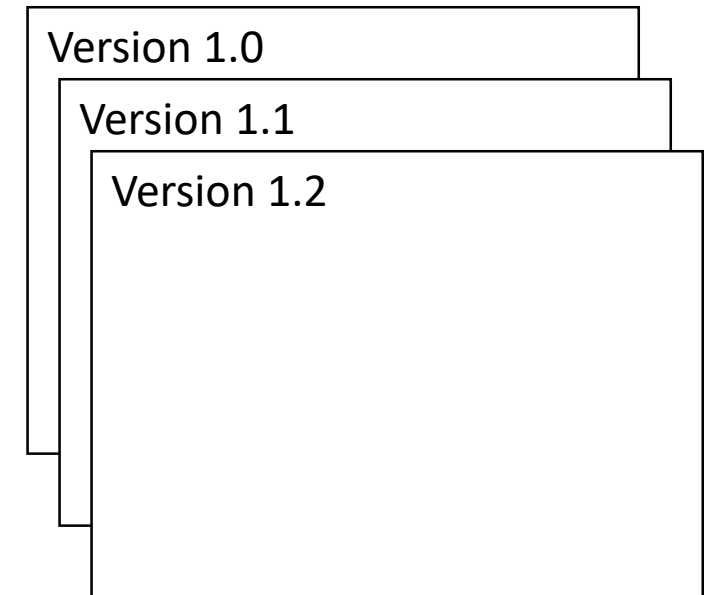




# Versionsverwaltung

- Änderungen von Dateien protokollieren
- früher Zustand
- Wer hat was, wann geändert



# Geschichte

- Linux Kernel
- BitKeeper
- Git ab 2005
- Ziele:
  - Geschwindigkeit
  - Einfaches Design
  - Nicht-linearer Entwicklung (tausende parallele Entwicklungszweige)
  - Vollständig dezentrale Struktur
  - Große Projekt

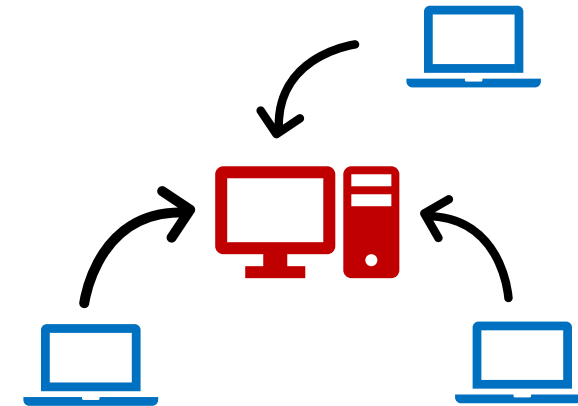


*Linus Torvalds*

# Versionsverwaltungssysteme

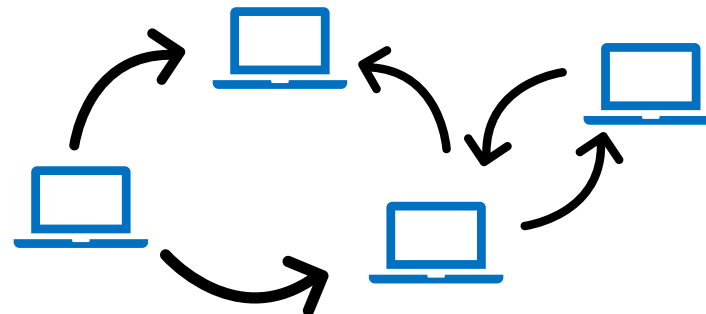
## Zentrale Systeme

- CVS
- Subversion (SVN)
- Team Foundation Version Control



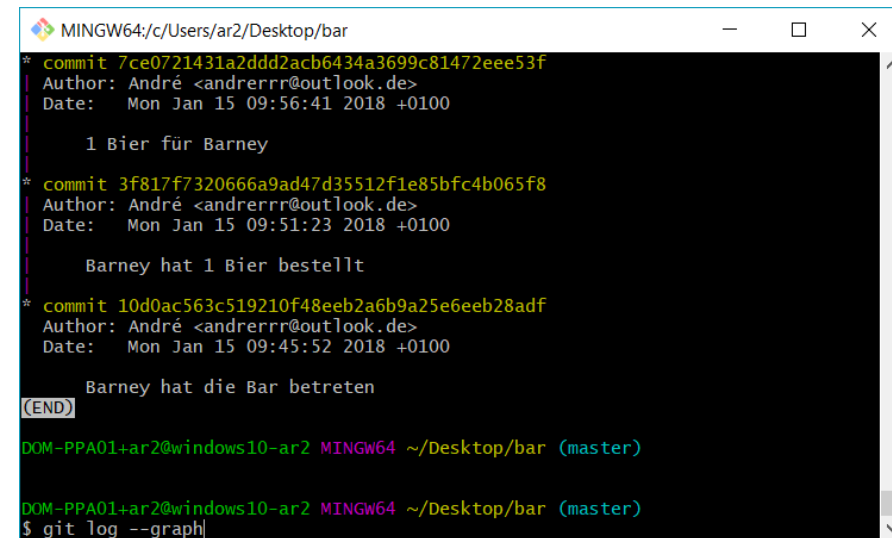
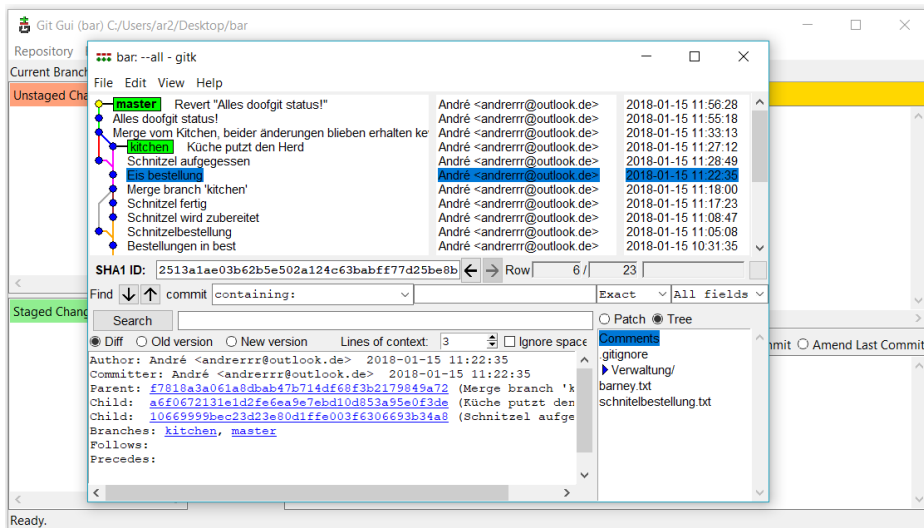
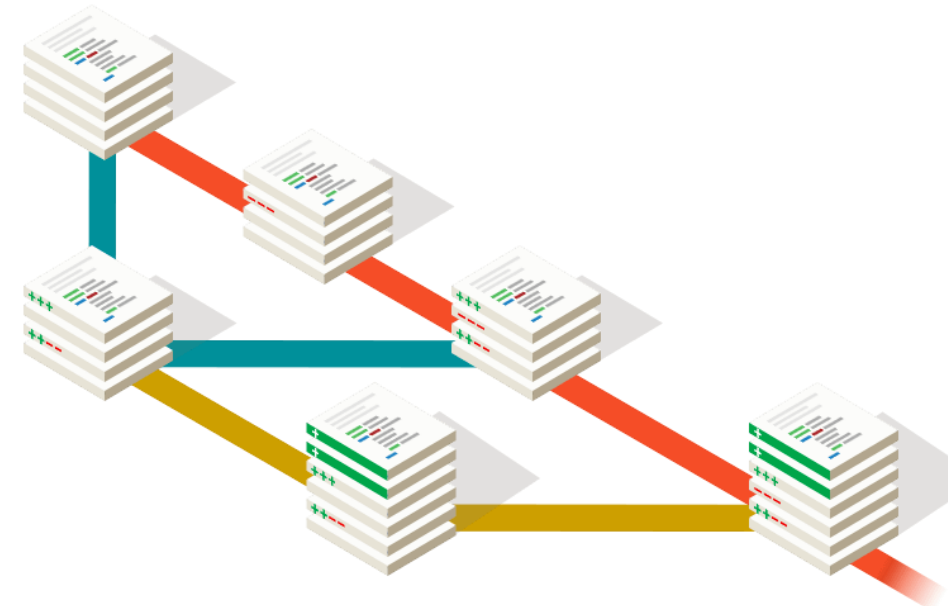
## Verteilte Systeme

