Apache Subversion (SVN)

Datamining und Sequenzanalyse

Marvin Meusel, Sascha Winter

18.10.2013

~~Apache Subversion (SVN)~~

Datamining und Sequenzanalyse

Marvin Meusel, Sascha Winter

18.10.2013

git

Datamining und Sequenzanalyse

Marvin Meusel, Kai Dührkop

18.10.2013

Motivation

Gemeinsam an Sourcecode arbeiten: Fileserver?

N F S

NETWORK FILE SYSTEM

Motivation

Gemeinsam an Sourcecode arbeiten: Fileserver?

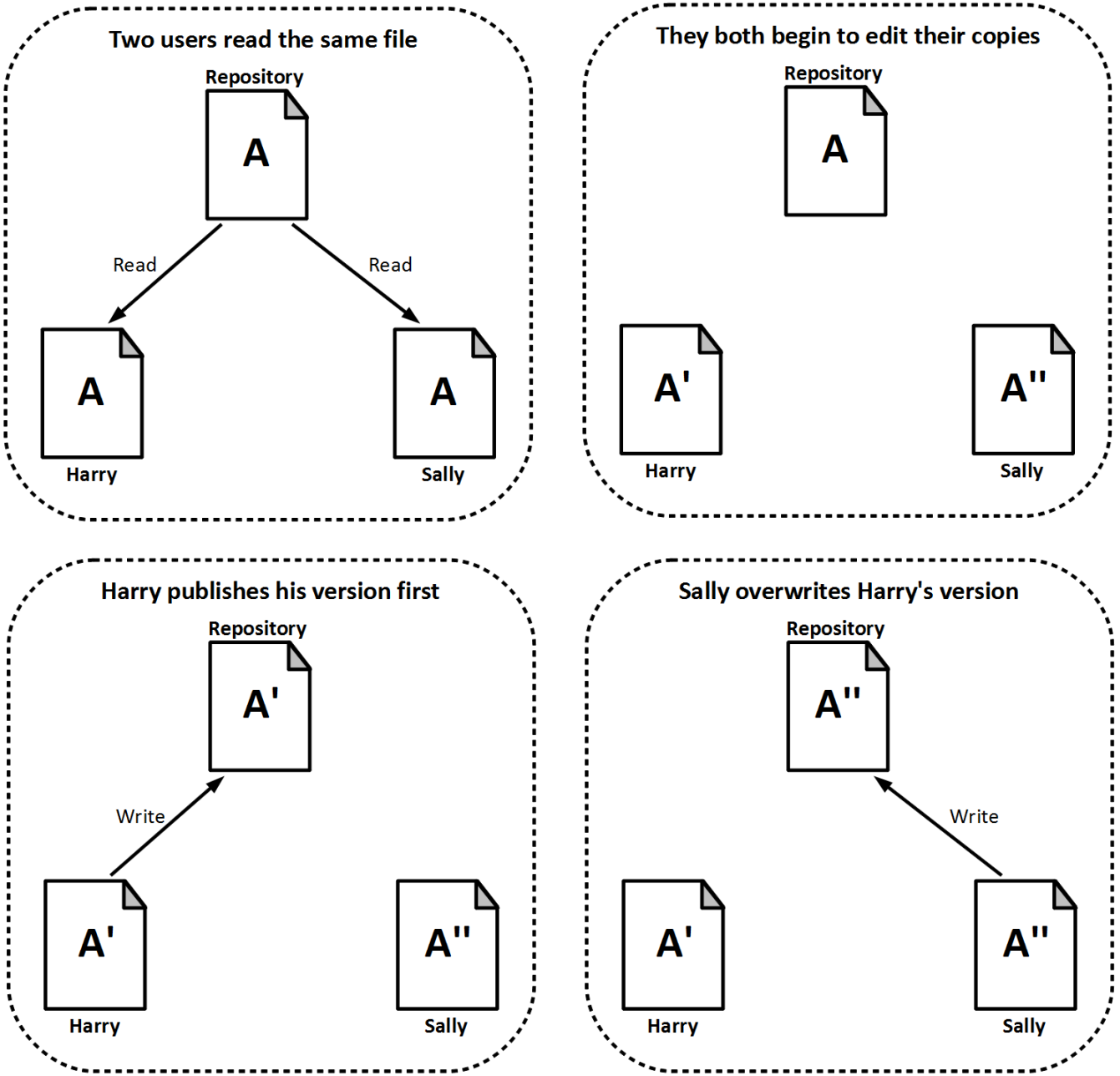
samba

ssh

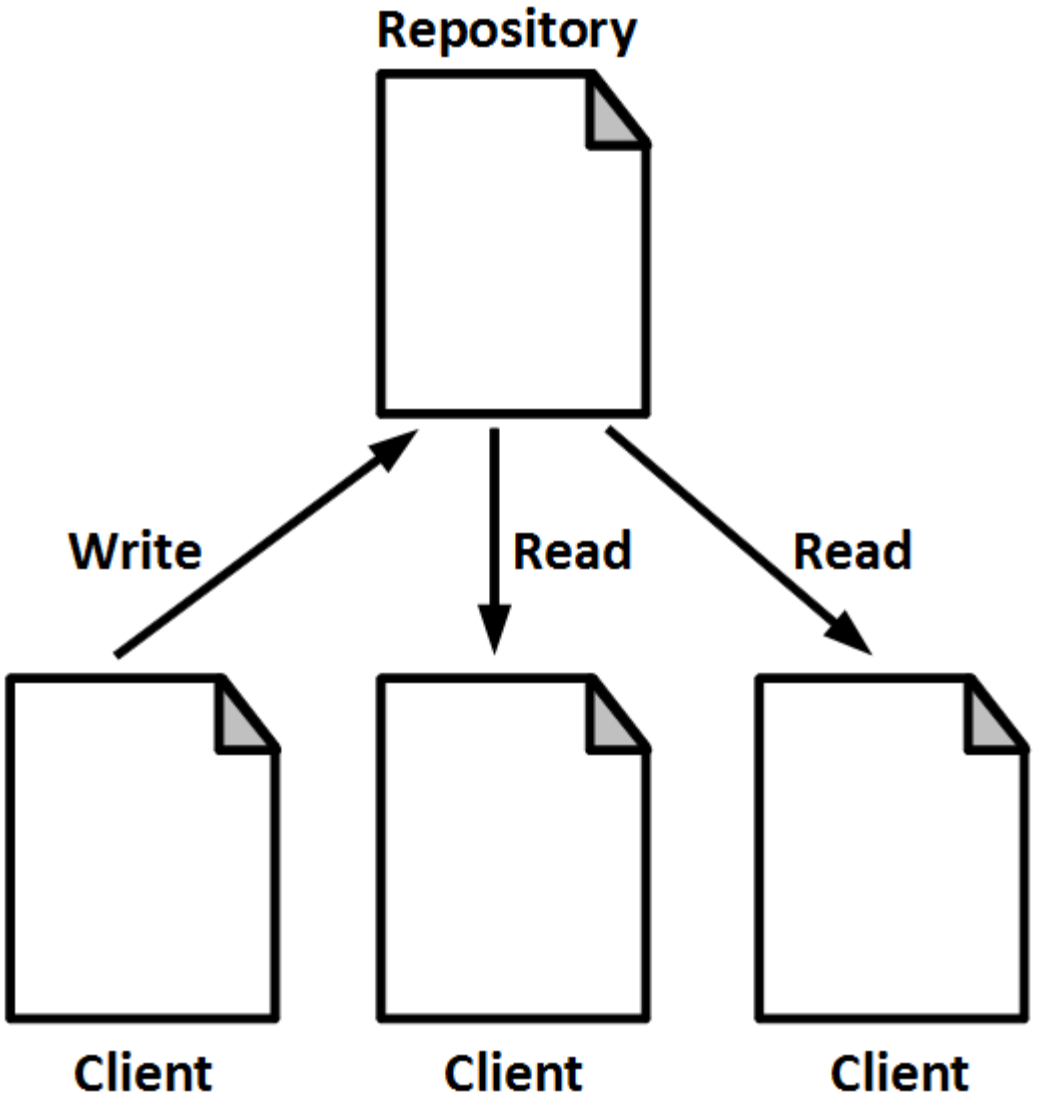
N F S

NETWORK FILE SYSTEM

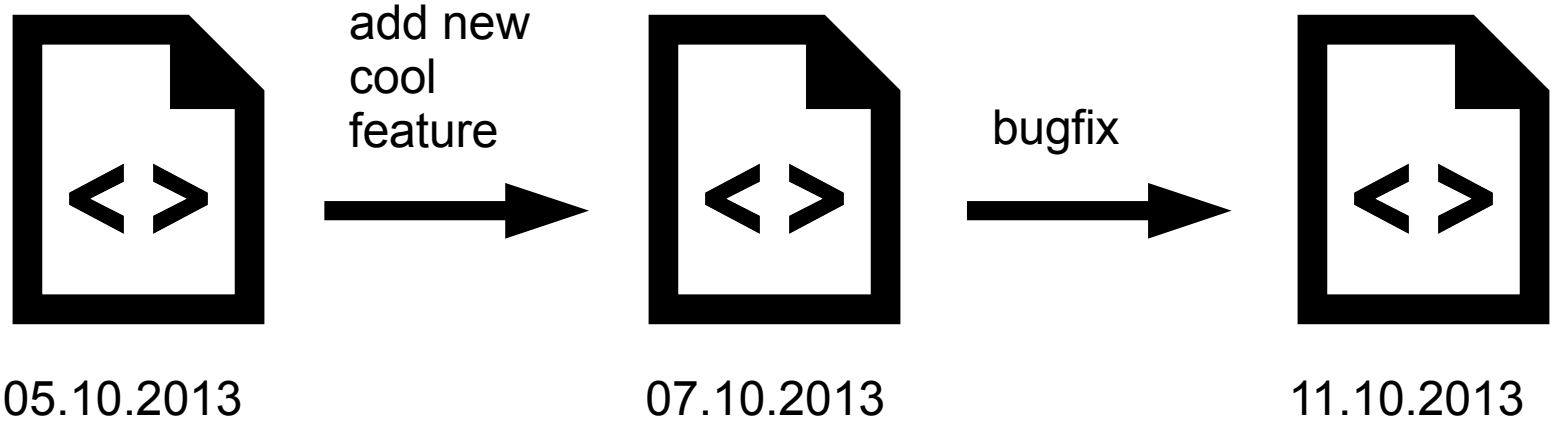
