

GRUNDLAGEN DER VERSIONSKONTROLLE

MIT GIT

INHALTE

1. Ziele

2. Datensicherung

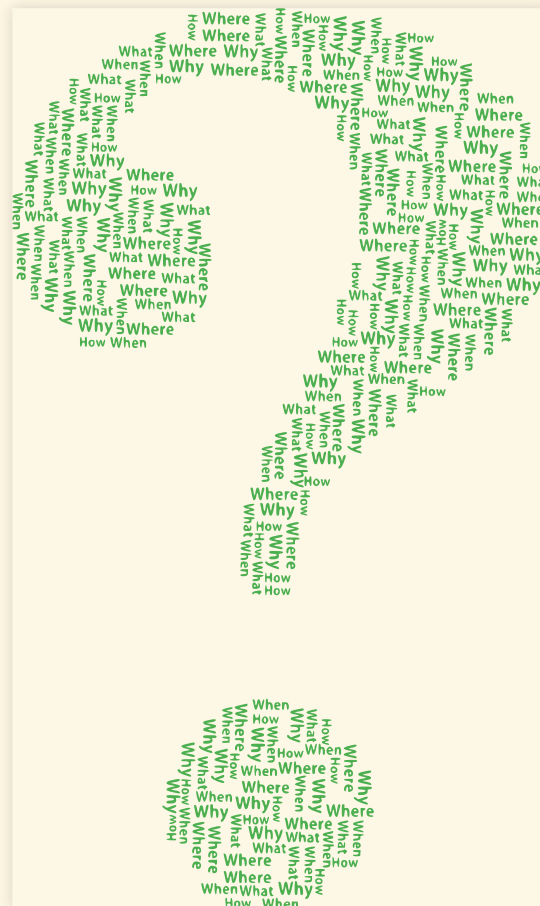
3. Git

- Web UI
- Grundlegende Befehle
- Lifecycle
- Branching
- Paralleles Arbeiten

AM ENDE DIESES MODULS ...

- ... kennen Sie die Vorteile von Versionskontrollsystemen
- ... haben Sie Git eingerichtet
- ... können Sie Git Projekte anlegen
- ... können Sie Konflikte auflösen

WARUM?

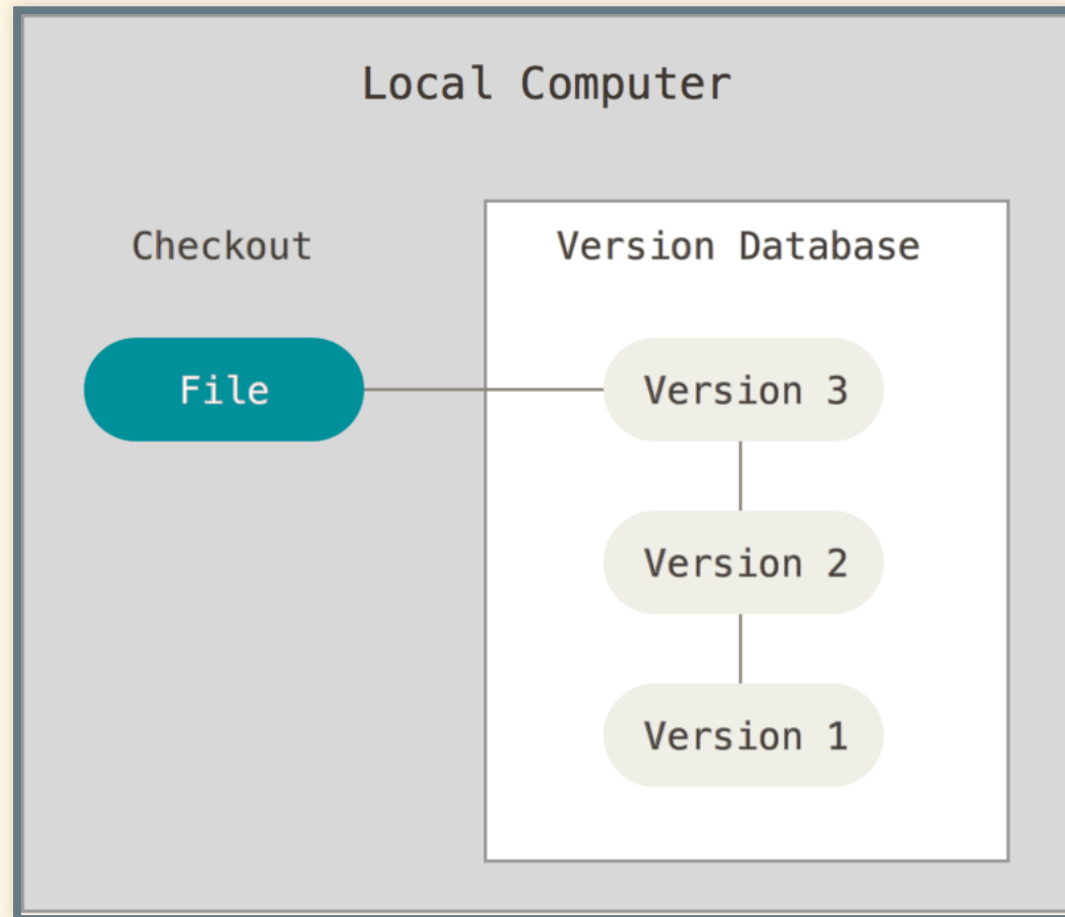


ZIELE DER VERSIONSKONTROLLE

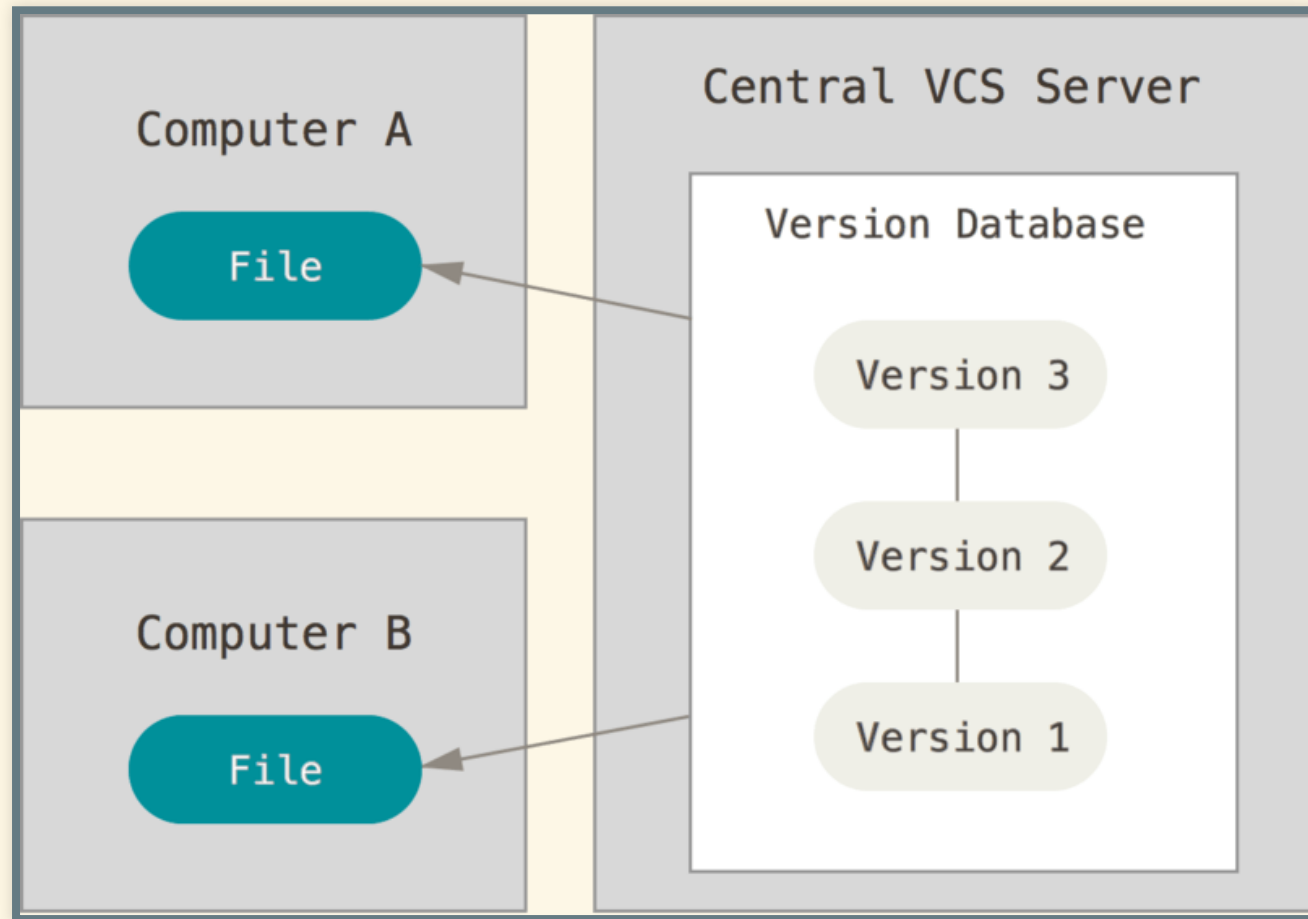
1. Eindeutigkeit bzgl. der aktuellen Version
2. Austausch von Daten
3. Datensicherung
4. Nachvollziehbarkeit
5. Wiederherstellung eines Softwarestandes