- Mercurial
- Git



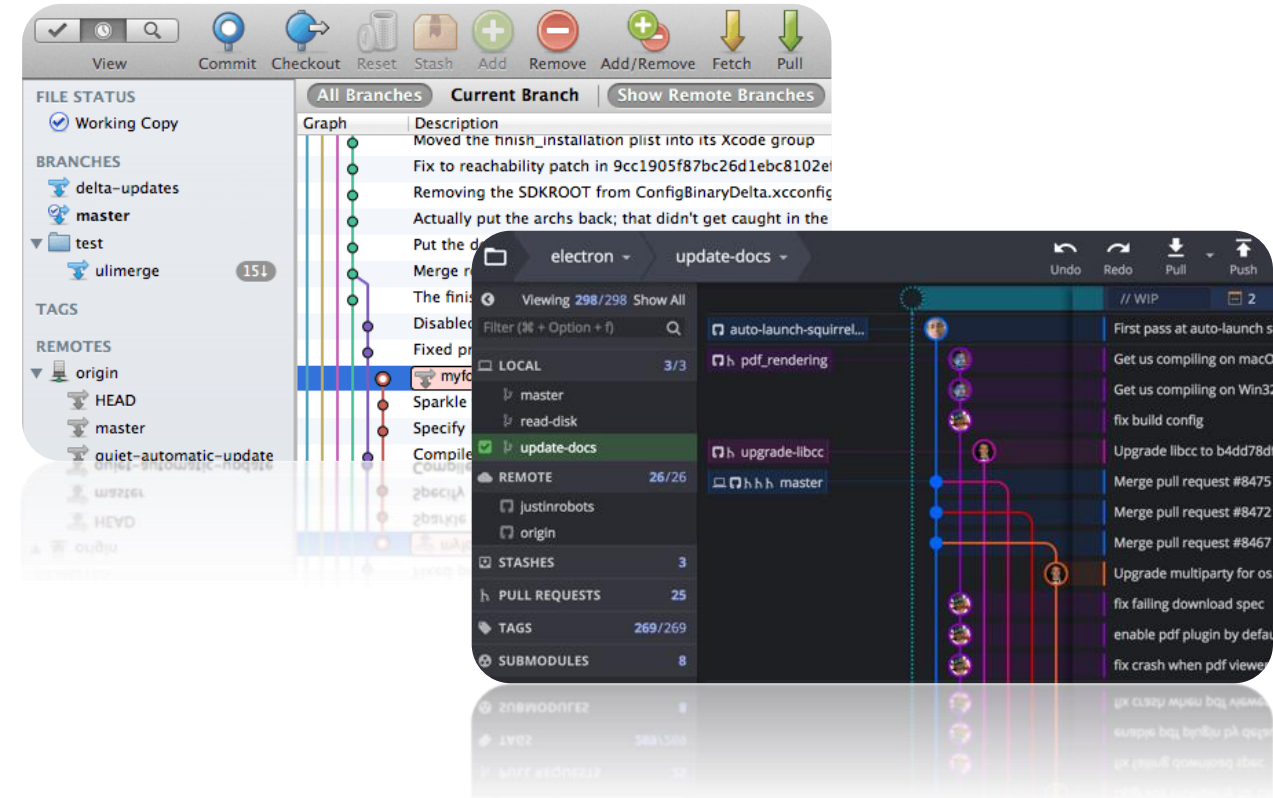
# Git Client

- <https://git-scm.com>
- Git GUI
- Git Bash



# Weitere GUI Client

- Visual Studio
- SourceTree [source-treeapp.com](https://source-treeapp.com)
- GitHub Desktop [desktop.github.com](https://desktop.github.com)
- TortoiseGit [tortoisegit.org](https://tortoisegit.org)
- GitKraken [gitkraken.com](https://gitkraken.com)
- ... und viele weitere: <https://git-scm.com/downloads/guis>



# Remote Repository

## Online Service

- GitHub
- Bitbucket
- GitLab
- Visual Studio Team Services

## On Premise Hosting

- Team Foundation Server
- GitLab
- Gitolite
- Gitstack
- Bonobo

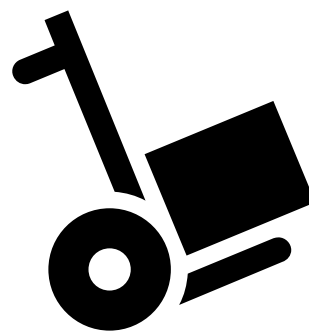


# Git repository



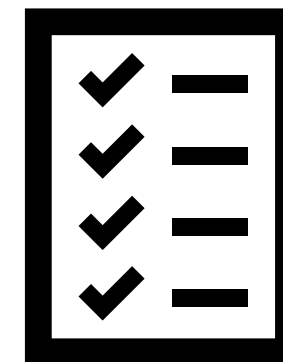
**Arbeitsverzeichnis**  
*working directory*

add



**Staging Area**  
*index*

commit



**Local Repository**  
*Commit History*

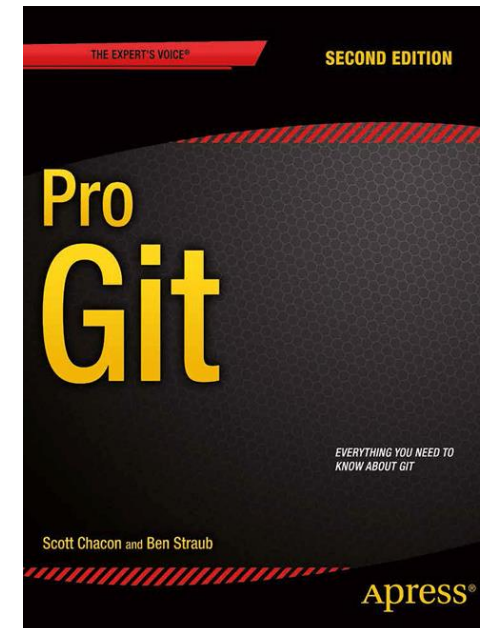


# Hilfe

```
git help  
git help <suchwort>
```

Zeigt Hilfe zu Gitbefehlen an

Kostenloses Buch: [git-scm.com/book/en/v2](https://git-scm.com/book/en/v2)