Das Problem verteilter Dateizugriffe



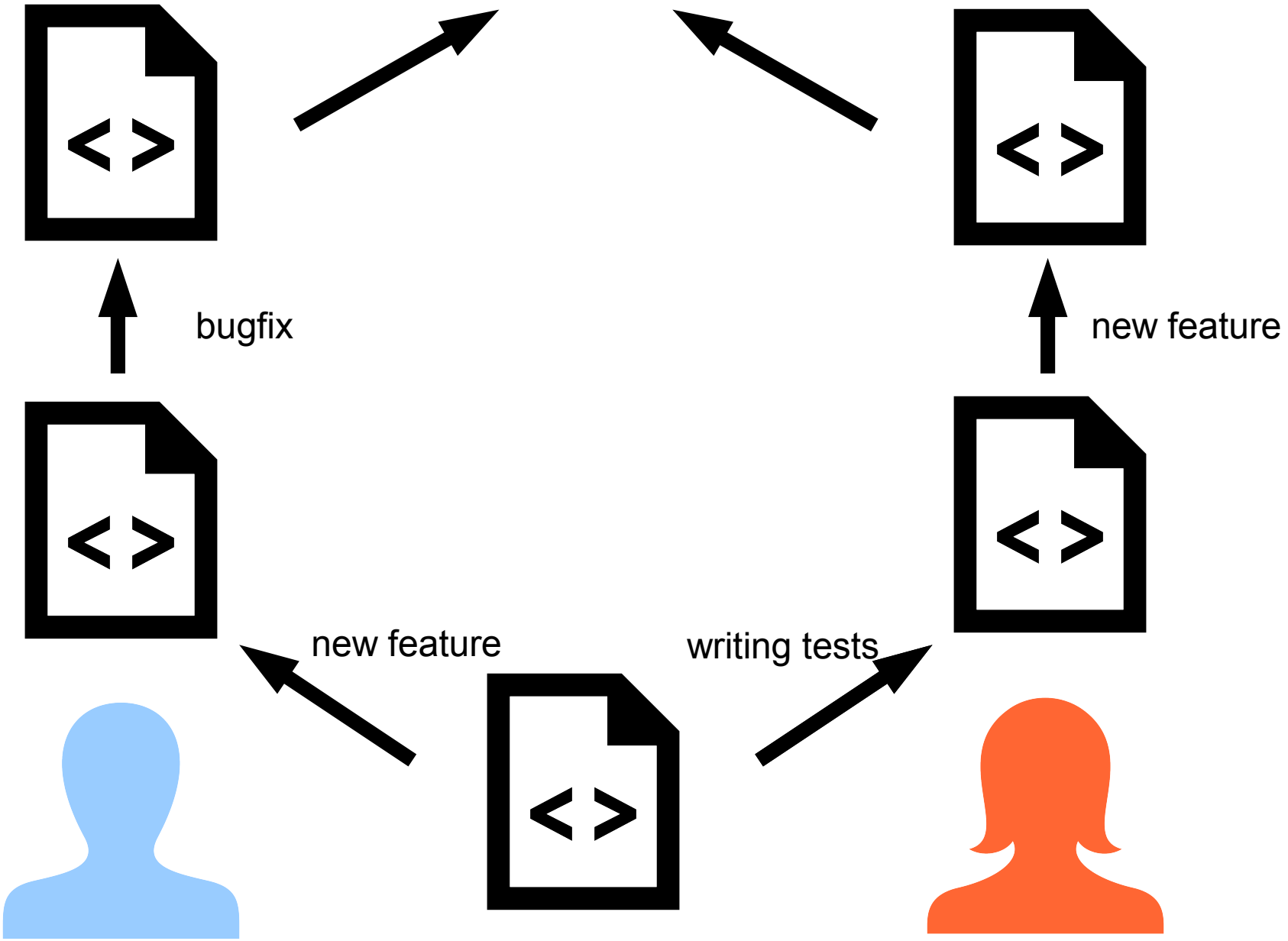
Versionsverwaltung



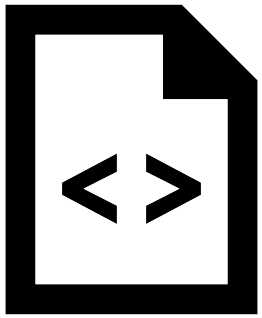
Versionierung... über die Zeit

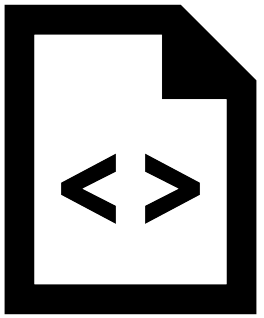


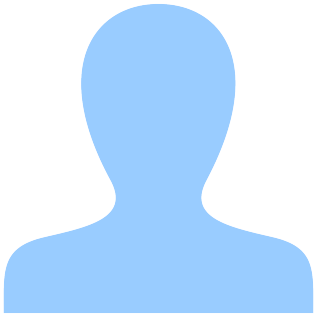
Versionierung... über mehrere Entwickler



Konfliktlösung







git Repositories

- login2.minet.uni-jena.de
- Repositories: /data/bipr/biodmXX
- Zugriff über git + ssh (Windows: putty client)

Beispiel:

ssh <nutzername>@login2.minet.unijena.de
cd /data/bipr/biodm08

Einmalig: Identifizieren

```
kaidu:linux$ $ git config --global user.name
"Kai"