VERTEILTE VERSIONSKONTROLLE

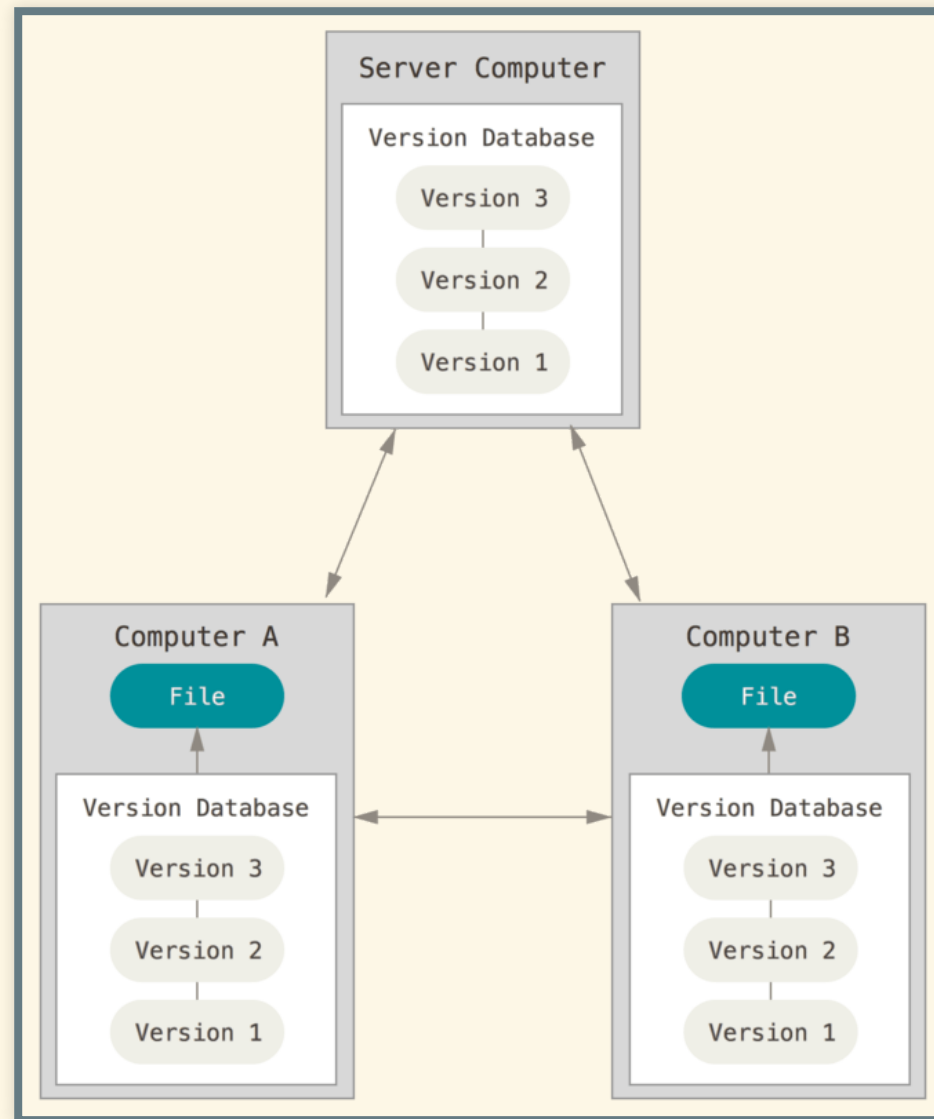
LOKALE VERSIONSKONTROLLE



ZENTRALE VERSIONSKONTROLLE



VERTEILTE VERSIONSKONTROLLE

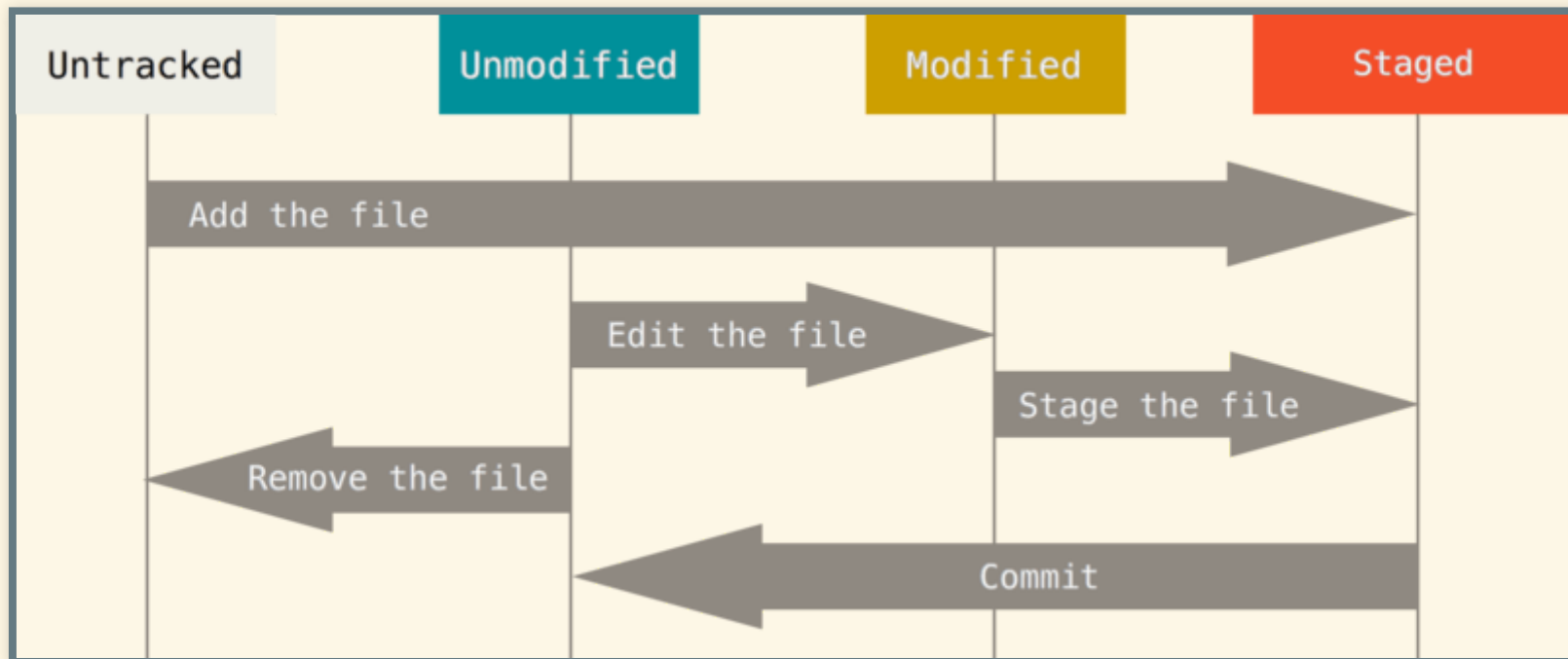


GIT

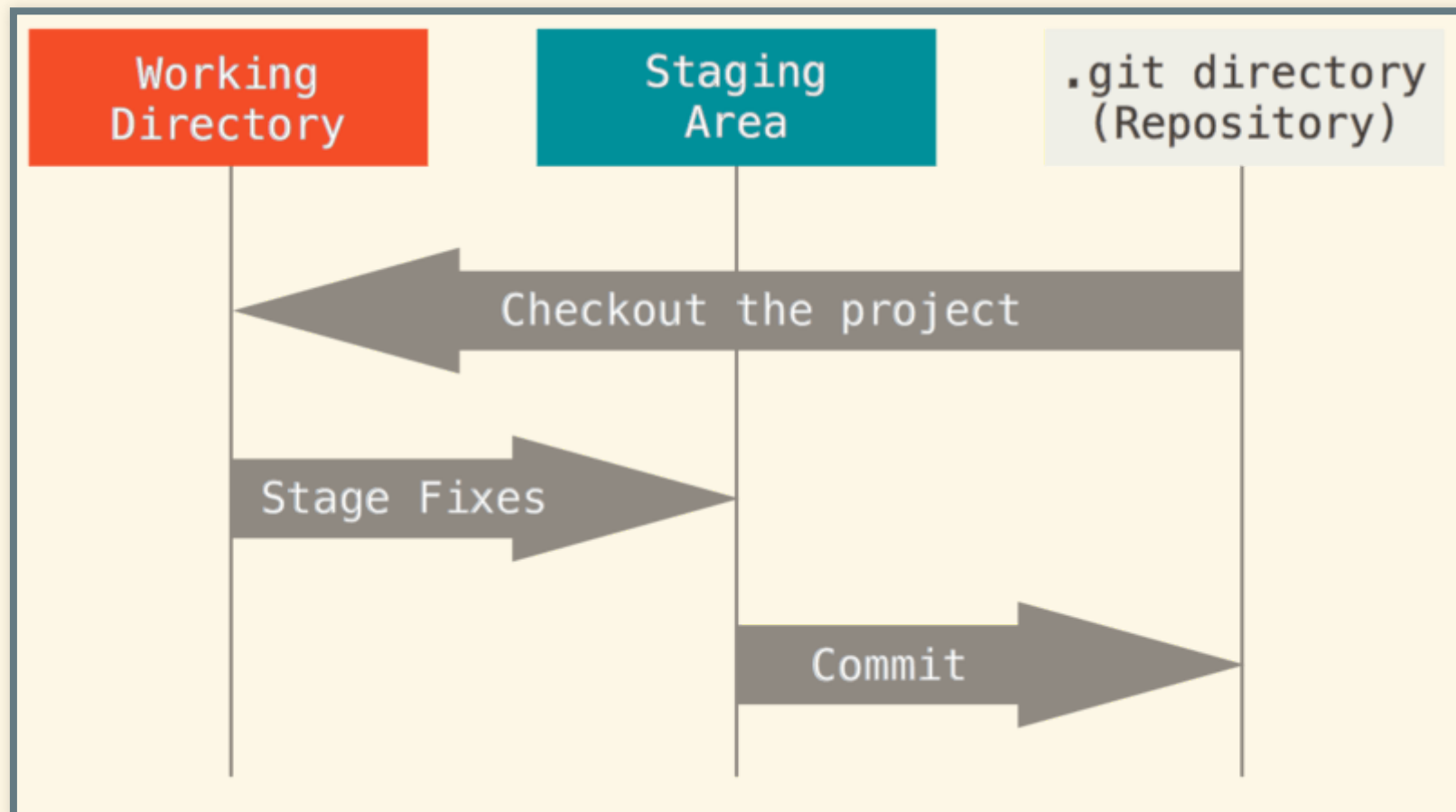
WEB UI

| Produkt | Hersteller | Features |
|---------|------------------------------|--|
| Gerit | Google | Review System, Changesets, Outdated |
| GitHub | Microsoft | Marktführer, Issues, Wiki, Editor, Pull Requests |
| GitLab | Open-Source / GitLab Inc. | CI, Container Registry, Issues, WebIDE, Wiki, Merge Requests |

LIFECYCLE EINER DATEI



GIT ARBEITSBEREICHE



GIT BEFEHLE

```
git --help
```

`--help` am Ende eines Befehls zeigt die Hilfe an. Unter anderem werden die möglichen Parameter aufgelistet und erläutert.

git config

Ermöglicht die Konfiguration von Git, dabei gibt es:

- Lokale Einstellungen (je Repository)
- Globale Einstellungen

```
git config --global user.name "Mona Lisa"  
git config --global user.email "mona.lisa@example.com"
```



```
git init
```

Initialisiert ein neues lokales Repository im angegebenen Ordner.

git status

Zeigt den Status des lokalen Repositories an:

- Aktueller Branch
- Status veränderter Dateien
- Inhalt des Staging Bereichs

```
On branch master
No commits yet
Changes to be committed:
(use "git rm --cached <file>..." to unstage)
    new file:   README.md
```

git add

Fügt eine Datei dem Staging Bereich des Versionskontrollsystems hinzu.

```
git add Readme.md
```

git commit

Sichert den aktuellen Stand des Staging Bereichs ins Repository. Dabei wird für diesen eine eindeutige ID generiert. Der Staging Bereich wird geleert.

```
git commit -m "My first commit"  
[master (root-commit) 562fbb8] My first commit  
1 file changed, 2 insertions(+)  
create mode 100644 Readme.md
```

git remote

Ermöglicht den Umgang mit Remote Repositories, indem es einen Alias hinzufügt. Dadurch kann leicht auf diesen verwiesen werden.

git remote add <alias> <URL>

```
git remote add origin https://github.com/<Nutzer>/myfirstrepo.git
```

git push

Aktualisiert das angegebene Remote Repository, sodass dort die Referenzen ebenfalls vorhanden sind.

```
git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 279 bytes | 139.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/<Nutzer>/myfirstrepo/pull/new/master
remote:
To github.com:<Nutzer>/myfirstrepo.git
 * [new branch]      master -> master
```

BASICS: SHELL COMMANDS

- *mkdir <folder>*

Legt einen neuen Ordner an.

- *cd <folder>*

Navigiert in das angegeben Verzeichnis.

Sonderfall: "*cd ..*" verlässt den Unterordner.

- *ls -al*

Listet alle Dateien und Ordner im aktuellen Verzeichnis.

- *cat <file>*

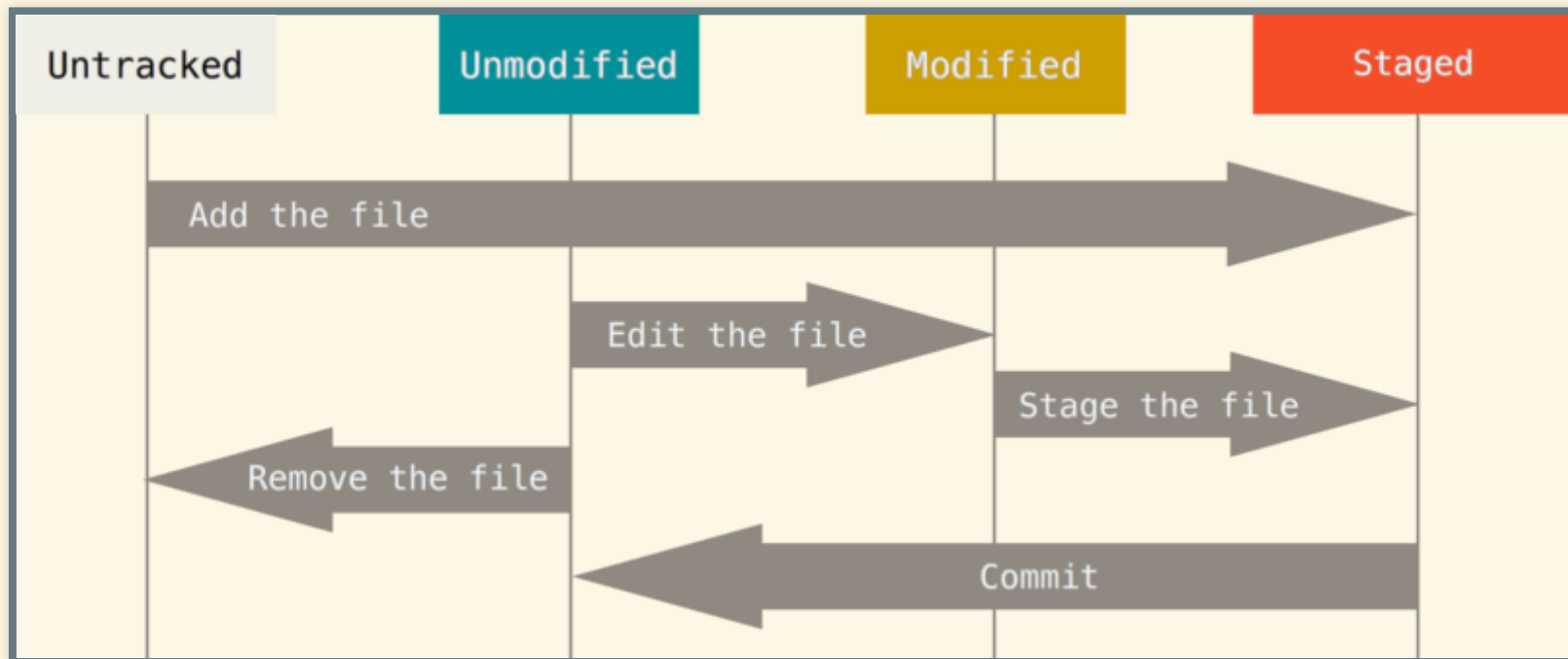
Gibt den Inhalt der Datei auf die Konsole aus.

