HERZLICH WILLKOMMEN

WARUM IST GIT FÜR DICH WICHTIG?

- Du kannst den Zustand von vor ein paar Tagen wiederherstellen
- Du kannst an Open-Source-Projekten mitarbeiten
- Du behältst den Überblick, wer wann wie den Quellcode entwickelt hat
- Die Arbeit im Team ist einfacher koordinierbar

WAS ERWARTET DICH IN DIESEM KURS?

- Git einrichten & konfigurieren
- Git lokal (Commits, Branches, Mergen, Rebase)
- Git Remote (z.B. mit Github)
- Git Tipps & Tricks

VIELEN DANK
VIEL SPAß IM KURS!

GIT GRUNDLAGEN

EIN NEUES REPOSITORY ANLEGEN: GIT INIT

GIT INIT: EIN NEUES REPOSITORY ANLEGEN

```
git init
```

- erstellt ein neues Repository im aktuellen Verzeichnis
- alle von git benötigten Dateien werden im Unterorder `.git` gespeichert

```
git status
```

- Zeigt uns an, welche Dateien ggf. geändert wurden

ÄNDERUNGEN DURCHFÜHREN: STAGING AREA & COMMITS

WAS IST EIN COMMIT?

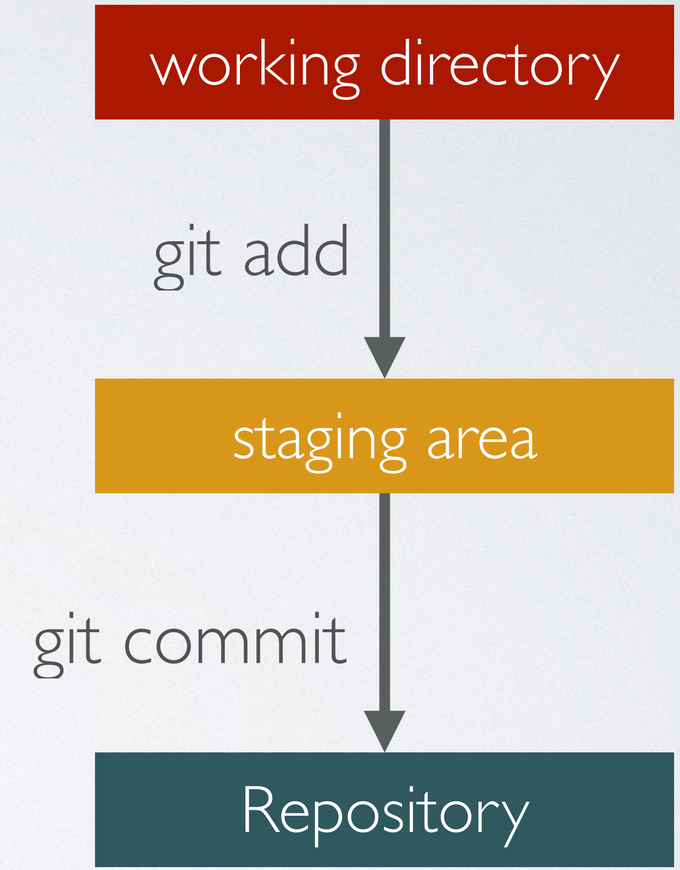
- Mit einem Commit wird der Zustand des Projektes zu einem bestimmten Zeitpunkt abgespeichert