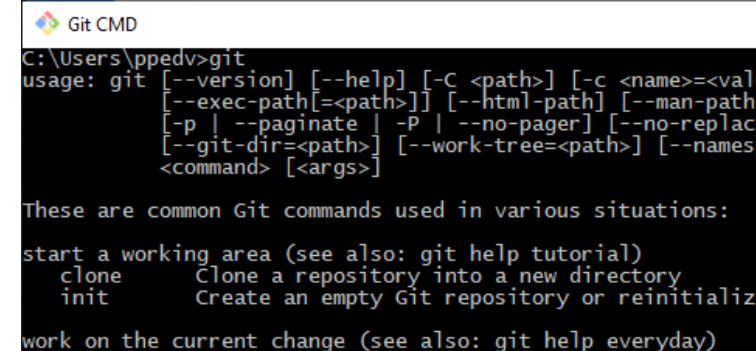


# Git Bash / CMD / PS

- Ersteinrichtung:

```
git config --global user.name "John Doe"
```

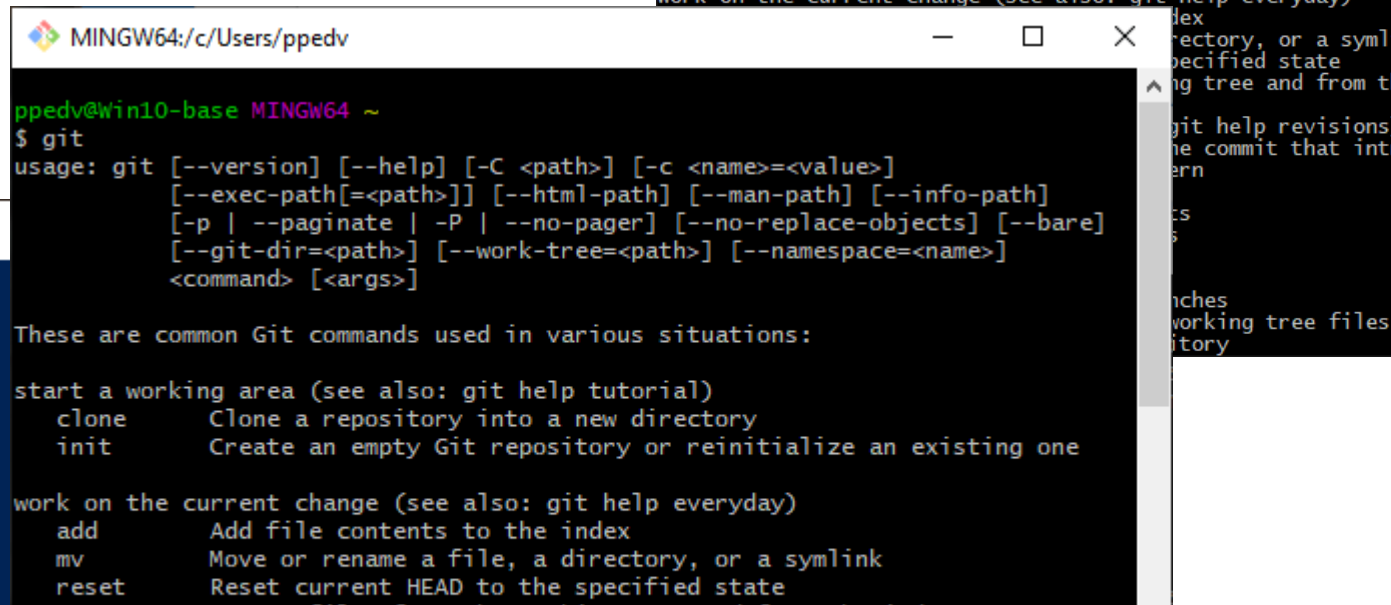
```
git config --global user.email johndoe@example.com
```



```
Git CMD
C:\Users\ppedv>git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-objects]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        <command> [<args>]

These are common Git commands used in various situations:


start a working area (see also: git help tutorial)
    clone      Clone a repository into a new directory
    init       Create an empty Git repository or reinitialize an existing one
work on the current change (see also: git help everyday)
```



```
MINGW64:/c/Users/ppedv
ppedv@win10-base MINGW64 ~
$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
    clone      Clone a repository into a new directory
    init       Create an empty Git repository or reinitialize an existing one
work on the current change (see also: git help everyday)
    add        Add file contents to the index
    mv         Move or rename a file, a directory, or a symlink
    reset      Reset current HEAD to the specified state
```

 Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

PS C:\Users\ppedv> git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        <command> [<args>]
```

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

# Repository erstellen

```
git init <name>
```

- Erstellt neues Verzeichnis
- .git versteckt

```
MINGW64:/c/Users/ar2/source/HalloGit

DOM-PPA01+ar2@windows10-ar2 MINGW64 ~/source
$ git init HalloGit
Initialized empty Git repository in C:/Users/ar2/source/HalloGit/.git/

DOM-PPA01+ar2@windows10-ar2 MINGW64 ~/source
$ cd HalloGit/

DOM-PPA01+ar2@windows10-ar2 MINGW64 ~/source/HalloGit (master)
$ ls

DOM-PPA01+ar2@windows10-ar2 MINGW64 ~/source/HalloGit (master)
$ ls -a
./ ../ .git/

DOM-PPA01+ar2@windows10-ar2 MINGW64 ~/source/HalloGit (master)
$ |
```

# Status

`git status`

Zeigt den Status des Arbeitsverzeichnis

- Untracked files
  - Mit *git add* hinzufügen
- Tracked files status:
  - new file, renamed, deleted, modified

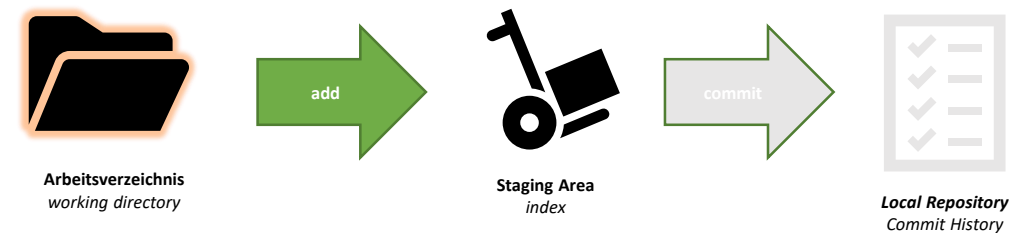
```
MINGW64:/c/Users/ar2/Desktop/hallowelt
DOM-PPA01+ar2@windows10-ar2 MINGW64 ~/Desktop/hallowelt (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   datei1
        deleted:    datei2
        renamed:    datei3 -> datei_drei