BASICS: VIM

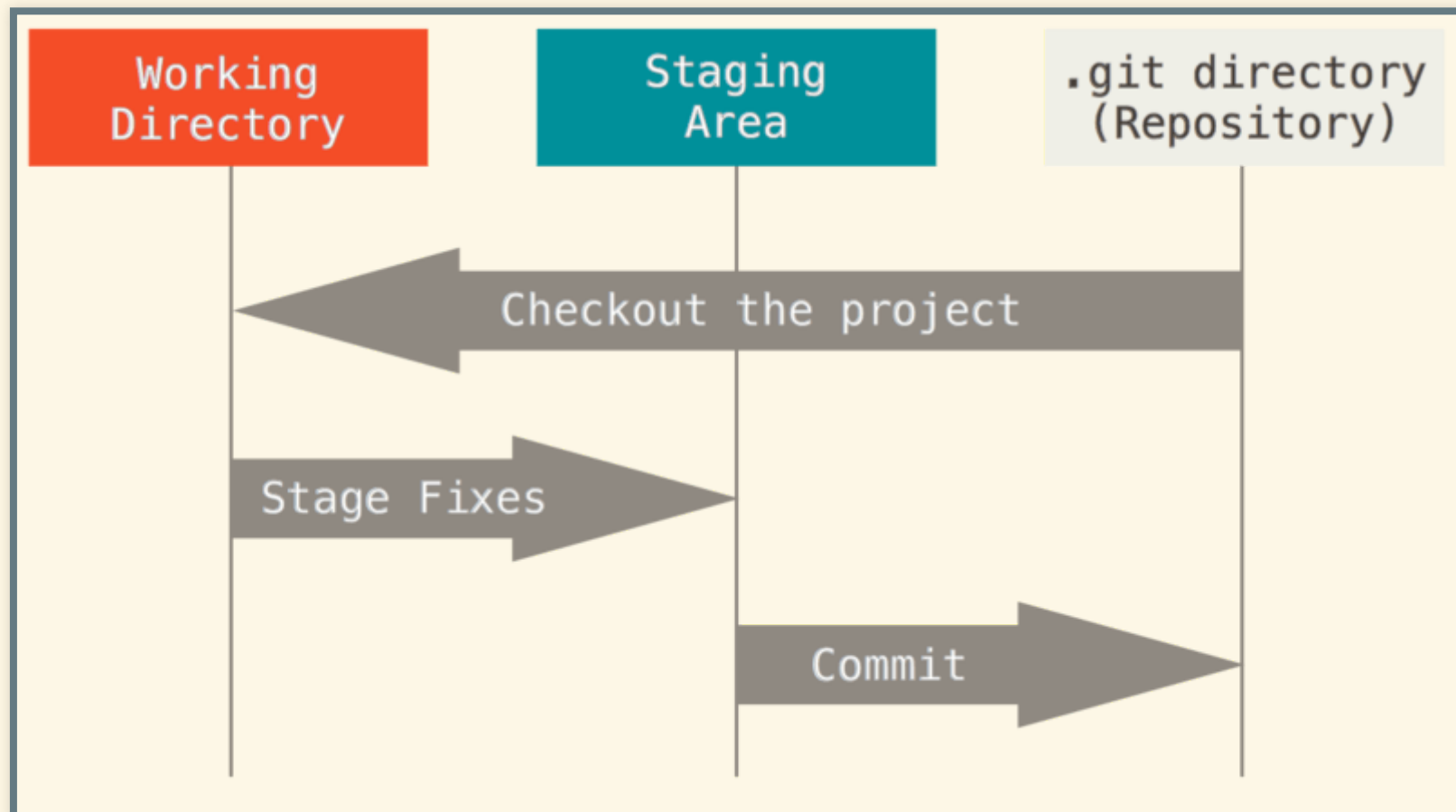
- `i` für interactive mode
- `esc` für normal mode
- `:wq!` beenden *mit* speichern
- `:q!` beenden *ohne* speichern

ÜBUNG 1: GIT SETUP

WIEDERHOLUNG: LIFECYCLE EINER DATEI



WIEDERHOLUNG: GIT ARBEITSBEREICHE



git clone

Erstellt eine lokale Kopie eines bestehenden Repositories in einem neuen Verzeichnis.

```
git clone https://github.com/<Nutzer>/myfirstrepo.git/  
Cloning into 'myfirstrepo'...  
remote: Counting objects: 3, done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), done.
```

BRANCHING

In Git gibt es Branches, die auf Commits bzw. Snapshots verweisen. Ein Branch verweist auf einen Commit. Allerdings können auch mehrere Branches auf den gleichen Commit verweisen. Der momentan aktive Branch wird durch den HEAD bestimmt.

git checkout

Ermöglicht es, einen Branch auszuwechseln.

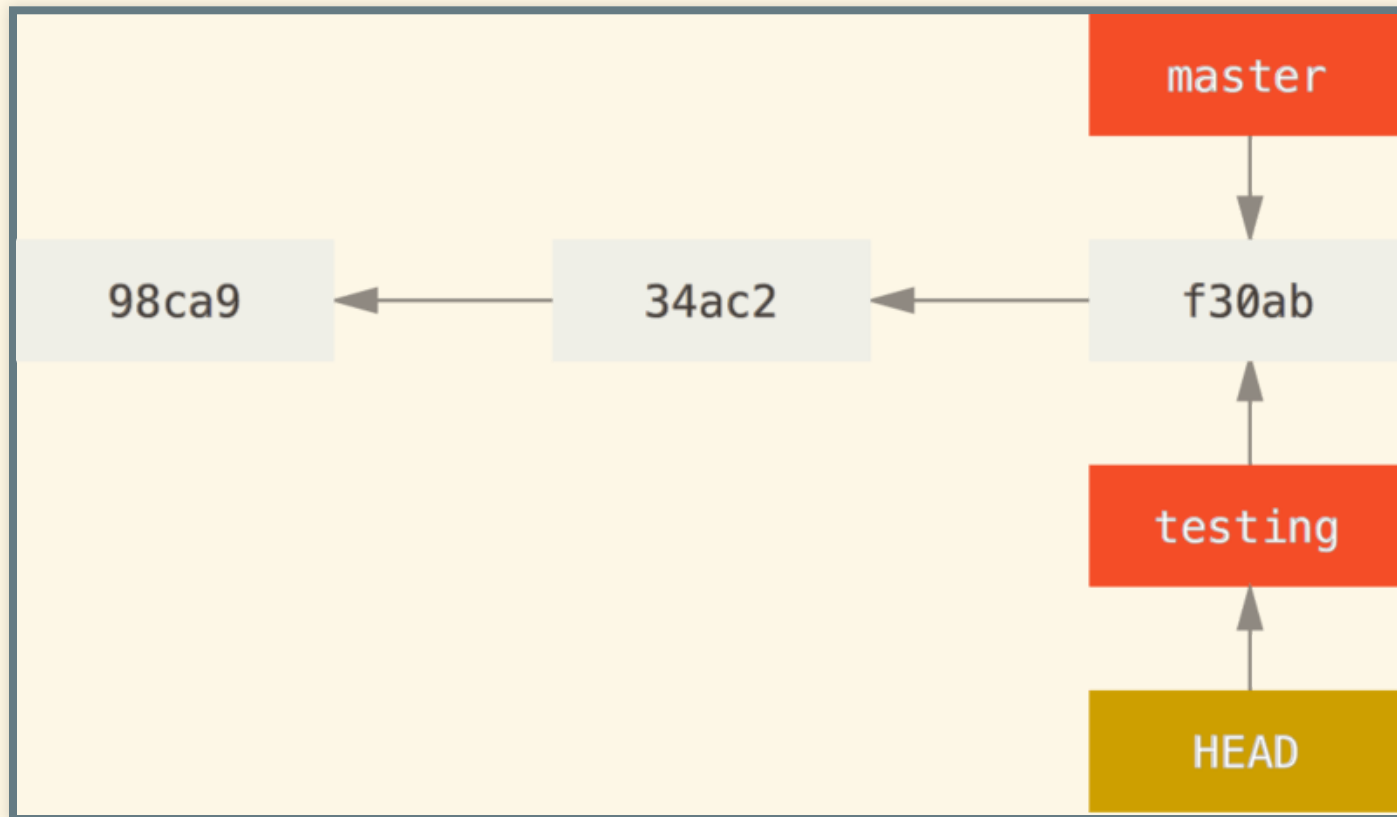
```
git checkout master  
Switched to branch 'master'  
Your branch is up to date with 'origin/master'.
```

Zudem können auch neue Branches erstellt werden z.b. mit

```
git checkout -b testing  
Switched to a new branch 'testing'
```

BRANCH TESTING IST AKTIV

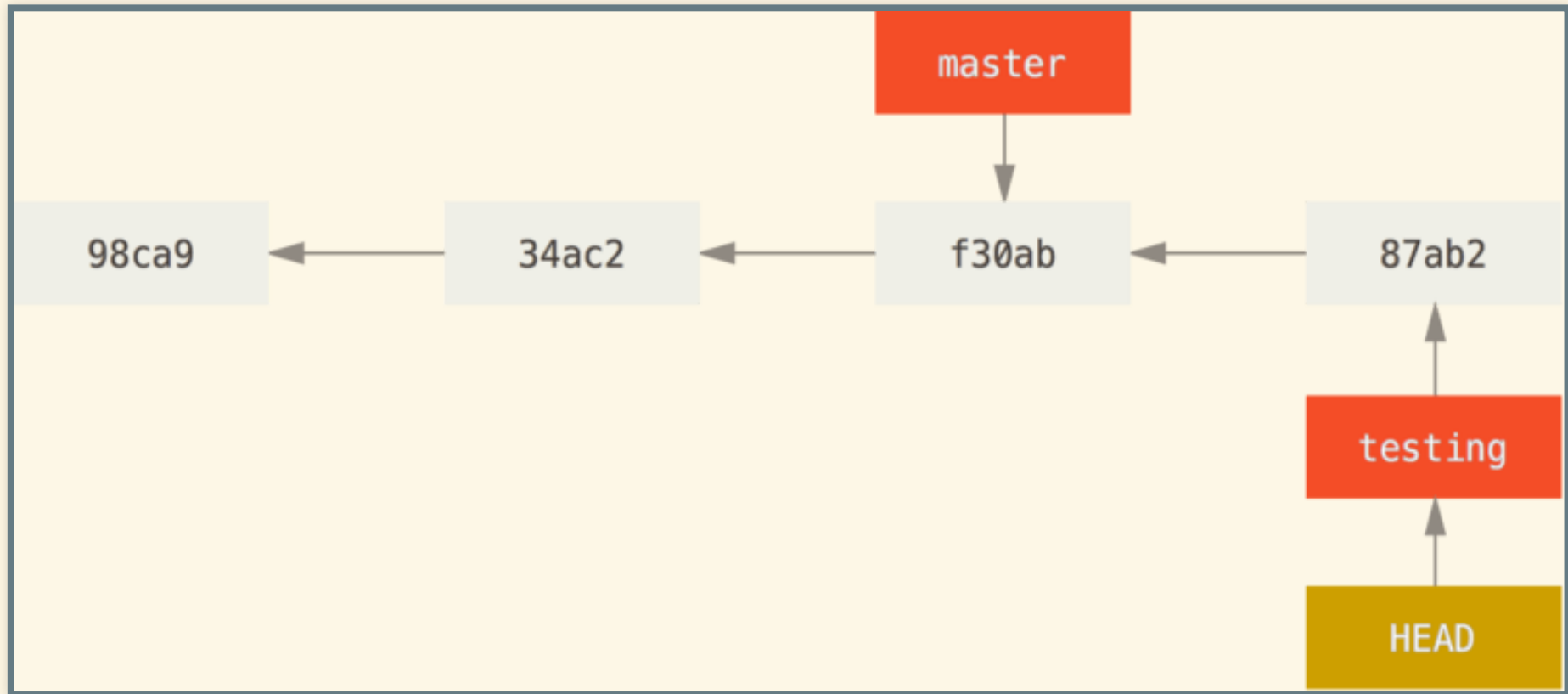
```
git checkout -b testing
```



Wie würde ein Commit im Branch `testing` aussehen?