- Ein Commit entspricht quasi einer „Version“ von unserem Projekt

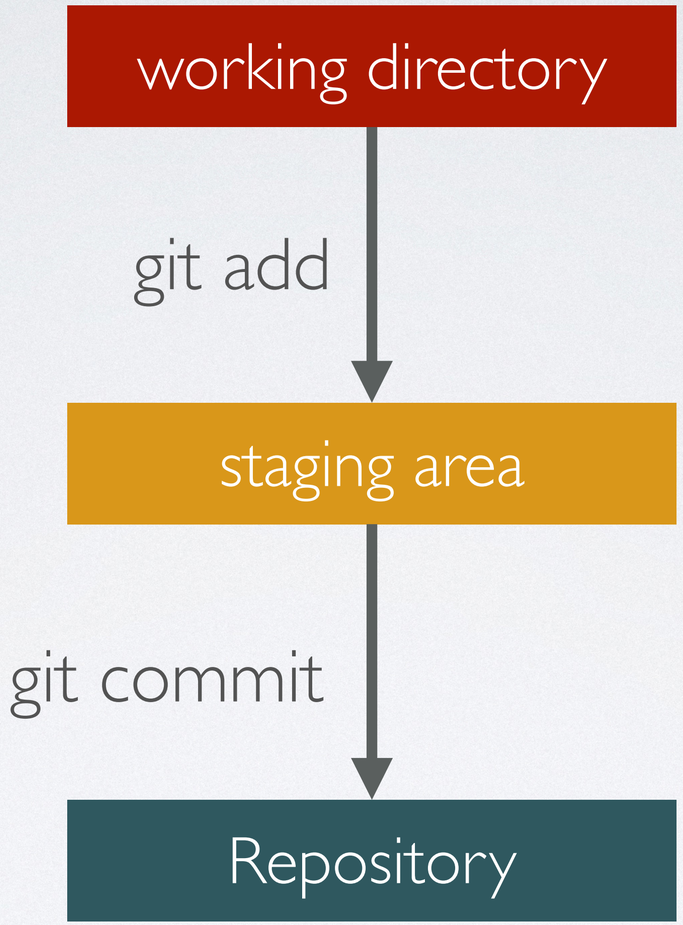
GIT ADD: STAGING AREA

In der Staging-Area werden Commits vorbereitet.

```
git add <file>
git add <folder>
```



ÄNDERUNGEN DURCHFÜHREN: STAGING AREA & COMMITS


working directory
git add
staging area
git commit
Repository

GIT COMMIT

`git commit`

Erstellt einen neuen Commit aus allen gestageten Änderungen.
Best Practice: eine atomare Änderung pro Commit

`git commit -m "<message>"`

Shortcut, um die Commit-Message direkt einzugeben

GIT LOG:
REPOSITORY-HISTORY AUSGEBEN

GIT LOG: HISTORIE ANZEIGEN


```
git log
```

Zeigt die Commit-Historie an.

Ein Commit wird identifiziert über seinen SHA1-Hash.

COMMITS ÜBERSCHREIBEN

COMMITS ABÄNDERN

```
git commit --amend
```

Letzen Commit überschreiben.

ACHTUNG: Vorsichtig verwenden.
Am besten nur bei lokalen Änderungen.

UNTERSCHIEDE ANZEIGEN: GIT DIFF

GIT DIFF: ÄNDERUNGEN ANZEIGEN

`git diff`

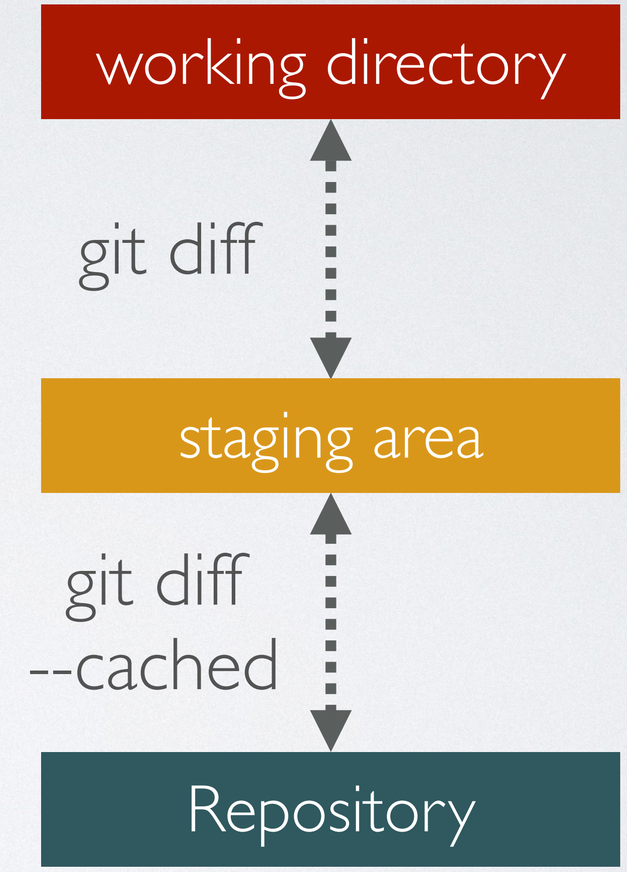
zeigt Änderungen zur Staging Area an

`git diff --cached`

zeigt gestagete Änderungen an

`git diff <hash1> <hash2>`

Zeigt die Änderungen zwischen zwei Commits.