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        datei5

DOM-PPA01+ar2@windows10-ar2 MINGW64 ~/Desktop/hallowelt (master)
$
```

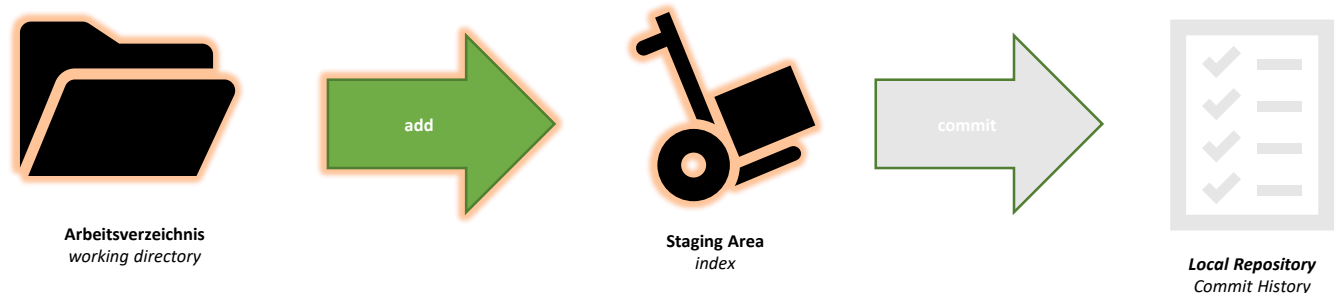


# Neue Dateien erfassen

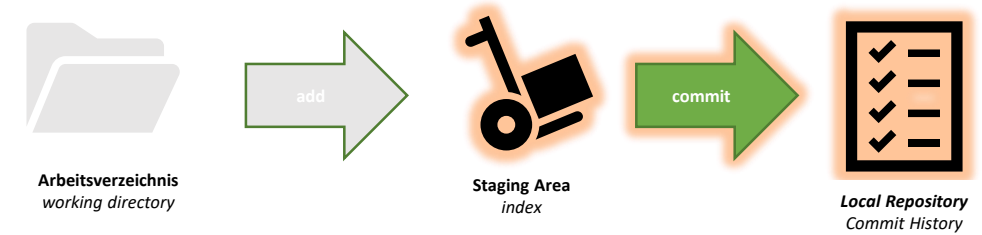
```
git add <dateiname>  
git add .
```

Fügt eine oder mehrere Datei zur Stageing Area/Index hinzu

- Status: new file
- Noch nicht ‚gesichert‘



# Einchecken



```
git commit
```

```
git commit -m „Fixed Y2k bug“
```

Übernimmt alle Änderungen ins Repository

- Status: nothing to commit, working tree clean
- Commit Message ist Pflicht

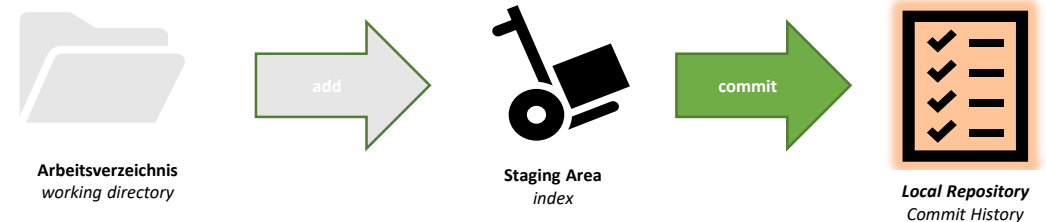
# Historie

`git log`

Zeigt die History des Repositories an

`gitk`

Einfache GUI



# Übung 1 – Barney betritt eine Bar



Szenario	Befehl	Ergebnis
Bar/Repository erstellen	<code>git init bar</code>	
Status prüfen	<code>git status</code>	nothing to commit, working tree clean
Barney betritt die Bar	<code>notepad barney.txt</code> „Hat die Bar betreten“	Neue Datei ‚barney.txt‘
Status prüfen	<code>git status</code>	untracked file: barney.txt
Datei hinzufügen	<code>git add barney.txt</code>	
Status prüfen	<code>git status</code>	new file: barney.txt
Datei einchecken	<code>git commit -m „Barney hat die Bar betreten“</code>	1 file changed, 1 insertion(+)
Status prüfen	<code>git status</code>	nothing to commit, working tree clean



# Übung 2 – Bestellt ein Bier



Szenario	Befehl	Ergebnis
Status prüfen	<code>git status</code>	nothing to commit, working tree clean
Bestellung aufnehmen	<code>notepad bestellungen.txt</code> „Barney möchte 1 Bier“	Neue Datei ‚bestellungen.txt‘
Alle Dateien hinzufügen	<code>git add .</code>	untracked file: ‚bestellungen.txt‘
Datei einchecken	<code>git commit -m „Bierbestellung on Barney“</code>	1 file changed, 1 insertion(+)
Status prüfen	<code>git status</code>	nothing to commit, working tree clean
Historie prüfen	<code>git log</code> <code>gitk</code>	( <code>git log --graph</code> )

# Übung 3 – Barney bekommt Bier

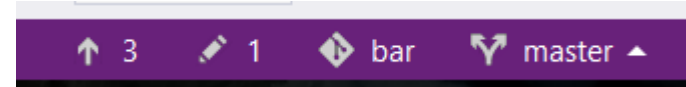


Szenario	Befehl	Ergebnis
Status prüfen	<code>git status</code>	nothing to commit, working tree clean
Bier an Barney liefern	<code>notepad barney.txt</code> „Hat 1 Bier bekommen“ hinzufügen	
Status prüfen	<code>git status</code>	modified: barney.txt
Bestellliste pflegen	<code>notepad bestellungen.txt</code> „ELEDIGT“ vor die Bestellung schreiben	ERLEDIGT 1 Bier für Barney
Status prüfen	<code>git status</code>	modified: barney.txt modified: bestellungen.txt
Änderungen direkt übernehmen	<code>git commit -a -m „1 Bier für Barney“</code>	2 files changed, 2 insertions(+), 1 deletion(-)
Status prüfen	<code>git status</code>	nothing to commit, working tree clean

# Übung 4 – Barney bekommt Schnaps per Visual Studio



- Visual Studio -> Open Folder
- Git Steuerung über Team Explorer
- Ignoredatei für VS comitten
- „1 Schnaps für Barney“ bestellung.txt
- Comitten
- ... Schnaps an Barney liefern



# Dateien Ignorieren

`.gitignore`

Hier wird festgelegt welche Datei ignoriert werden sollen

- Datei im repository Ordner
- [github.com/github/gitignore](https://github.com/github/gitignore) – Vorlagen

# Übung 5 – Lenny und Carl

- Lenny und Carl kommen in die Bar
- Beide trinken ein Bier
- Beide gehen wieder



# Dateien umbenennen oder verschieben

## Per Git Befehl

`git mv <quelle> <ziel>`                      -> Status: renamed

## Per OS Befehl oder Explorer

`move <quelle> <ziel>`                      -> Status: added, remove

## Historie bleibt in beiden fällen erhalten

`git log --follow <dateipfad>`

# Alte Version anschauen

Zurück zu einem Commit gehen