COMMIT IN BRANCH TESTING

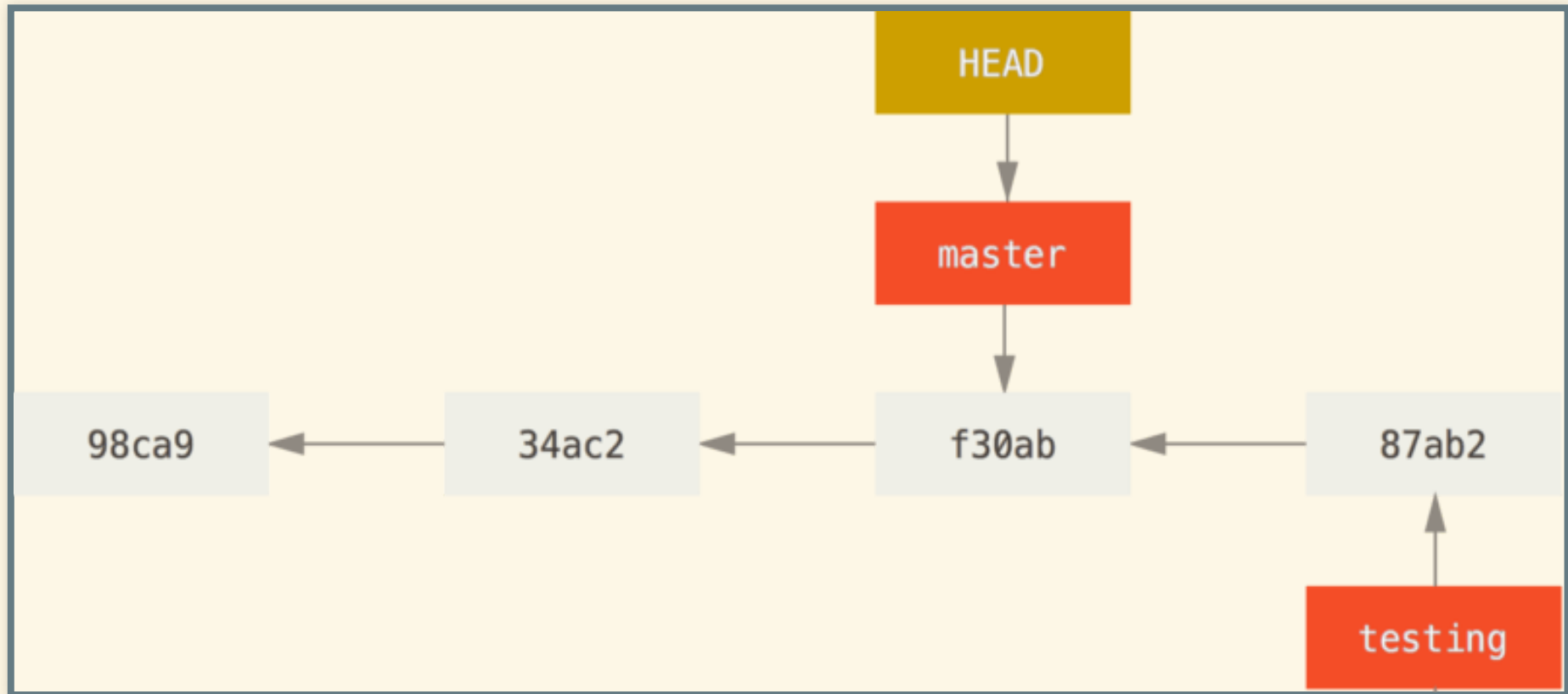
```
vim neuedatei.txt  
git add neuedatei.txt  
git commit -m 'Added neuedatei.txt'
```



Was würde bei einem Checkout des Branches `master` passieren?

CHECKOUT MASTER

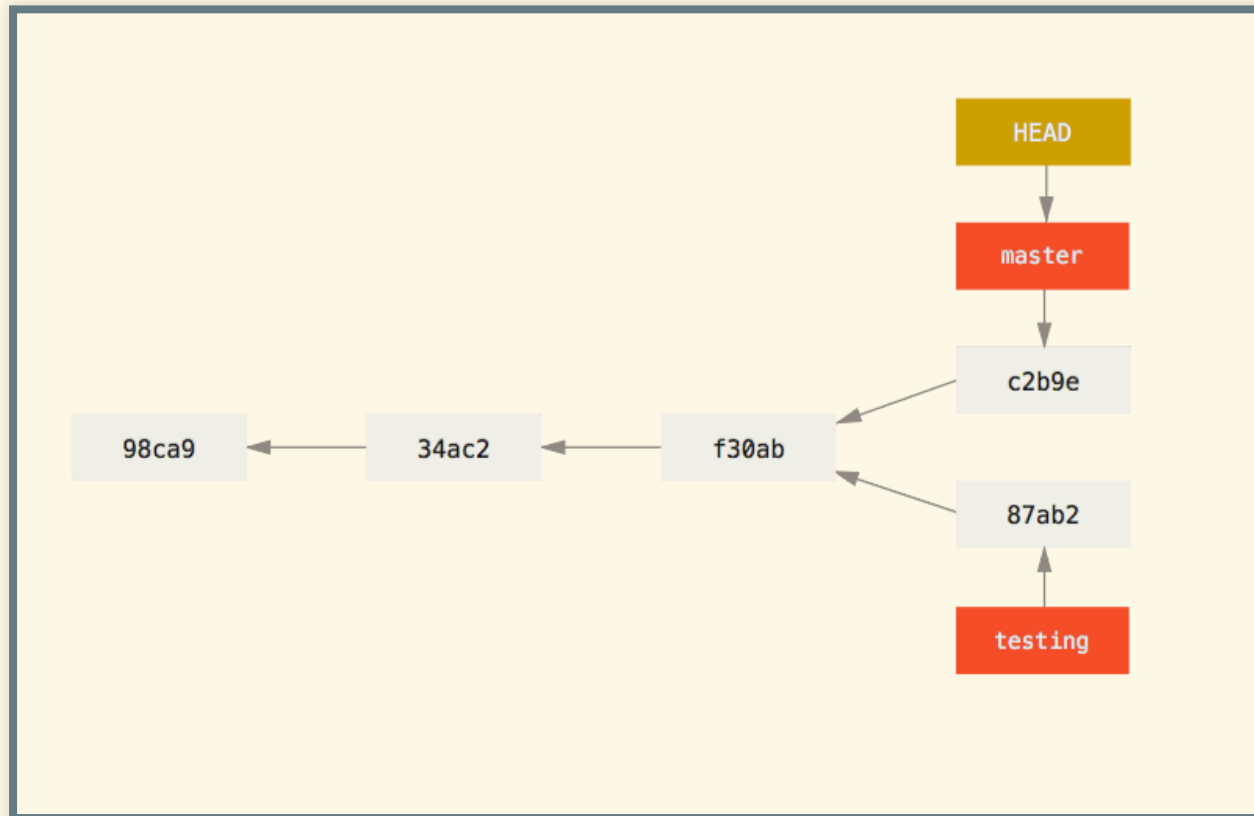
```
git checkout master
```



Wie würde ein Commit im Branch `master` aussehen?

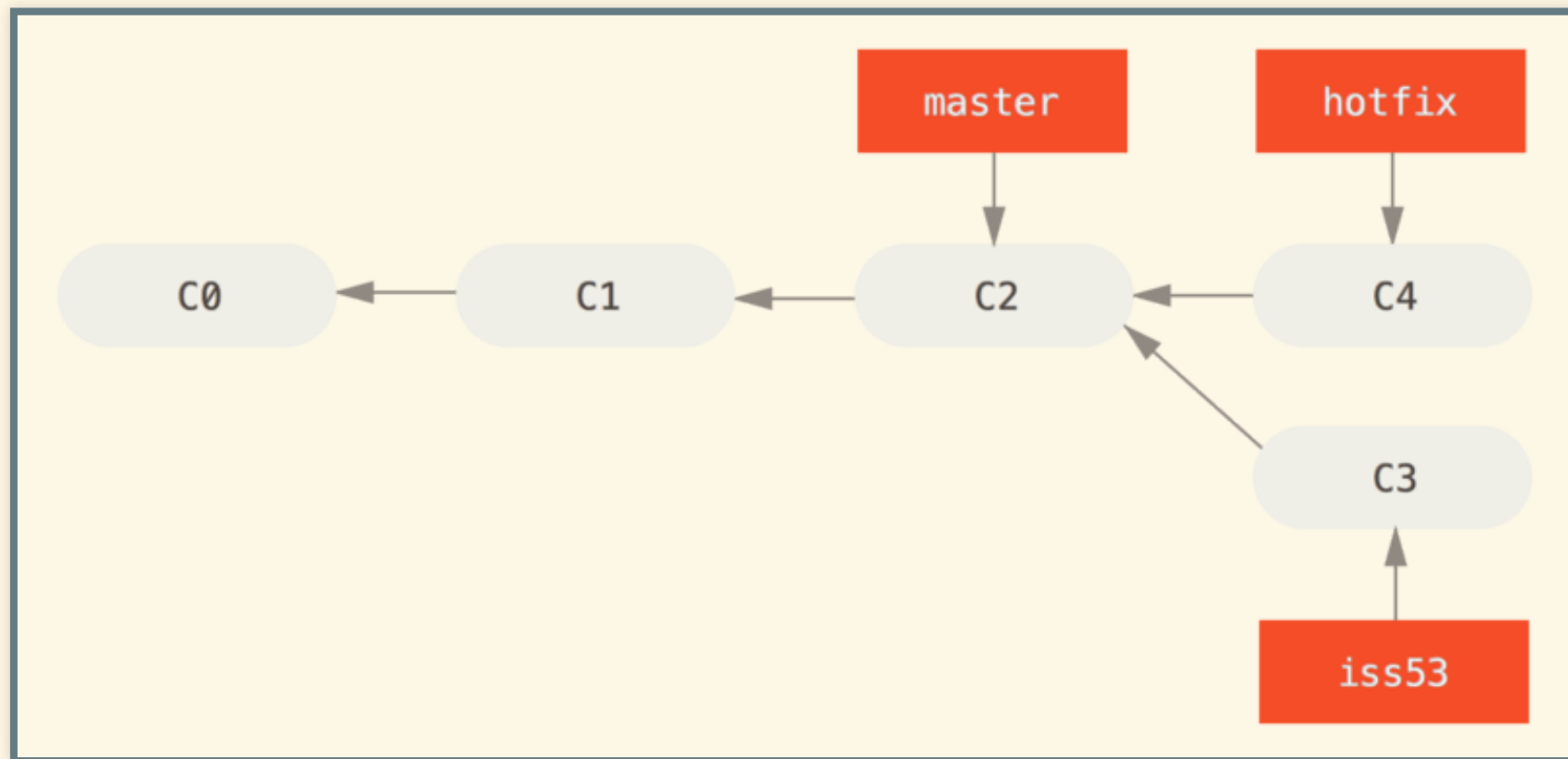
COMMIT MASTER

```
vim readme.md  
git commit -m 'Fix some minor issues'
```



ÄNDERUNGEN ZUSAMMENFÜHREN

Um Änderungen in Git zusammenzuführen gibt es die beiden Befehle `git rebase` und `git merge`.