ÄNDERUNGEN ZURÜCKNEHMEN: GIT RESET

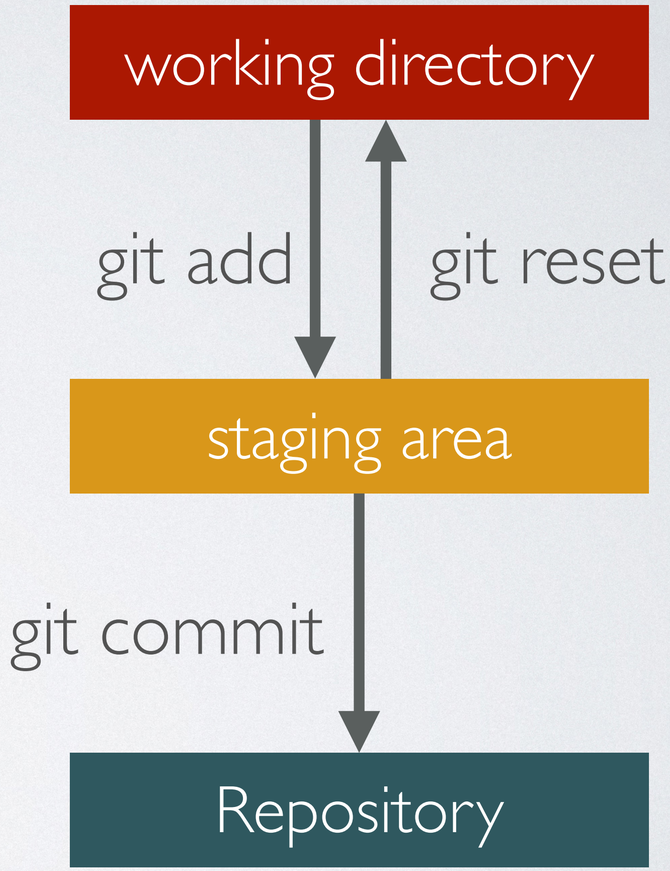
GIT RESET (STAGING-AREA)

`git reset`

Nimmt alle Änderungen aus der Staging-Area raus.

`git reset <filename>`

Nimmt Änderungen aus der Staging-Area raus.



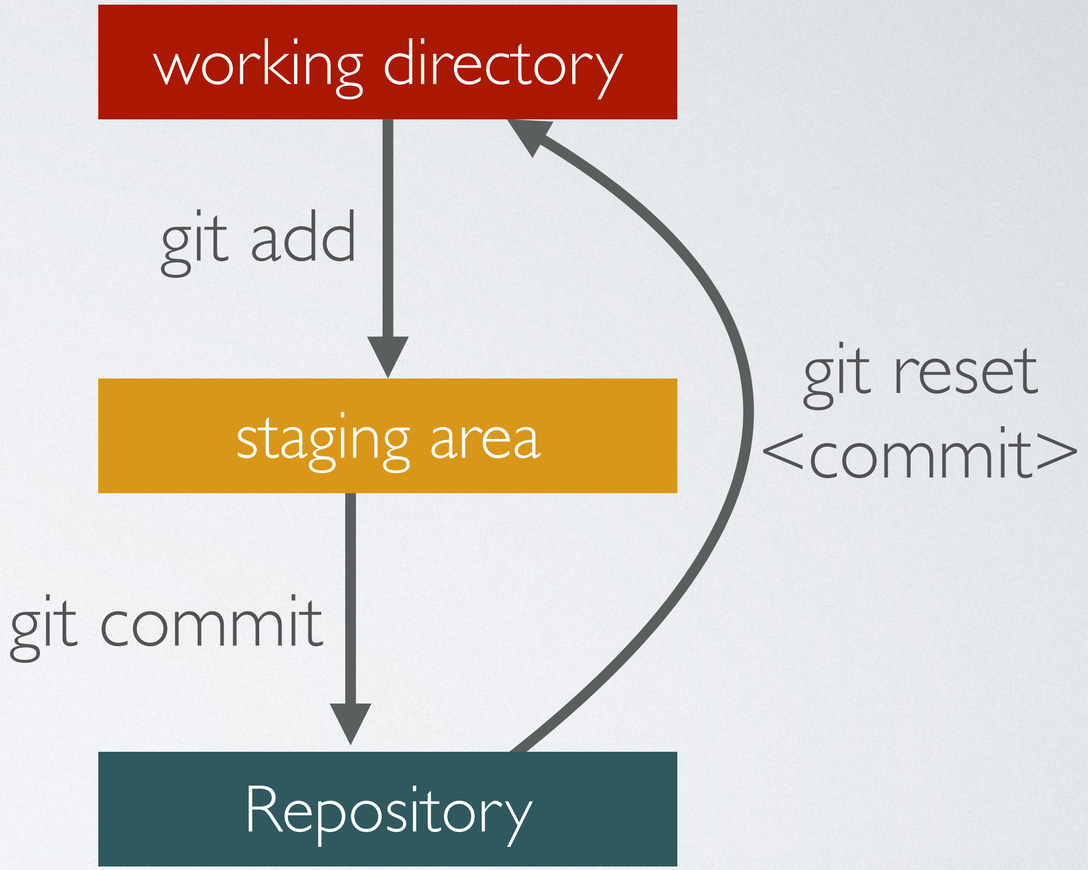
GIT RESET (COMMITS)

`git reset <commit hash>`

Löscht alle Commits nach <commit hash>.
Alle Änderungen bleiben im working directory bestehen.