```
git checkout <commitID>
```

- Alle Dateien sind im Zustand des Commits
- Es kann nichts kaputt gemacht werden

Wieder auf den aktuellen Stand gehen

```
git checkout master
```

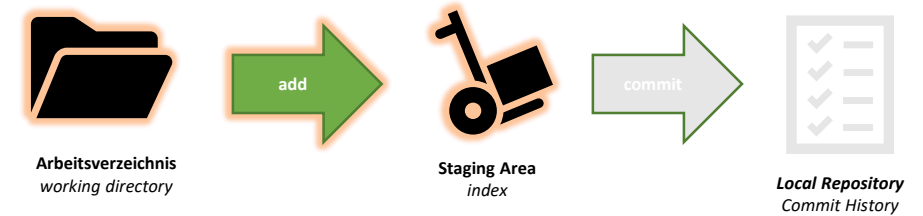
# Staged rückgängig machen

Status:

```
git reset <dateiname>
```

Dateiinhalt:

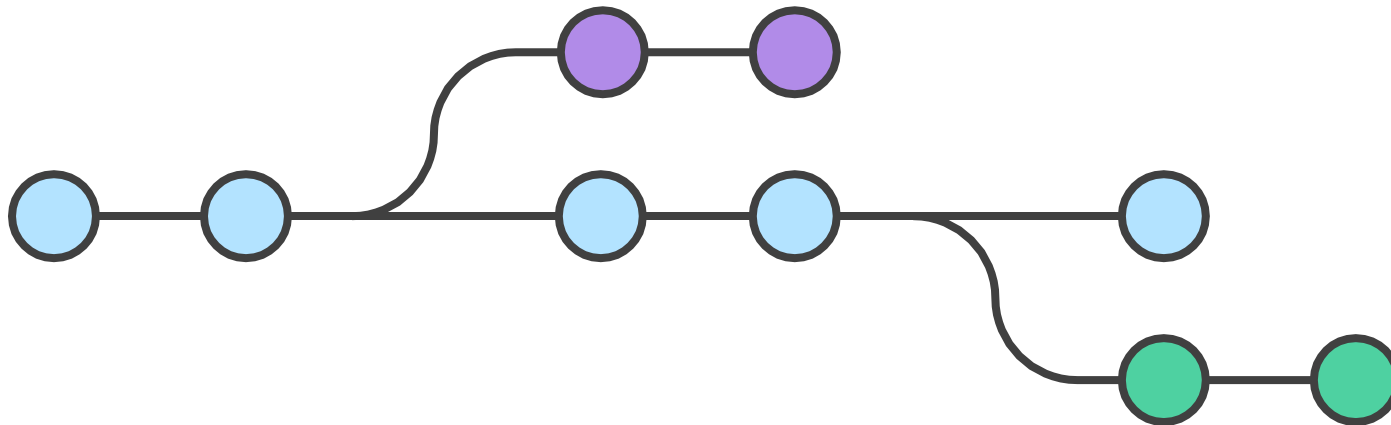
```
git checkout <dateiname>  
(version vom letzten commit)
```





# Branching

- Master ist der Standard Branch
- Isolierte Entwicklungszweig
- Branches sind klein und schnell bei git
- Features werden in eigenen Branches entwickelt



# Branches verwalten