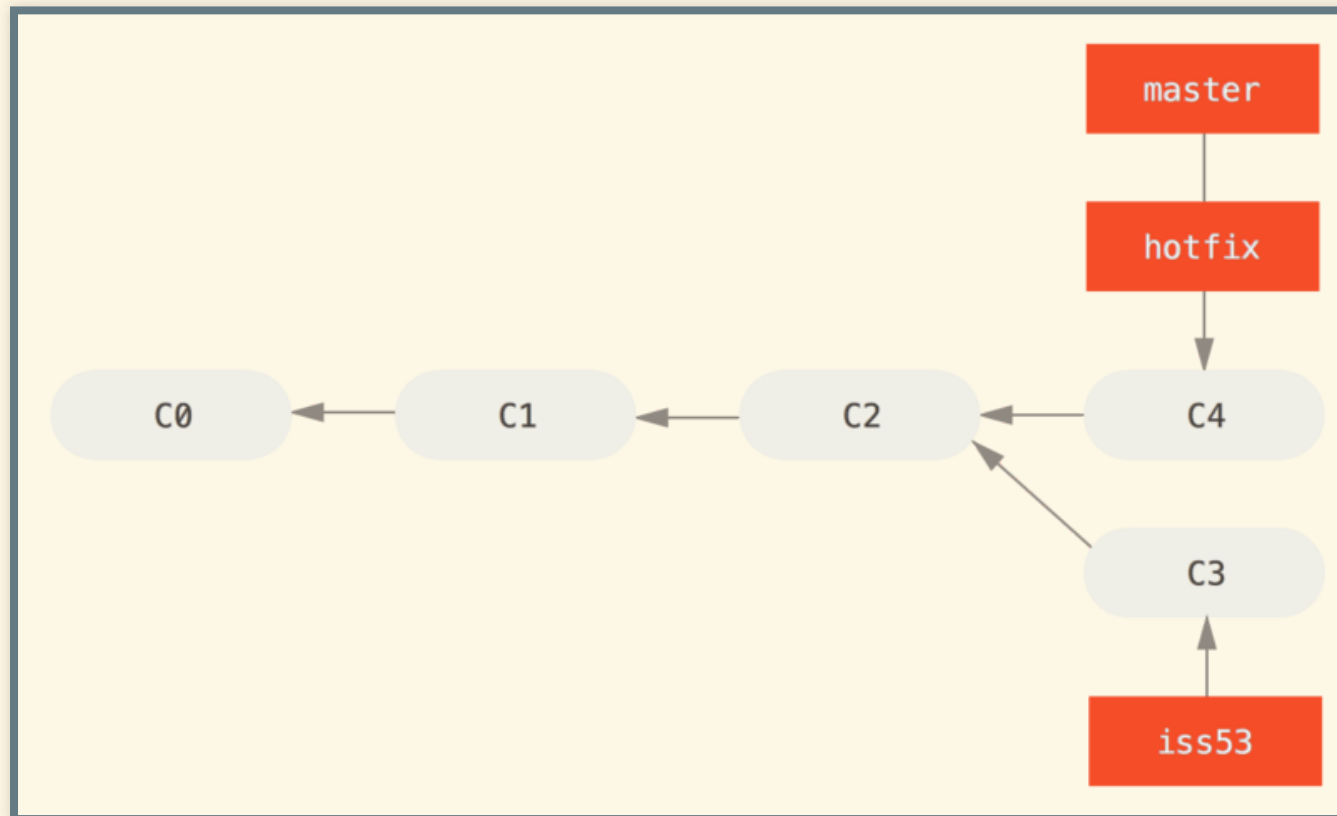
git merge

Der Befehl fügt mehrere Historien zusammen. So können z.B. Branches zusammengeführt werden. Je nach Situation verhält sich der Befehl unterschiedlich:

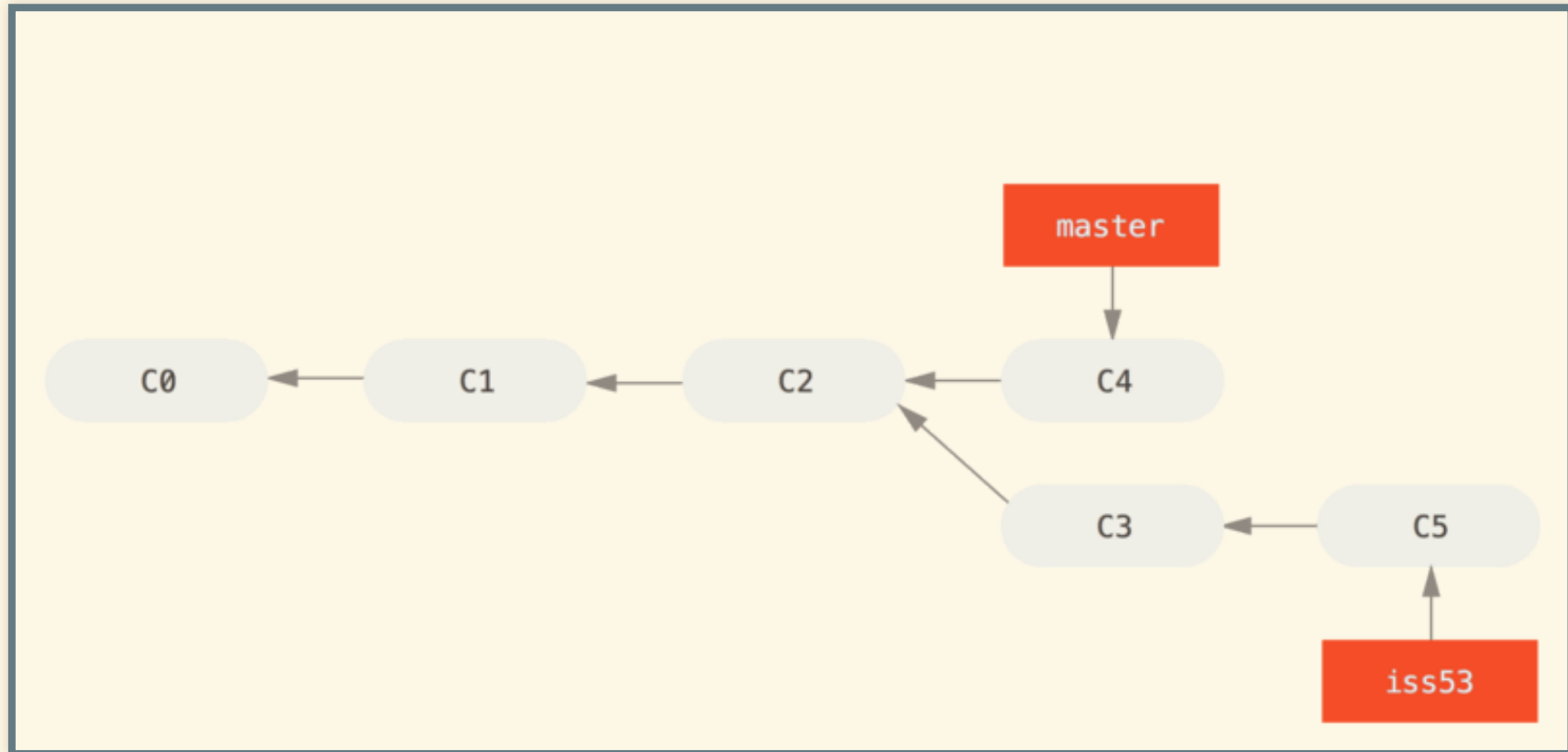
- Eine der Historien hat keine Änderung: Fast Forward
- Beide Historien haben Änderungen: Merge Commit
 - In unterschiedlichen Dateien: Automatisch
 - In gleicher Datei: Manuell

git merge - FAST-FORWARD

```
git merge hotfix  
Updating f42c575..3a0874d  
Fast-forward
```



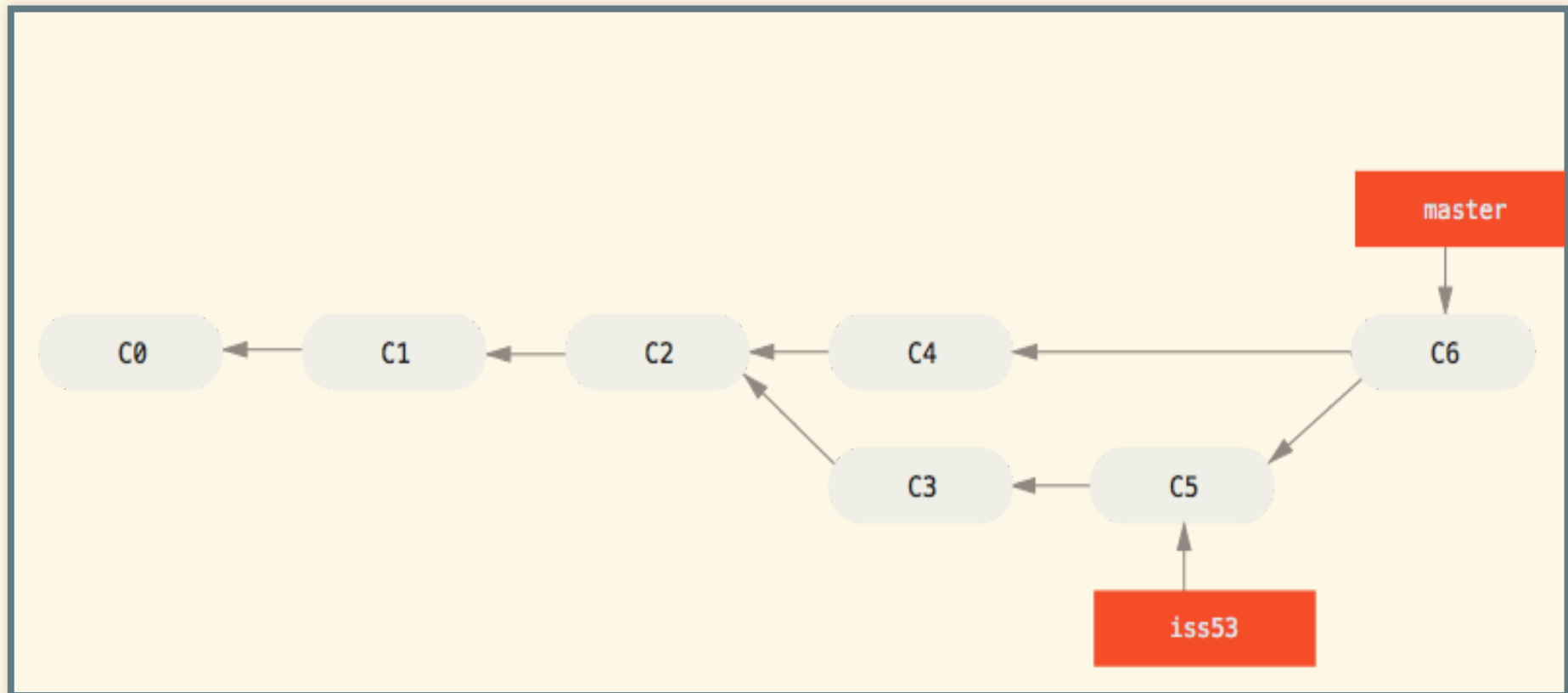
git merge



In dieser Situation muss der Merge einen Commit durchführen, der die Änderungen zusammenführt.

git merge

```
git checkout master  
git fetch  
git merge iss53
```



`git merge` - **KONFLIKT**

Falls Git nicht in der Lage ist die Änderungen selbständig zusammenzuführen erhalten Sie einen Konflikt. Die Meldung sieht dann wie folgt aus:

```
git merge
Auto-merging helloWorld.bat
CONFLICT (content): Merge conflict in helloWorld.bat
Automatic merge failed; fix conflicts and then commit the result.
```

Diesen Konflikt müssen Sie selbst auflösen.

git merge - KONFLIKT

Git hat Ihnen in den betreffenden Dateien Kommentare hinzugefügt. Diese sehen wie folgt aus:

```
echo Hello World
<<<<<<< HEAD
echo Zusaetzlicher Commit aus meiner Branch
=====
echo Zusaetzlicher Commit aus einer anderen Branch
>>>>>>>
```

- "<<<<<<< HEAD" bis "=====": Änderungen aus der lokalen Branch.
- "=====" bis ">>>>>>>": Änderungen aus der entfernten Branch.

`git merge` - **KONFLIKT**

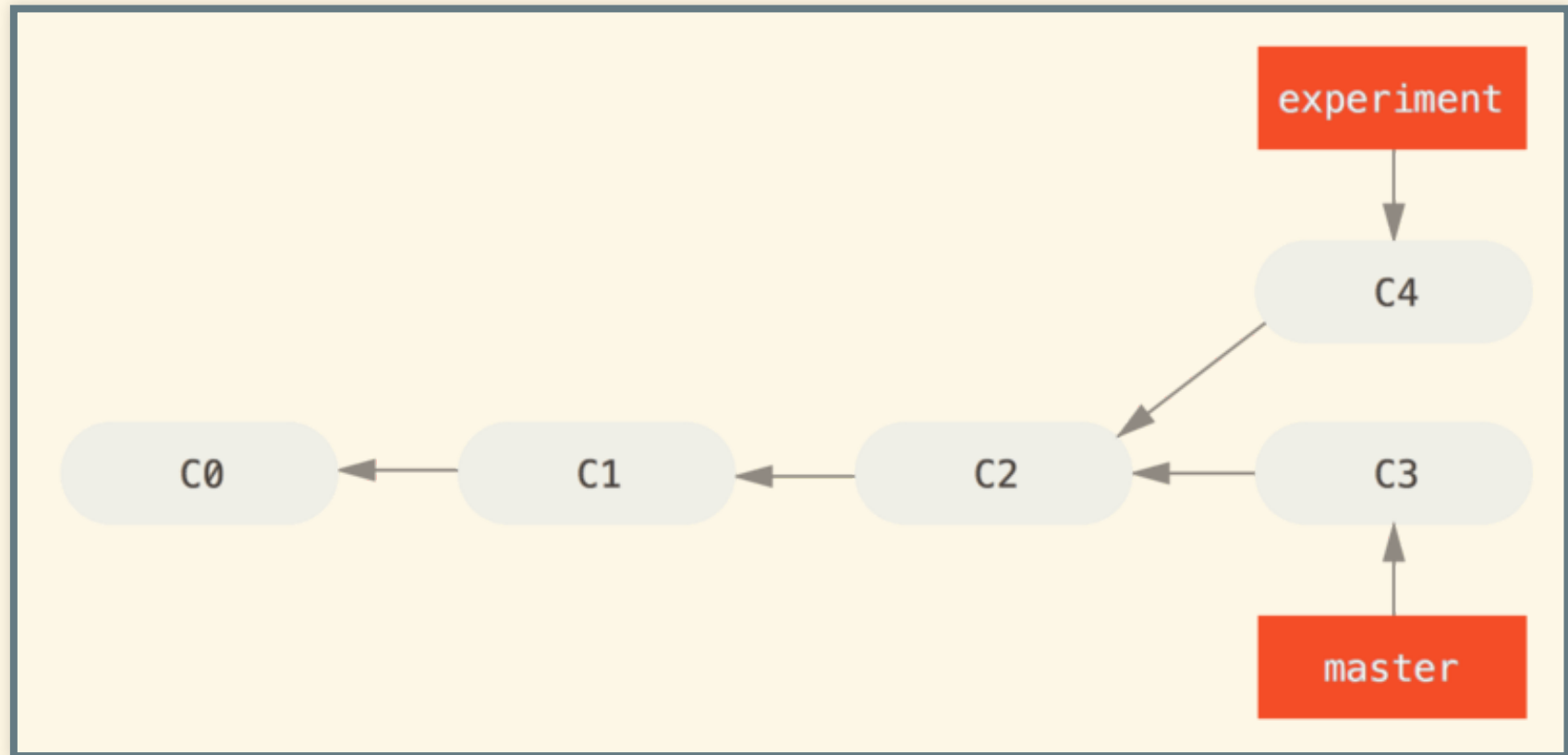
1. Führen Sie die Änderungen zusammen.
2. Entfernen Sie die Sonderzeichen von Git.
3. Fügen Sie die Änderung mit `git add` hinzu.
4. Überprüfen Sie den Status mit `git status`.