`git reset --hard <commit hash>`

Löscht alle Commits nach <commit hash>.
Änderungen werden verworfen.



ZU EINEM COMMIT SPRINGEN: GIT CHECKOUT

GIT CHECKOUT

`git checkout <commit hash>`

Stellt den Zustand von einem Commit wieder her.

Achtung: git versucht, lokale Änderungen zu übernehmen.
Dies kann zu Konflikten führen.

`git checkout master`

Springt zurück zum master.

WAS GENAU BEDEUTET „HEAD“?

EXPERTENWISSEN: NAVIGATION

HEAD: ist der gerade ausgecheckte Commit

HEAD~1: ist der Parent vom gerade ausgecheckten Commit

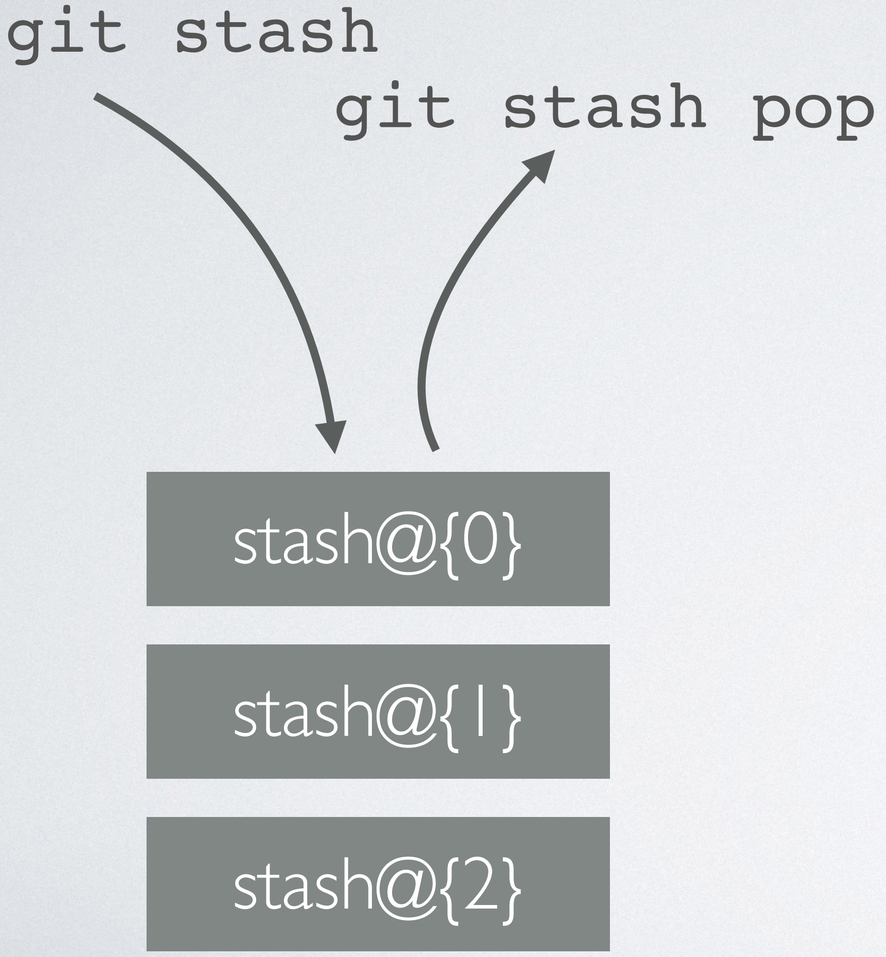
HEAD~2: ist der Parent vom Parent vom gerade ausgecheckten Commit

EXPERTENWISSEN:
GIT STASH

GIT STASH

- Git Stash:
  - Erlaubt dir, den aktuellen Zustand vom Working directory und der Staging Area zwischenzuspeichern
  - Beispiel-Workflow: Du speicherst den Zustand im Stash, schaust dir einen alten Commit an, wechselst wieder zum aktuellen Stand, und liest den alten Zustand aus dem Stash aus