```
git branch <branchName>
```

Erstellt einen neuen Branch

```
git branch --list
```

Zeigt alle Branches an

```
git checkout <branchName>
```

Wechselt zu dem angegebenen Branch

```
git branch -d <branchName>
```

Löscht zu dem angegebenen Branch

# Branches zusammenführen

1. Wechseln zum ,empfangenden Branch'  
`git checkout <branchName>`

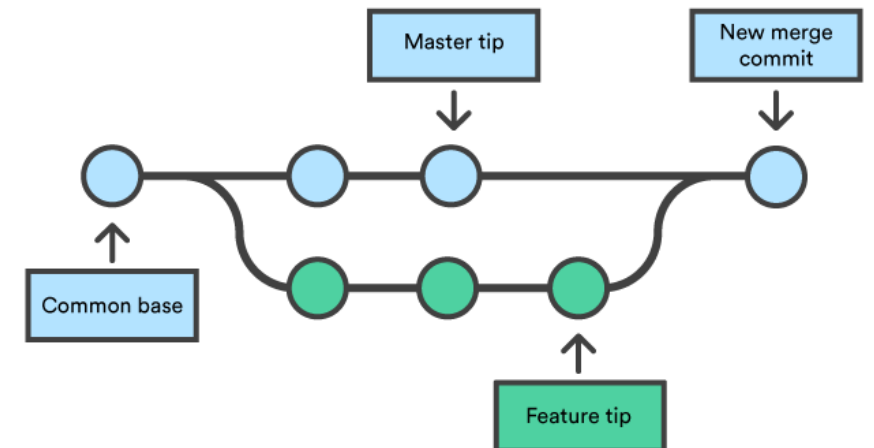
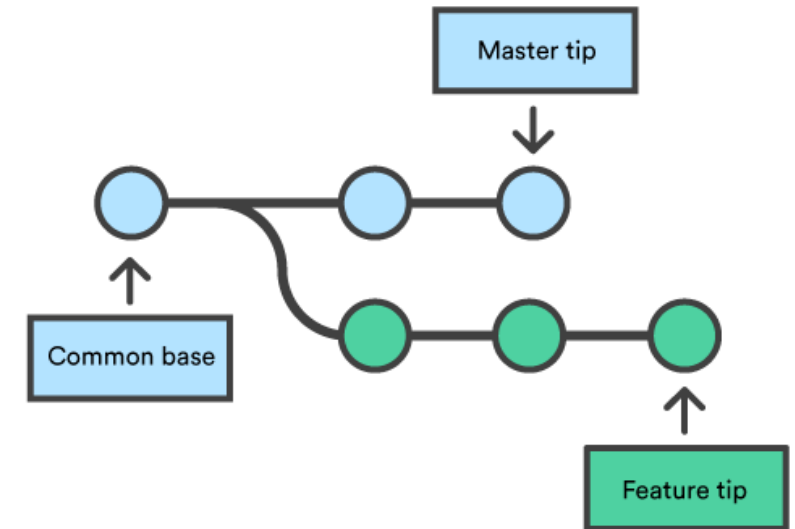
2. Die neuste Version von Remote laden

`git fetch`

`git pull`

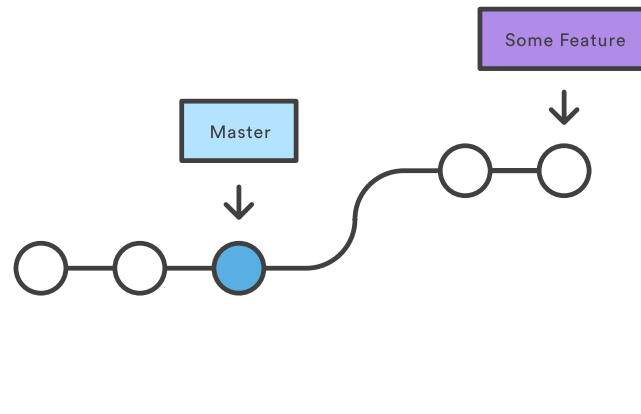
3. Branches zusammenführen

`git merge <brachName>`

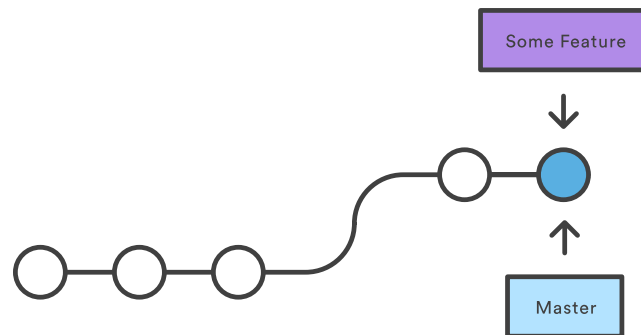


# Fast Forward Merge

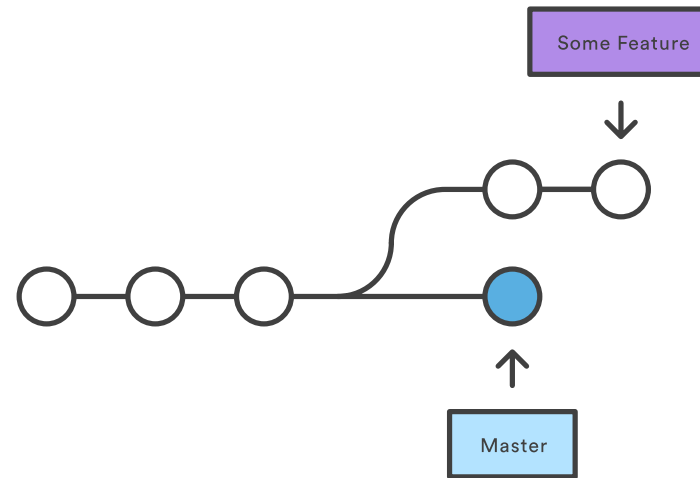
Before Merging



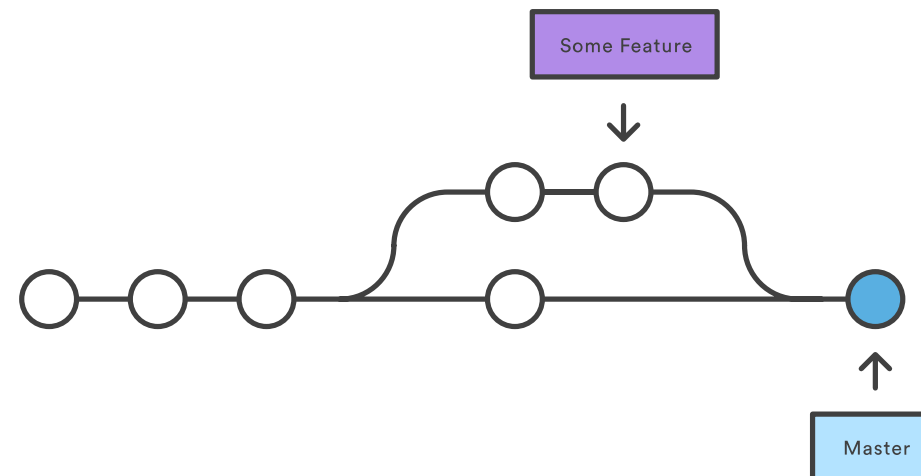
After a Fast-Forward Merge



# 3-Way-Merge



After a 3-way Merge



# Branches Zusammenführungskonflikte

`git status`

Zeig Dateien die im Konflikt sind

`git mergetool`

Tool und Dateien zu mergen

Besser VS benutzen

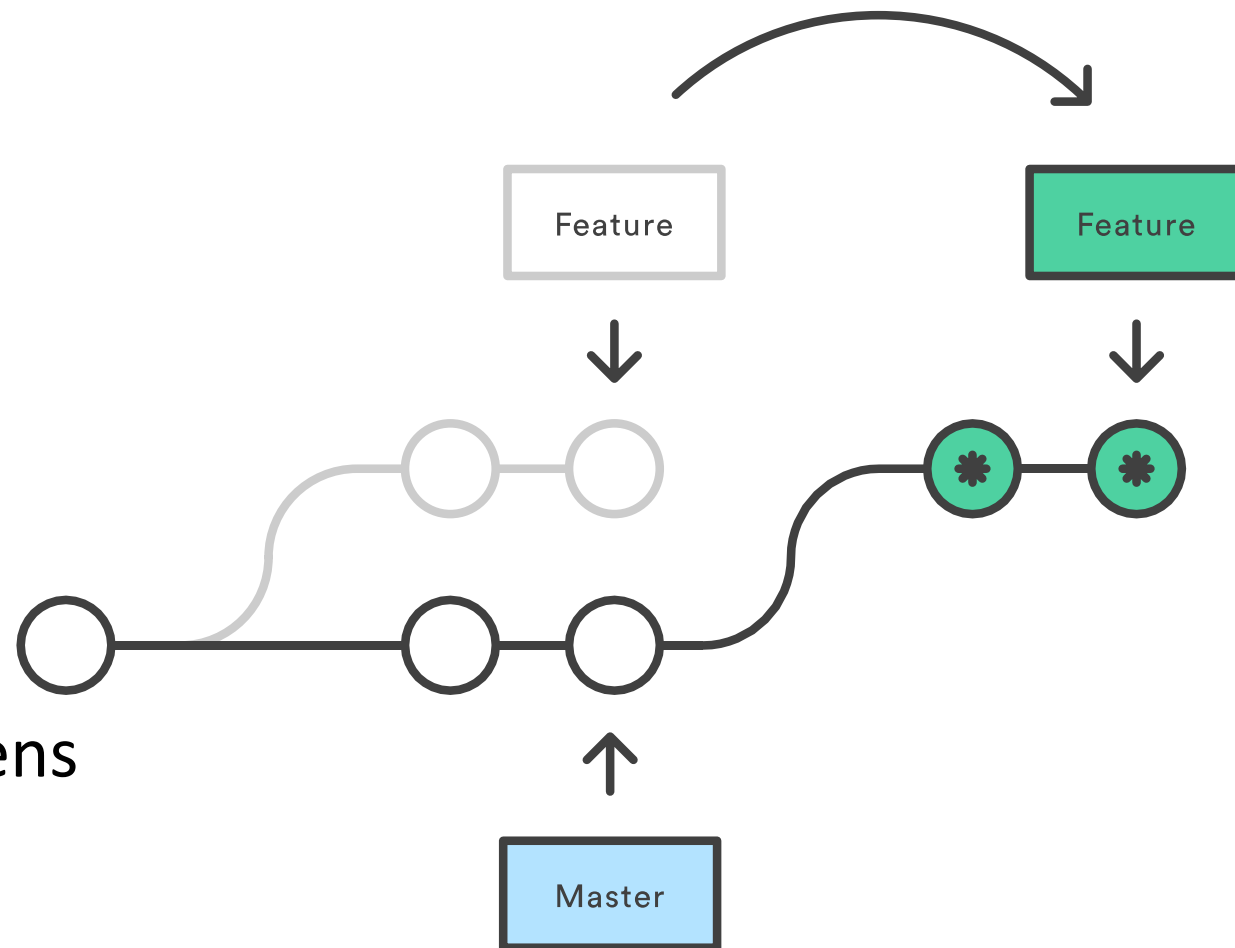
`git commit`

Finalisiert die merge Vorgang

# Git rebase

git rebase

- Nur bei „eigenen“ Branchens