Wenn alle Dateien mit Konflikten zusammengeführt sind beenden Sie den Vorgang mit

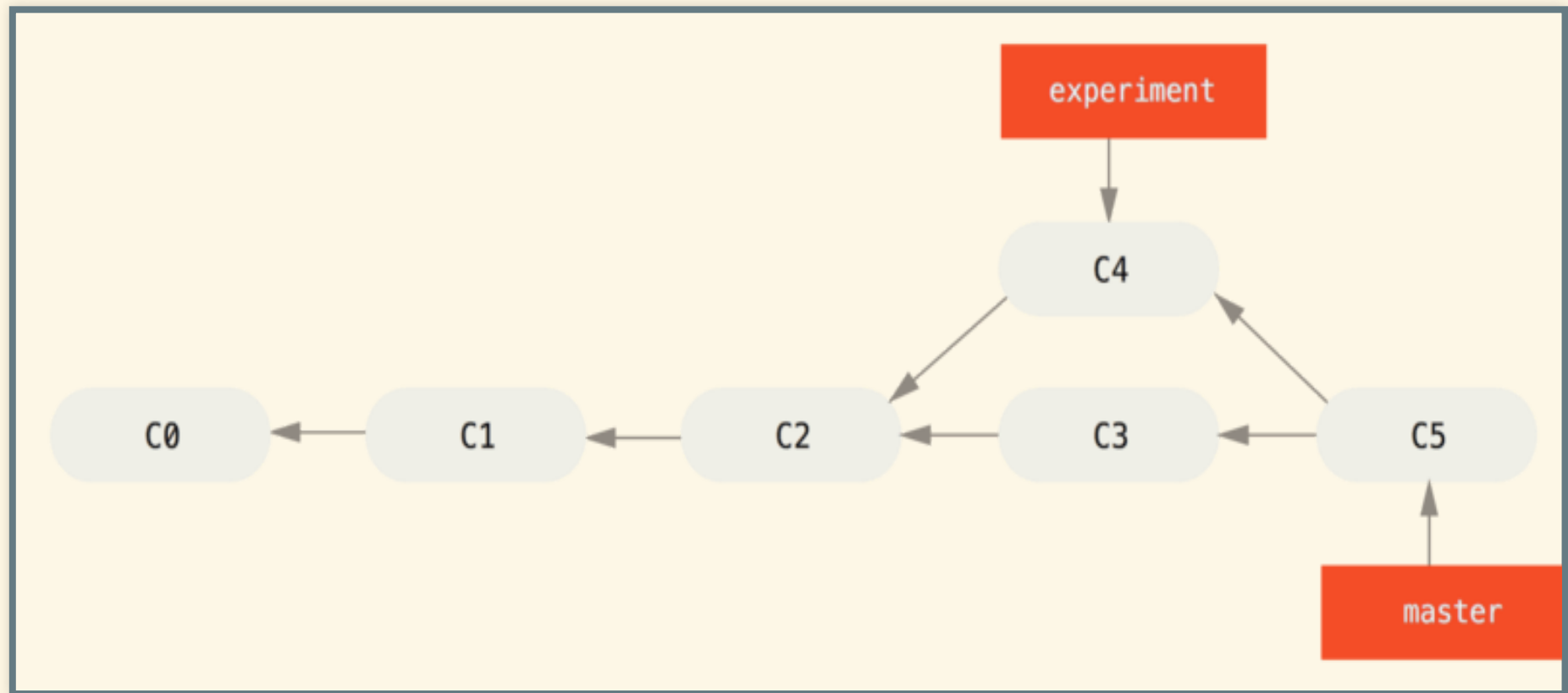
```
git merge --continue
```

git merge **VS** git rebase

Wie würde das Ergebnis eines Merge bei folgendem Ausgangszustand aussehen?

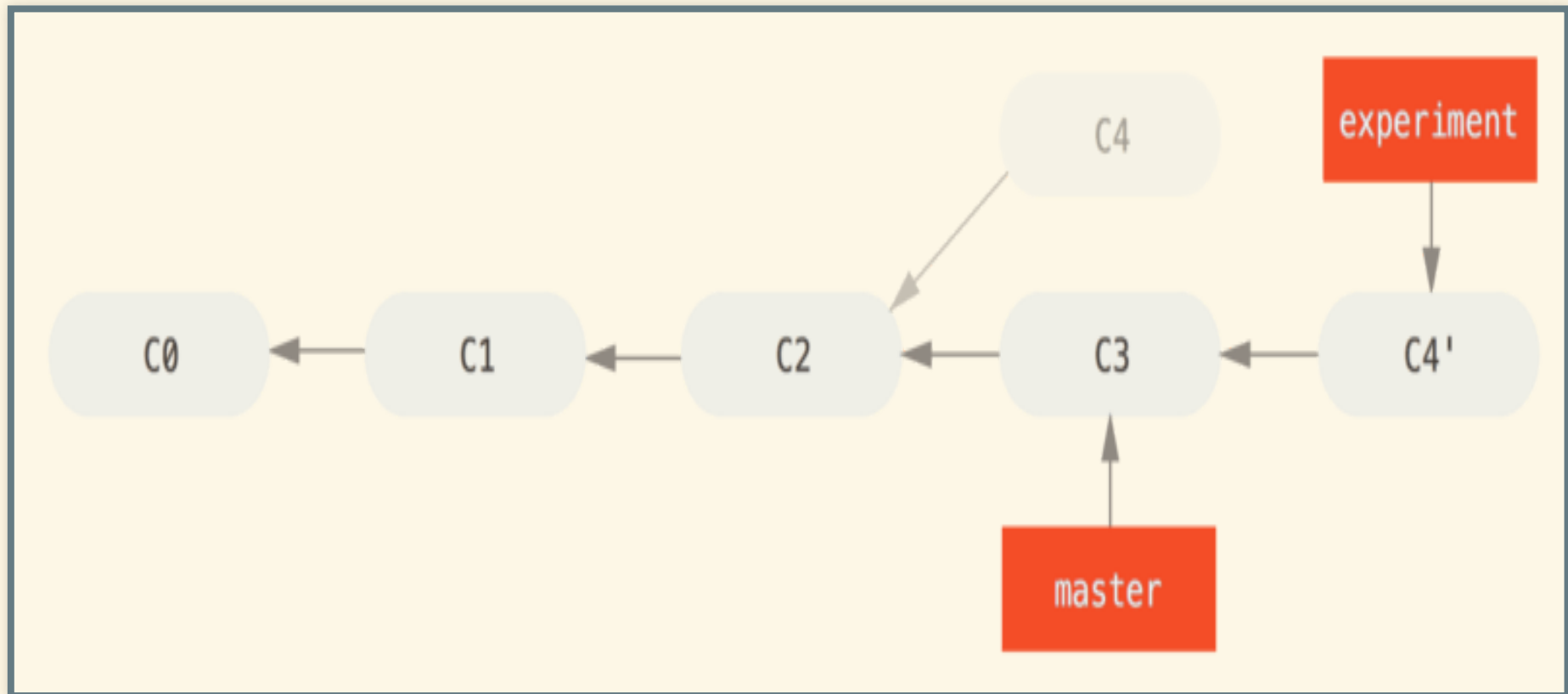


git merge - **ERGEBNIS**



git rebase

`git rebase` wendet die Veränderung eines Commits auf einen anderen Softwarestand an.



`git rebase` **VORSICHT!!!**

`git rebase` verändert die Historie. D.h. Sie sollten Git Rebase niemals auf Commits anwenden, die Sie bereits mit anderen geteilt haben.

```
git fetch
```

Empfängt Informationen aus dem Remote Repository. Folgenden Entitäten (refs) werden abgeholt:

- Branch
- Tag
- Commit

Der HEAD wird noch nicht verändert.

ÜBUNG 2 & 3 - PARALLELES ARBEITEN (MIT KONFLIKT)

.GITIGNORE

In dieser Datei können Dateien aufgenommen werden, sodass diese bei Änderungen nicht berücksichtigt werden. Auch Dateiendungen bzw. Wildcard Patterns sind möglich.

```
# ignore all .log files
*.log

# but do track lib.log, even though you're ignoring .log files above
!lib.log

# ignore the target directory
target/
```

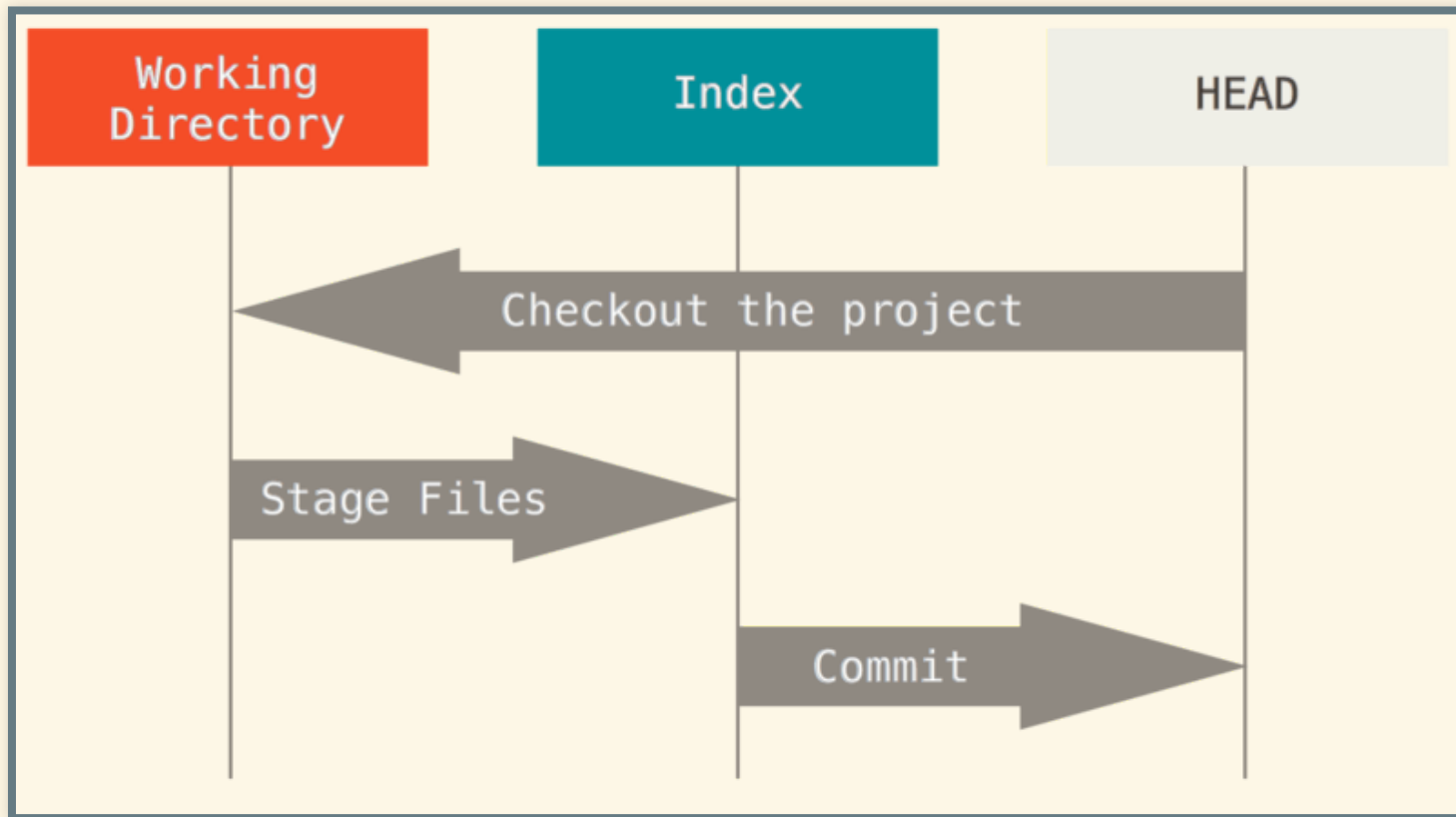
git log

Zeigt die Historie von Git an. Die Option **--graph --oneline** verbessert die Lesbarkeit.