EXPERTENWISSEN:
GIT STASH / GIT STASH POP



`git stash`

sichert den aktuellen Stand vom working directory

`git stash pop`

stellt den letzten Stand vom working directory wieder her

`git stash list`

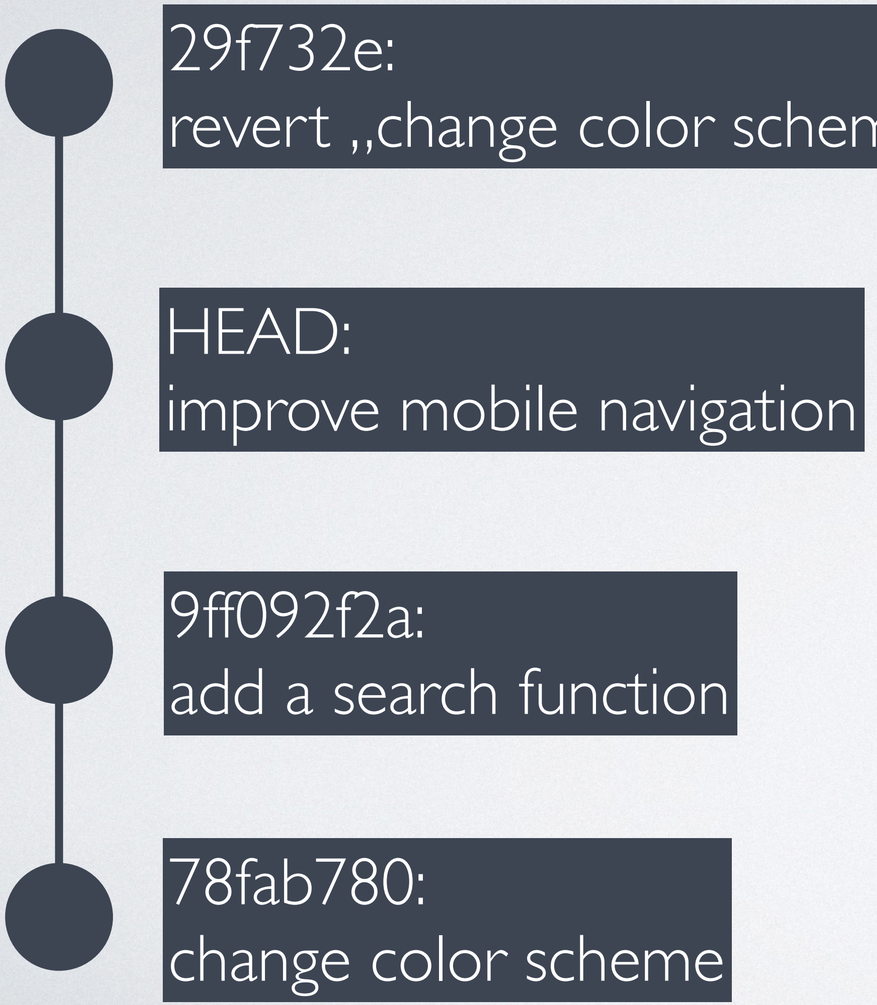
zeigt alle Stashes an

`git diff stash`

zeigt den diff des letzten Stashes an

ÄNDERUNGEN RÜCKGÄNGIG MACHEN: GIT REVERT

GIT REVERT: ÄNDERUNG RÜCKGÄNGIG MACHEN




```
git revert <commit hash>
```

Erstellt einen neuen Commit, der die Änderungen rückgängig macht.

DATEIEN IGNORIEREN: .GITIGNORE

EXPERTENWISSEN: .GITIGNORE

`.gitignore` enthält eine Liste an Pfaden, die von git ignoriert werden

Änderungen an diesen Dateien haben dann keine Auswirkung auf `git diff`, git add, etc.

`.gitignore` sollte mit commitet werden

DATEIEN LÖSCHEN / UMBENENNEN

DATEIEN LÖSCHEN UND UMBENENNEN

`Git rm:` Löscht eine Datei

`Git mv:` Bewegt eine Datei

VERÄNDERUNGEN EINER DATEI ANZEIGEN

EXPERTENWISSEN: GIT BLAME

```
git blame <filename>
```

Zeigt Zeile für Zeile an, in welchem Commit diese zuletzt geändert wurde.

```
git blame --color-lines <filename>
```