```
$ git checkout experiment
```

```
$ git rebase master
```

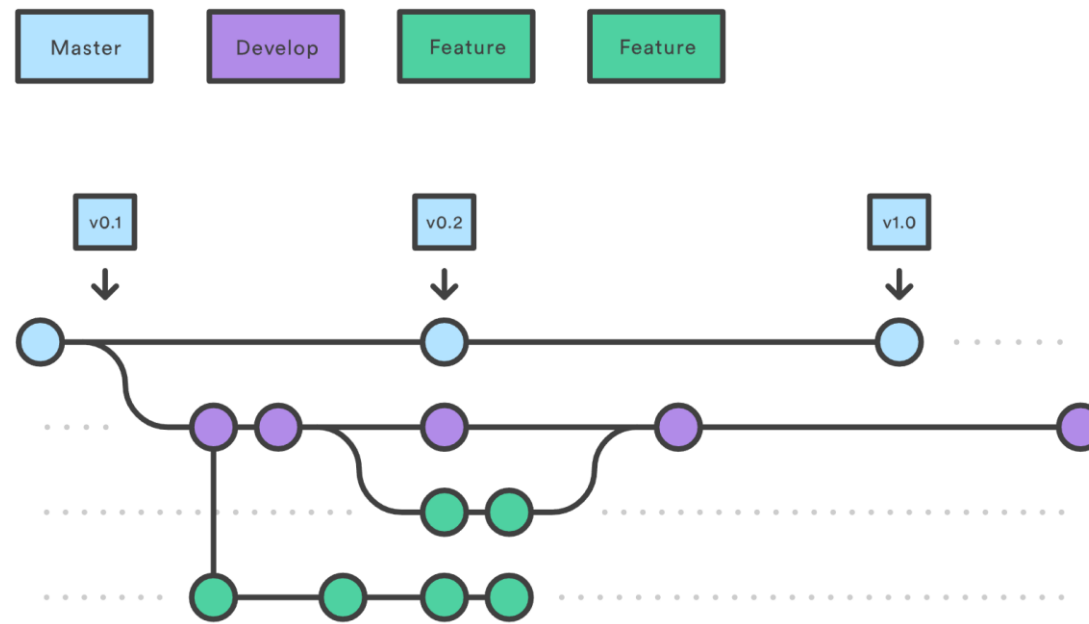
First, rewinding head to replay your work on top of it...

Applying: added staged command

\* Brand New Commits

# Workflows

- Zentralisiert
  - Nur Master
- Feature Branches
  - Featuresbranches in Master
- Gitflow





# Letzen Commit korrigieren

```
git commit --amend
```

- Korrektur der letzten Commit Message
- Datei vergessen

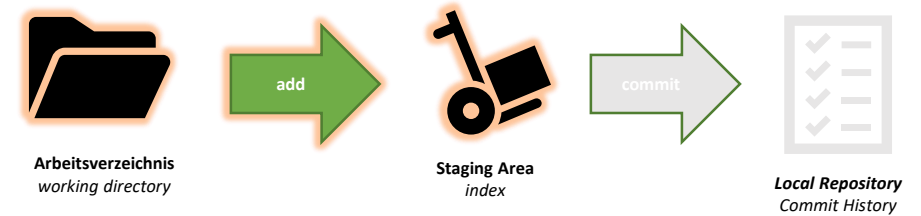
# Staged rückgängig machen

Status:

```
git reset <dateiname>
```

Dateiinhalt:

```
git checkout <dateiname>  
(version vom letzten commit)
```

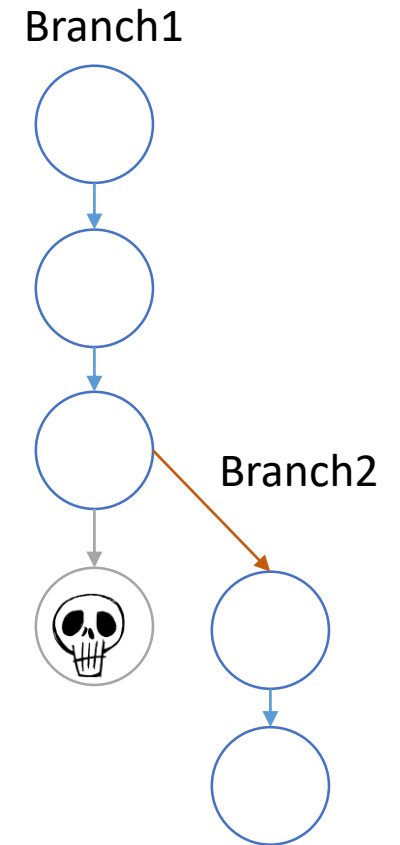


# Commits rückgängig machen

- Mehrere Wege
  - Checkout
  - Revert
  - Reset
  - Amend

# Checkout um änderungen Rückgängig zu machen

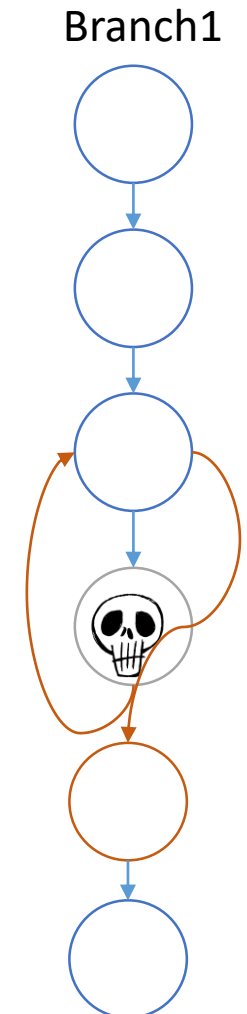
- Checkout auf einen alten Commit machen
- Daraus einen Branch erstellen:
  - `git checkout -b new_branch_without_crazy_commit`
- Im neuen Branch weiter arbeiten
- Alter Branch ist 'verloren'
  - Nicht machbar bei master etc.



# Revert um änderungen Rückgängig zu machen