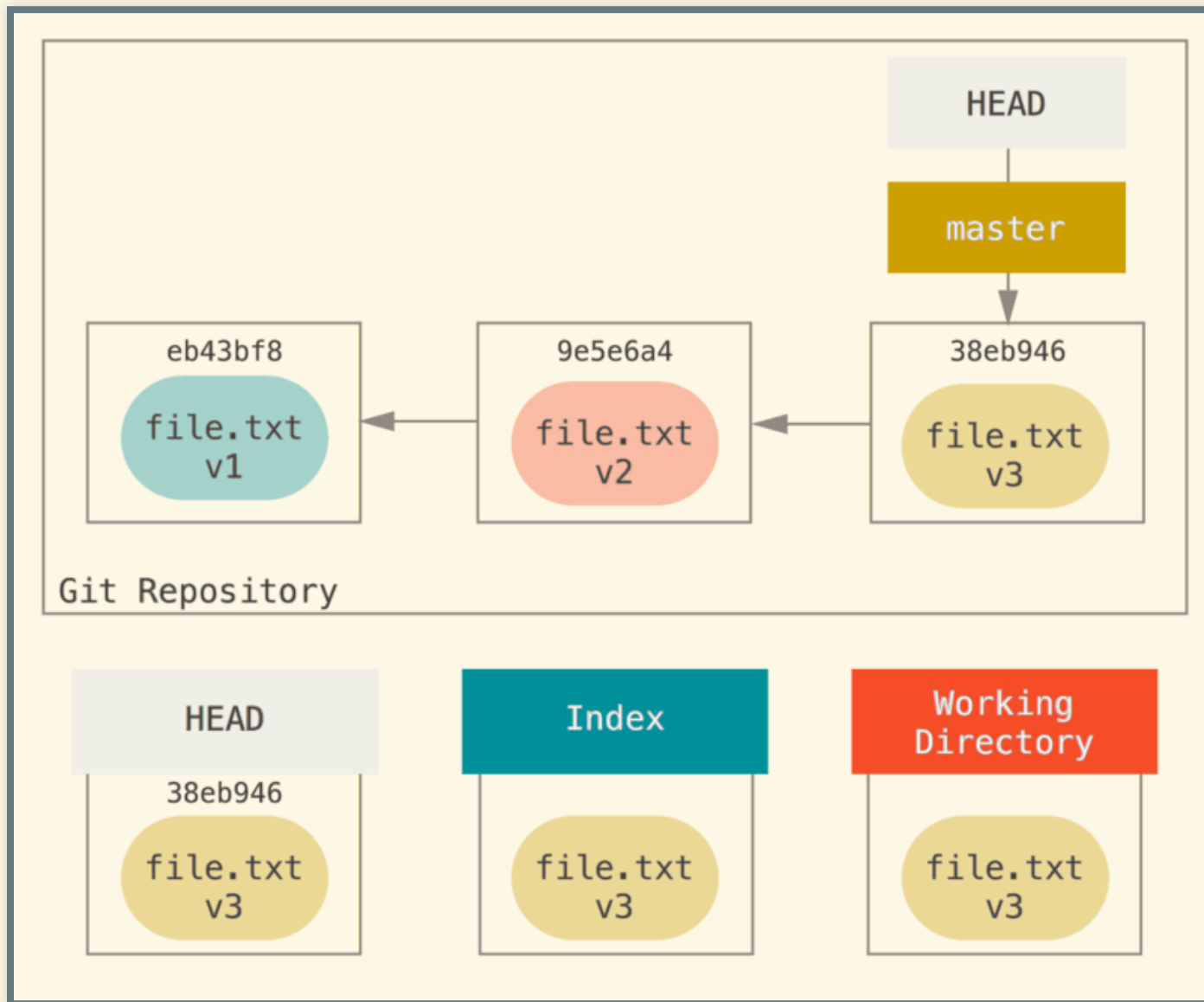
```
git log --graph --oneline
* 92e03be (HEAD -> master, sshorigin/master) Merge remote-tracking branch 'refs/remot
|\
| * 02ac5df Meine Erwartungen
* | 3930dff Erwartungen 3
|/
* 1b72bf0 Meine Erwartungen 2
* 774da85 (origin/master, origin/HEAD) Feature: Contributor.md added
* 562fbb8 (newBranch) My first commit
```

git reset

Manchmal muss man den *Arbeitsbereich*, den *Index* oder den *HEAD* verändern.

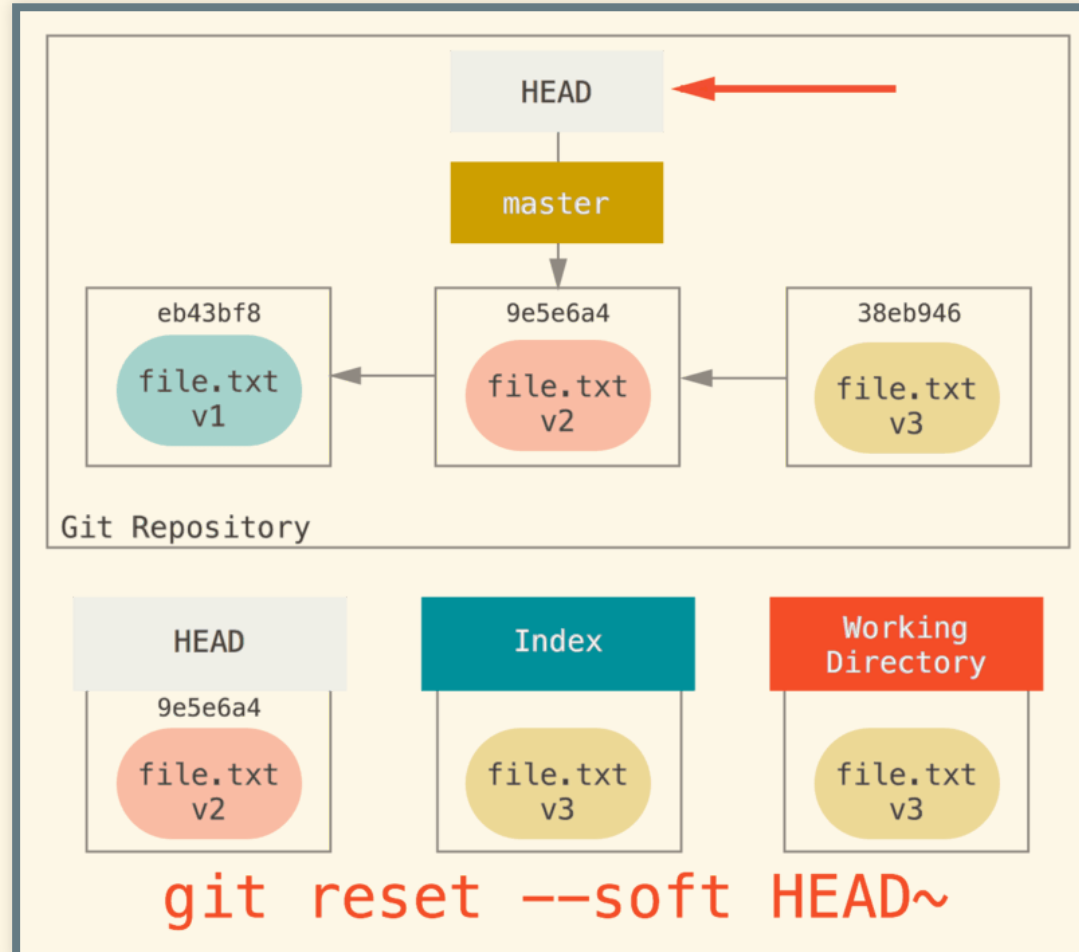


git reset



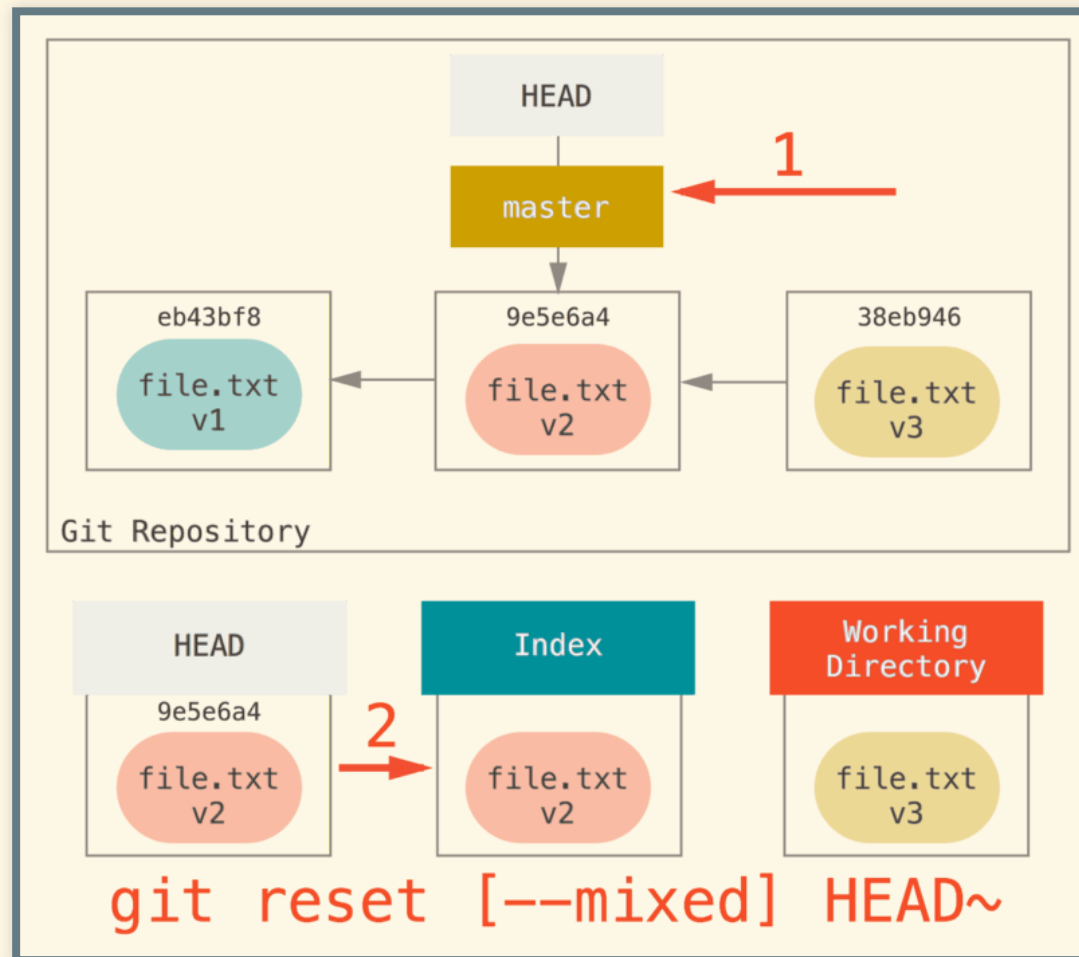
```
git reset --soft
```

Setzt lediglich HEAD zurück, Index und Arbeitsbereich bleiben gleich.