gleiche Commits farblich hervorheben

BRANCHES

BRANCHES

GIT BRANCHES

`git branch`

zeigt alle vorhandenen Branches an

`git branch <branch name>`

legt einen neuen Branch an

`git checkout <branch name>`

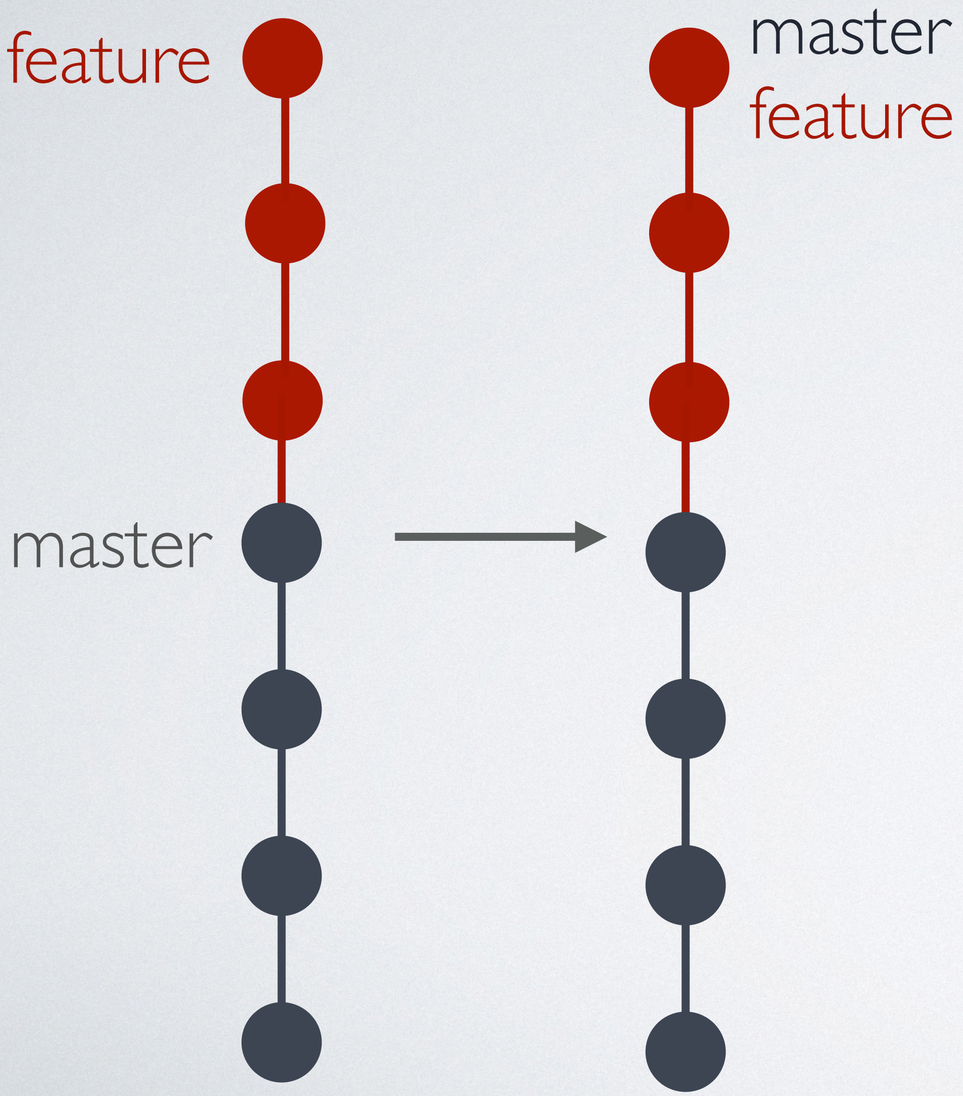
wechselt zu einem Branch

Shortcut: **`git checkout -b <branch name>`**

legt einen neuen Branch an, und wechselt direkt dorthin

BRANCHES ZUSAMMENFÜHREN: FAST-FORWARD-MERGE

GIT MERGE: FAST FORWARD




```
git merge <branch>
```

Versucht den Branch **branch** mit dem aktuellen Branch zusammenzuführen (zu „mergen“).

Hier gezeigt: ein „fast-forward“ Merge.