```
git revert <commitID>
```

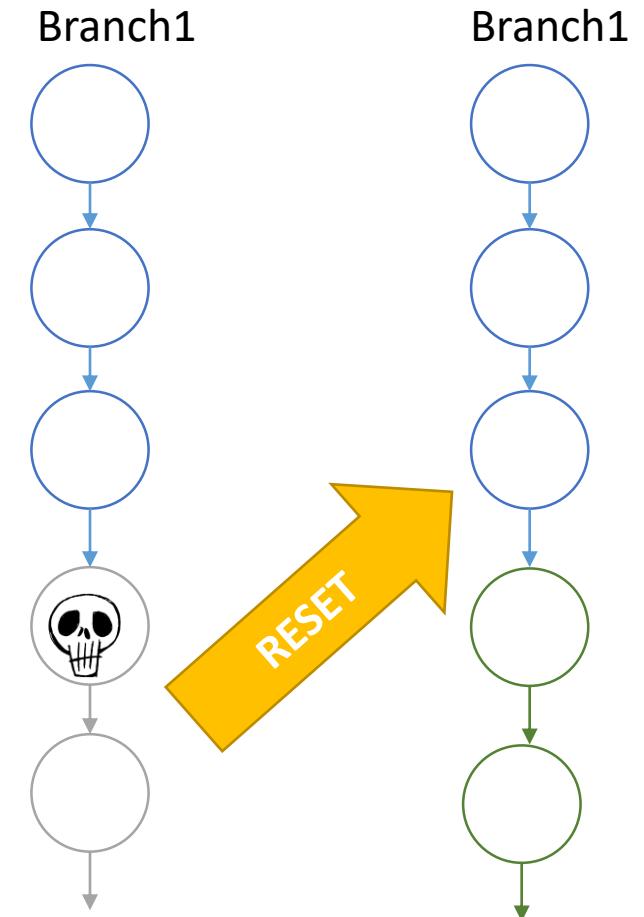
- Aktion erzeugt einen Commit
- Fehlerhafter commit mit Daten bleibt in Historie



# Reset um Änderungen rückgängig zu machen

```
git reset --hard <commitID>
```

- Historie wird geändert
- Problematisch bei remote repositories



# Stash – Branch wechsel

**Änderungen ohne Commit speichern**

```
git stash push
```

**Gespeicherte Änderungen anzeigen**

```
git stash list
```

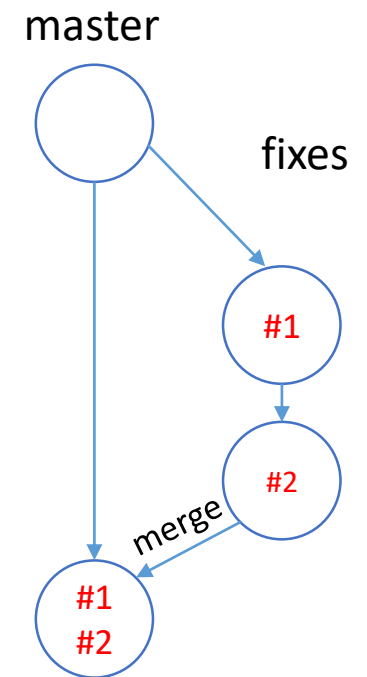
**Letzte gespeicherte Änderung laden**

```
git stash pop
```

# Squash mit merge

Mehrere commits aus als 1 commit mergen

```
git checkout master  
git merge --squash fixes  
git commit
```



- Fixes Branch ist unverändert und sollte gelöscht werden

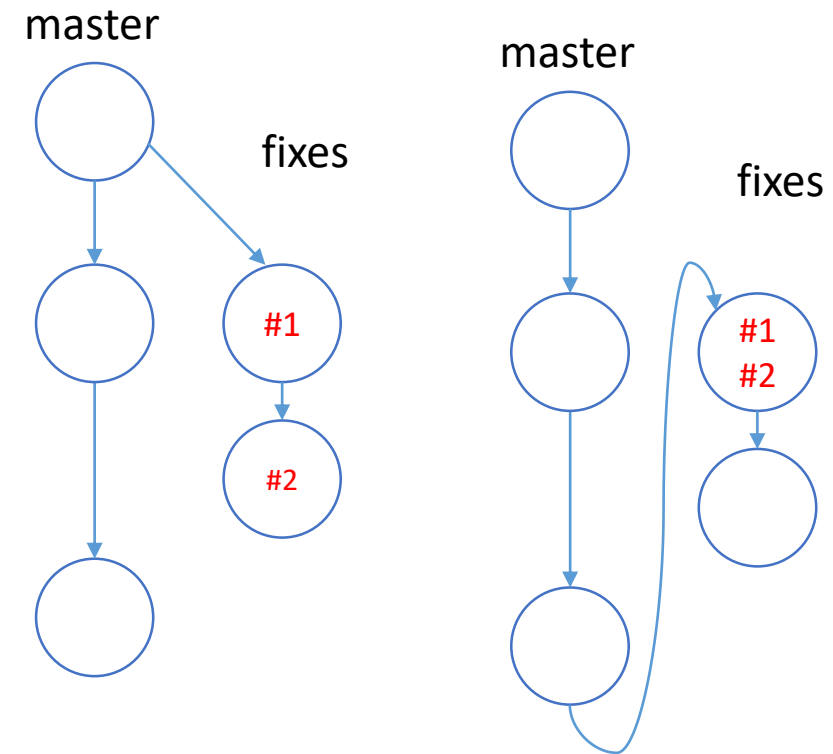


# Squash mit rebase

Mehrere commits aus als 1 commit mergen

```
git checkout fixes  
git rebase -i master  
...vim...
```

- Kein extra commit
- In *fixes* kann weiter gearbeitet werden
- Kann auch auf eigenem Branch gemacht werden



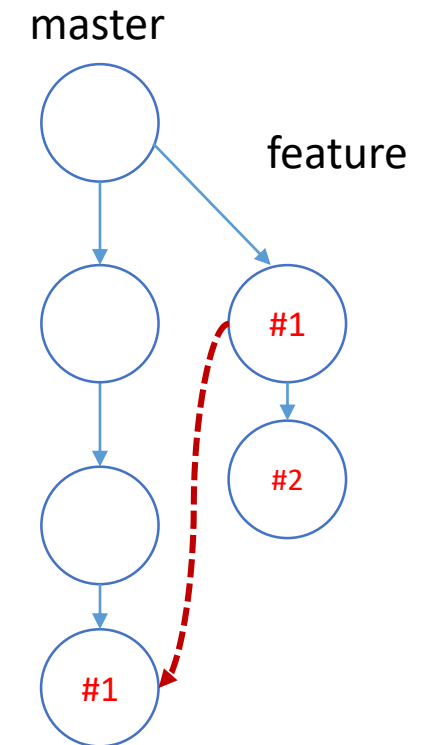
# cherry-pick

Einen commit kopieren

```
git checkout master
```

```
git cherry-pick -x <commit-hash>
```

- -x für eine ordentlich commit message



# Submodules

Fremd-Repository als Bestandteil des eigenen Repositories

- Zeigt auf einen Commit im SubRepository

```
git submodule add <repo URI> <lokaler pfad>  
git submodule add ../anyRepo/ subRepo/  
git commit -a -m "sub hinzugefügt"
```

```
cd subRepo/  
git fetch  
git pull
```

