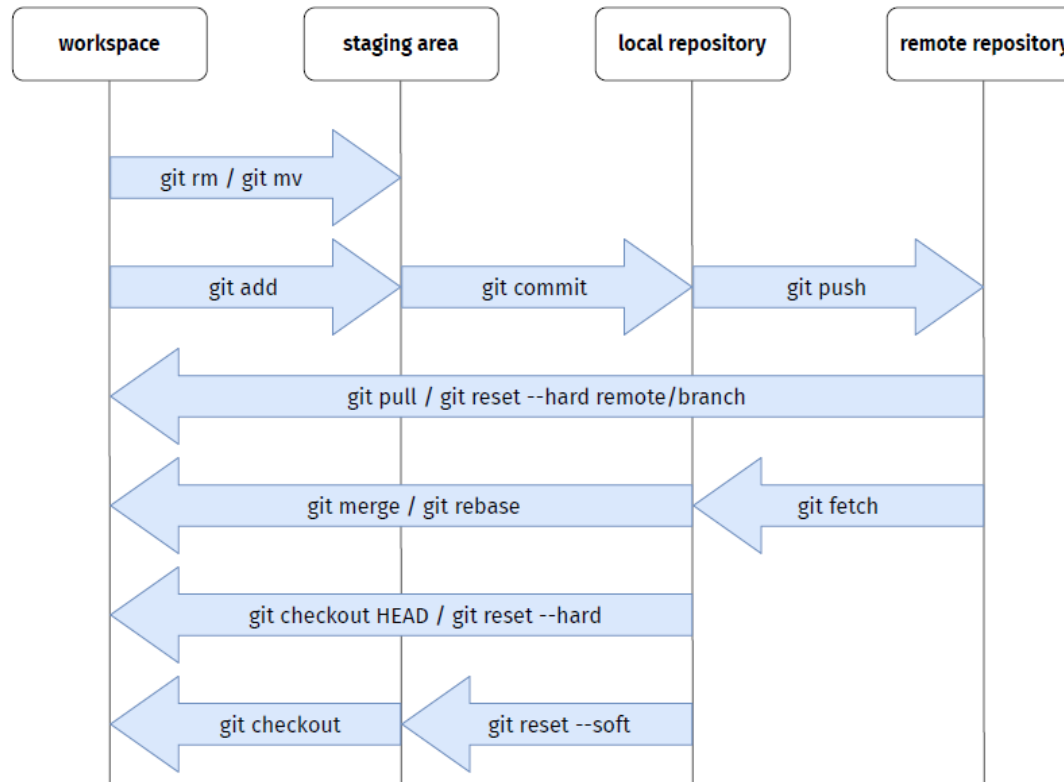
```
$ git tag
```

- Label für einen Commit
- Tags zeigen immer auf den selben Commit
- Beispiel: Tag für Version 1.0

```
$ git blame <file>
```

- Zeigt für jede Zeile an, wer sie zuletzt geändert hat

Übersicht



Tutorials

- Einführung in Git: <https://git-scm.com/docs/gittutorial>
- Weitere Infos:
 - <http://think-like-a-git.net/>
 - <https://github.com/pluralsight/git-internals-pdf>