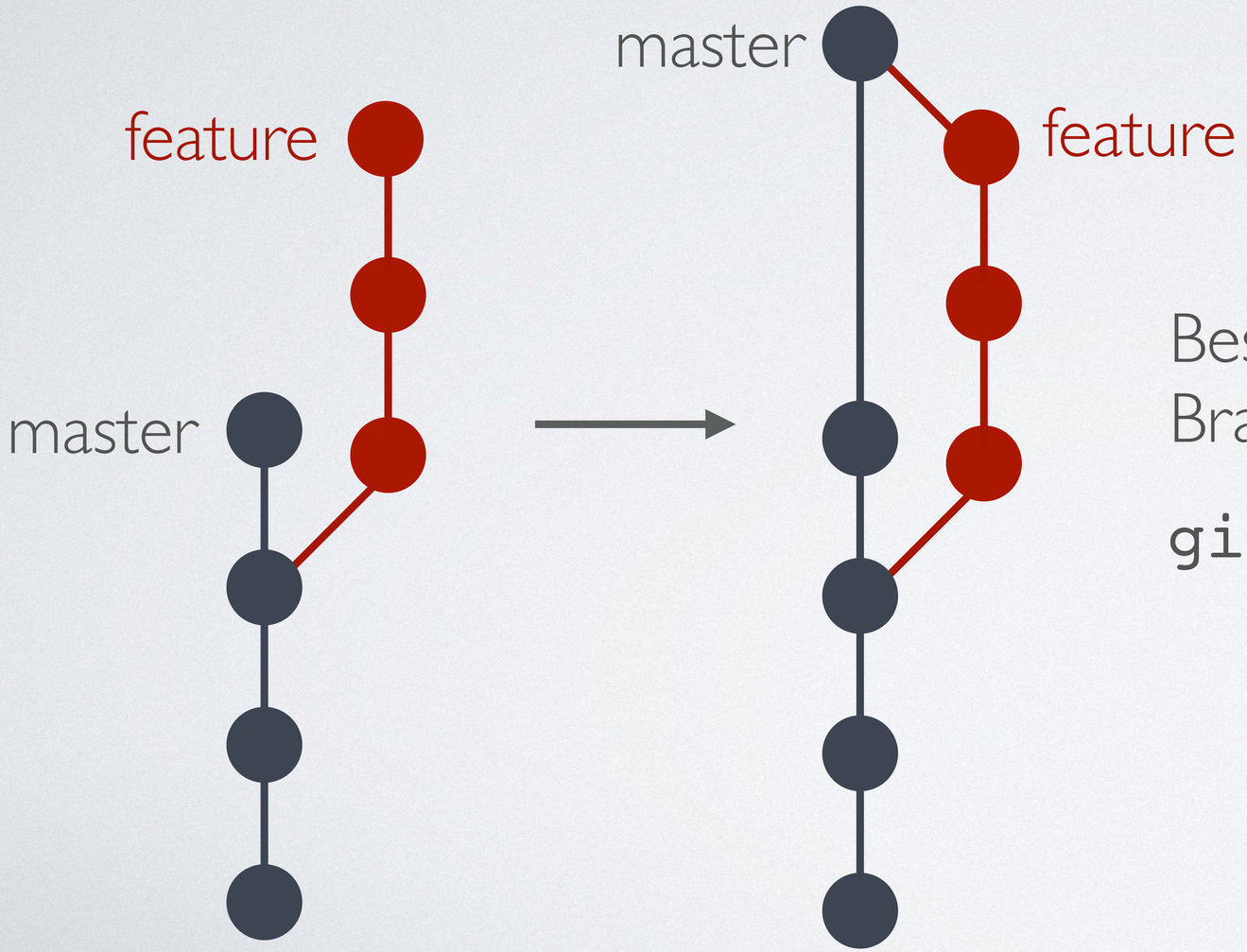
```
git checkout master
git merge feature
```

Best practice: Nach dem Mergen den Branch löschen

```
git branch -d feature
```

BRANCHES ZUSAMMENFÜHREN: 3-WAY-MERGE

GIT MERGE: 3-WAY MERGE



Wenn kein Fast-Forward-Merge möglich ist, erstellt git einen Merge-Commit.