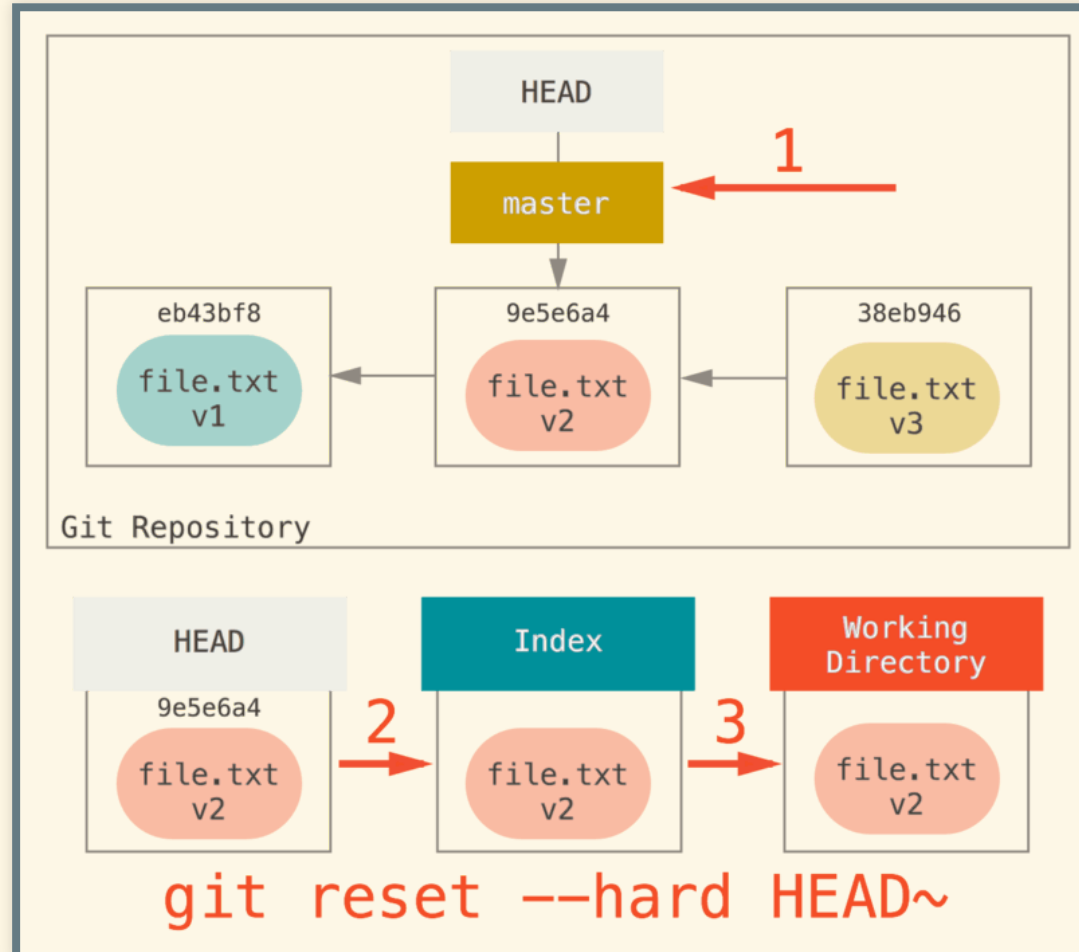
```
git reset --mixed
```

Setzt HEAD und Index zurück, der Arbeitsbereich bleibt gleich.



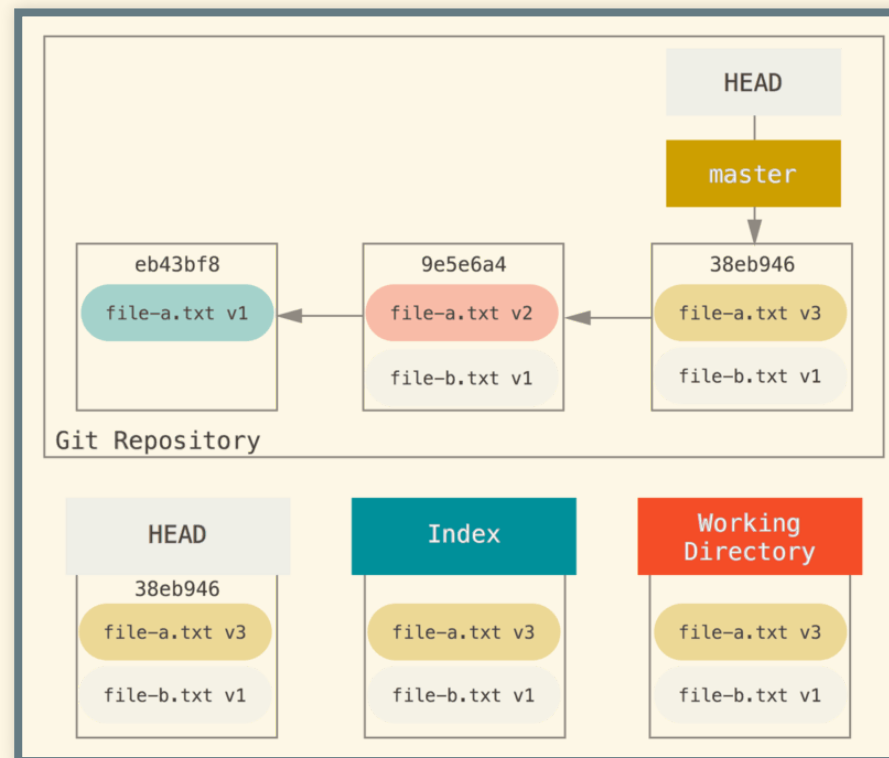
```
git reset --hard
```

Setzt HEAD, Index und Arbeitsbereich (Vorsicht!) zurück.

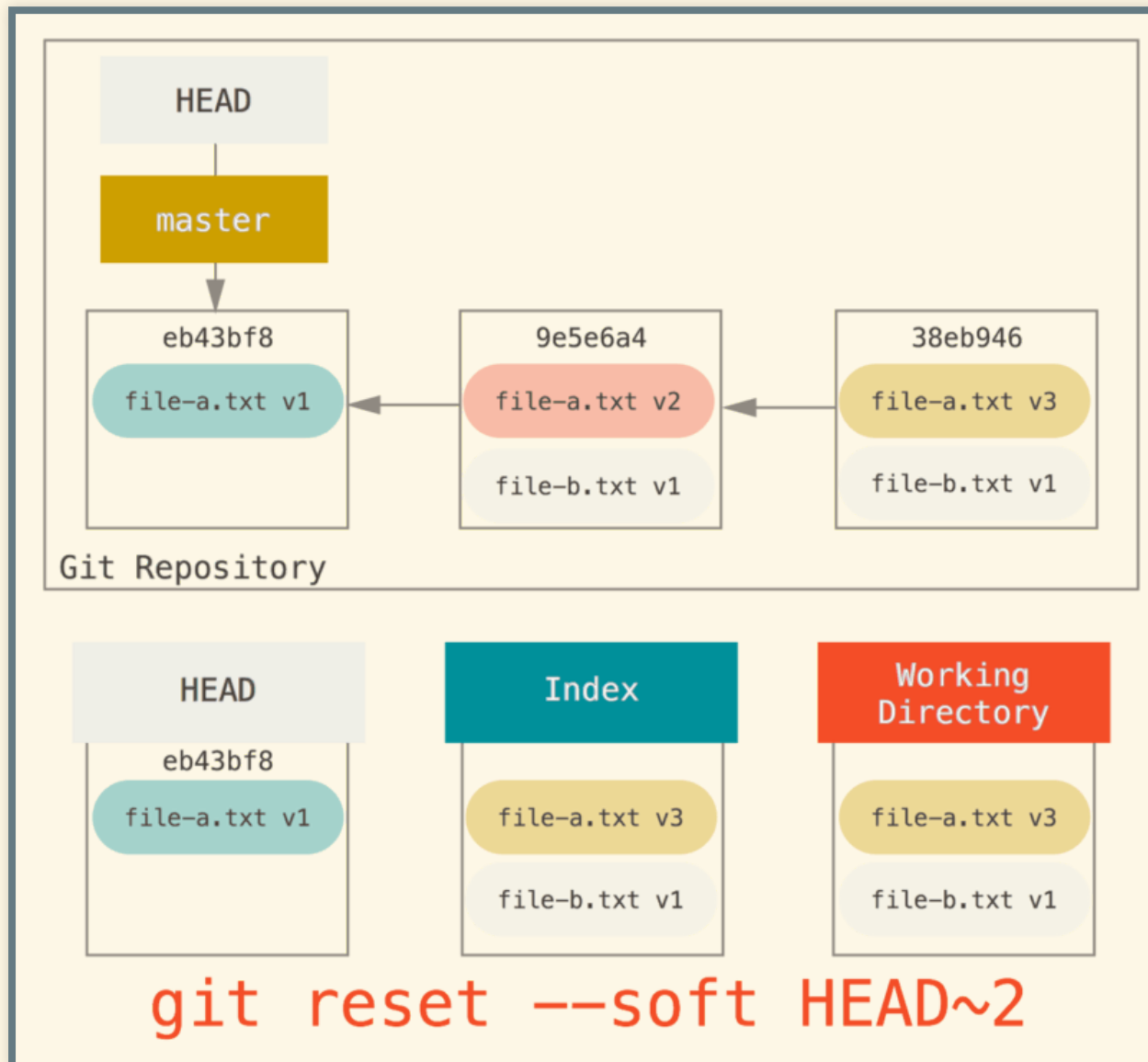


git reset **SQUASH COMMITS**

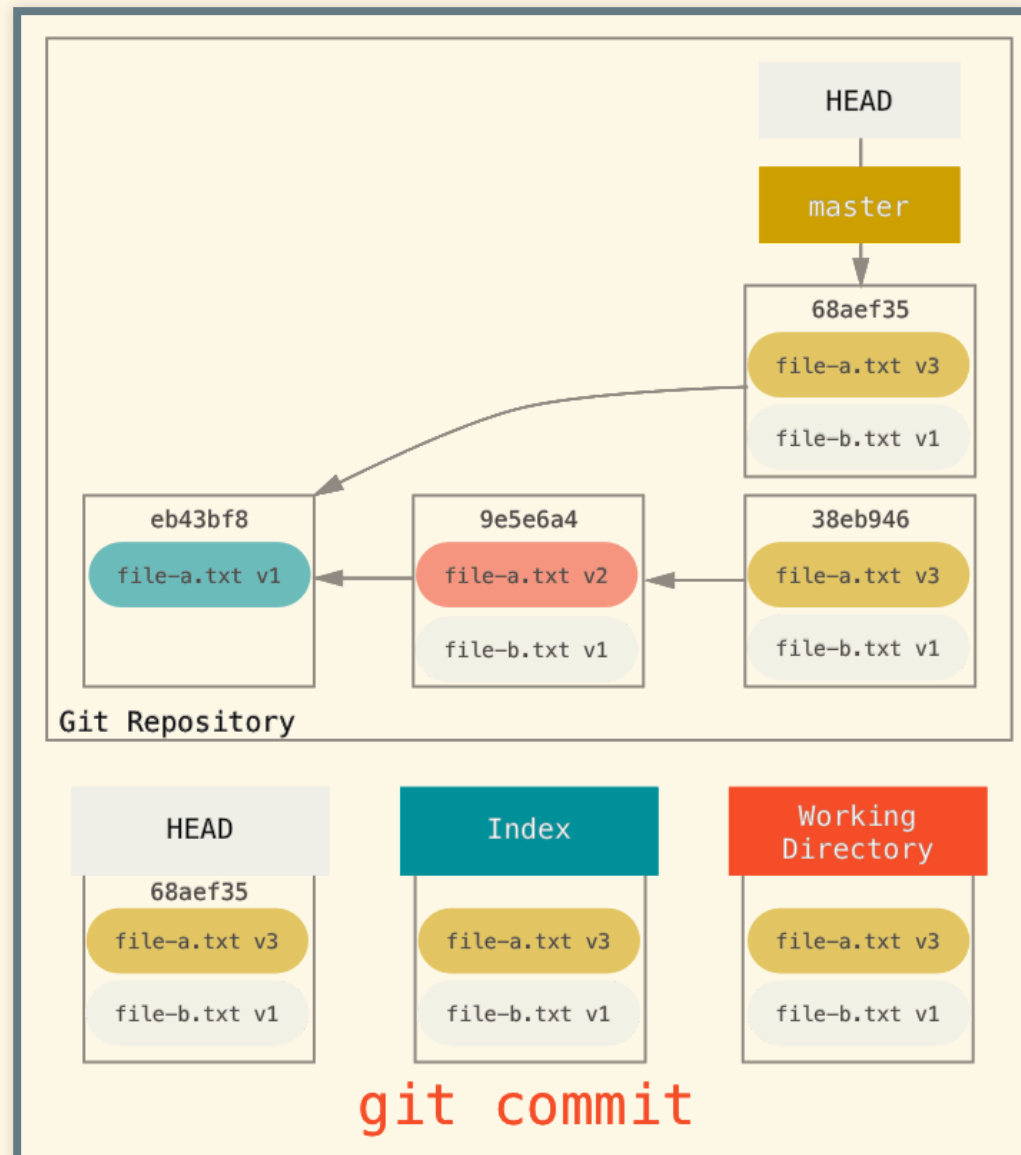
Wenn man Commits wie "WIP", "noch nicht fertig" oder "Buggy" hat, möchte man diese in der Regel zusammenführen.



git reset SQUASH COMMITS



git reset SQUASH COMMITS



THE END

LINKS

- [Git Cheat Sheet](#)
- [Git Book](#)
- [Source Tree \(Git GUI\)](#)