Best practice: Nach dem Mergen den Branch löschen

```
git branch -d feature
```

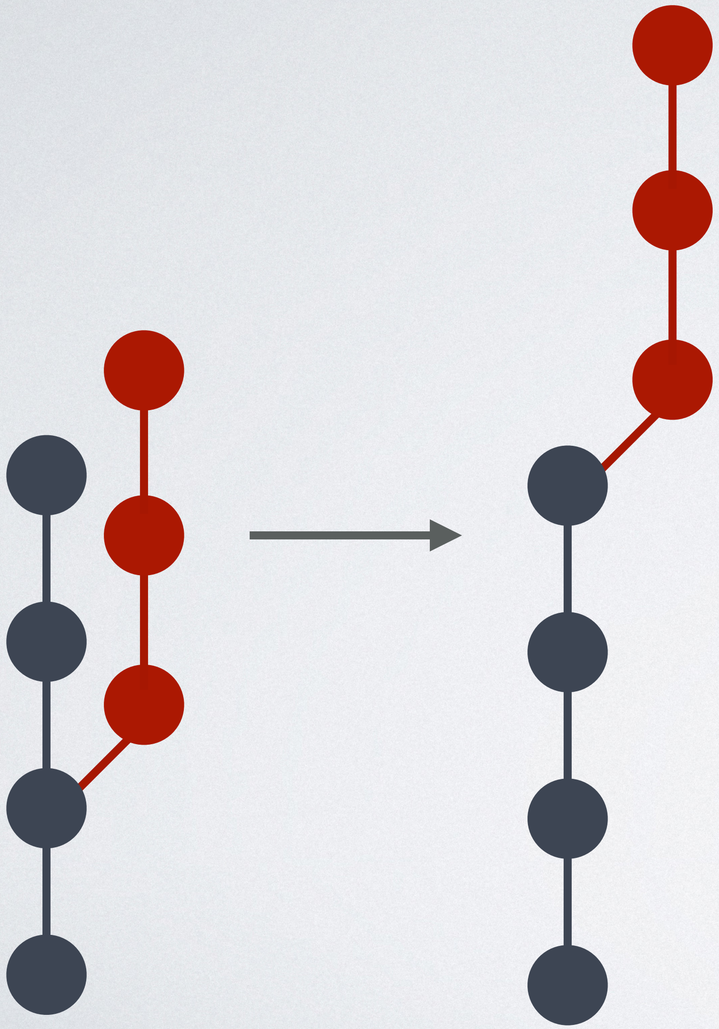
PROBLEME BEIM MERGEN: MERGE-KONFLIKTE

MERGE-KONFLIKTE

- Manchmal kann git die Änderungen für uns nicht automatisch mergen
- Das passiert insbesondere, wenn z.B. die gleiche Zeile in 2 Branches abgeändert wurde
- Diese Konflikte müssen wir dann manuell beheben

GIT REBASE

GIT REBASE



`git rebase <branch>`

baut die Commit-Historie basierend auf einem anderen Branch auf

`git rebase -i <branch>`

erlaubt es, dabei:

- Commits anzupassen (**`edit`**)
- Commit-Messages zu editieren (**`fixup`**)
- mehrere Commits zusammenzufassen (**`squash`**)

MERGE-STRATEGIEN

GIT MERGE

`git merge --no-ff`

Verhindert, dass git einen fast-forward-Merge durchführt.
Es wird also immer ein Merge-Commit erstellt.

`git merge --ff-only`

Erzwingt einen fast-forward Merge.

NUR FAST-FORWARD-MERGE

- Das Repository ist „in einem Strang“
- Dadurch ist einfach zu sehen, wie die Änderungen aufeinander aufbauen

NUR THREE-WAY-MERGE

- Jedes Feature wird in einem eigenen Branch entwickelt
- Es ist einfach zu sehen, wie welches Feature entwickelt wurde

GIT TAG

GIT TAG: RELEASES MARKIEREN

`git tag -a v<x.y>`

einen neuen Tag für Version x.y erstellen

`git tag`

alle bestehenden Tags anzeigen

GIT: REMOTE REPOSITORIES

GIT: REMOTE REPOSITORIES

GIT CLONE: REMOTE REPOSITORIES

`git clone <url>`

erstellt eine lokale Kopie des master-Branches eines Repositories

`git remote -v`

zeigt an, welche remote Repositories getrackt werden

GIT: REMOTE REPOSITORIES
GIT FETCH

GIT FETCH


```
git fetch
```

lädt Commits und Branches von einem remote Repository herunter

```
git log origin/<branch>
git merge origin/<branch>
```

SHORTCUT: GIT PULL

`git pull`

führt ein `git fetch` durch,
gefolgt von einem `git merge` (für den ausgecheckten Branch)

GITHUB

BONUS:
WELCHE DATEIEN SOLLTEST DU COMMITTEN?

BONUS: WELCHE DATEIEN COMMITTEN?

- Das ist abhängig von der Programmiersprache
  - Generell gilt:
    - Alles was gebraucht wird, um das Projekt zu bauen

BEISPIEL: PHP

- PHP:
    - Committen:
        - Unseren Anwendungscode
        - composer.json
        - composer.lock
    - Auslassen:
        - Ordner: vendor/

BEISPIEL: NODEJS

- NodeJS:
  - Committen:
    - Unseren Anwendungscode
    - package.json
    - package-lock.json
  - Auslassen:
    - Ordner: node_modules/

BEISPIEL: C++

- C++:
    - Committen:
        - Unseren Anwendungscode
        - Makefile
        - ./configure.sh
    - Auslassen:
        - Ordner: out/

BONUS:
GIT BISECT

BONUS: GIT BISECT

- git bisect hilft uns, einen problematischen Commit zu finden
- Hierbei wird eine binäre Suche verwendet!
- Beispiel: Bei 1024 Commits brauchen wir nur 10 Durchläufe, um den problematischen Commit zu finden!
- **Verwendung:**
  - git bisect start
  - git bisect bad [commit-id]
  - git bisect good [commit-id]
  - git bisect reset

SCHLUSSWORTE

SCHLUSSWORTE

- Du beherrscht jetzt git:
  - Sowohl lokal
  - Als auch in Kombination mit Github

WIE GEHT ES JETZT WEITER?

- GUI:
  - SourceTree
  - Viele IDEs haben git integriert:
    - Xcode
    - IntelliJ / PHPStorm / PyCharm / …

VIELEN DANK, DASS ICH DIR GIT BEIBRINGEN DURFTE!