$ git config --global user.email kai@du.de
```

Anlegen eines zentralen git Repositories

```
kaidu:biodm08$ mkdir exactSearch.git
kaidu:biodm08$ git init --bare --shared
Initialisierte leeres gemeinsames Git-
Repository in
/data/bipr/biodm04/exactSearch.git/
```

- **shared** setzt die **Gruppenrechte** aller Files und Ordner im git repository, so dass andere Gruppenmitglieder darauf zugreifen können
- **bare** setzt das Repository als „**zentrales**“ Repository, in dem niemand direkt Änderungen durchführt. Ohne --bare wird das Repository zum **Arbeitsrepository**

Anlegen eines lokalen git Repositories

```
kaidu:linux$ git clone kaidu@login2.minet.uni-
jena.de:/data/bipr/biodm04/exactSearch.git
Cloning into 'exactSearch'
```

- **clone** legt eine lokale Kopie eines externen Repositories an

Hinzufügen von Dateien

```
kaidu:linux$ cat > anewfile.txt
Hello World!
kaidu:linux$ git add anewfile.txt
```

• cat > file ist ein bekannter „Trick“ um eine Datei mit Text zu füllen. Ihr könnt auch einfach einen Texteditor benutzen ;)

• **add** fügt eine Datei der **stage area** hinzu (= Menge der Dateien, die beim nächsten Commit berücksichtigt werden)

• **add <directory>** führt add auf alle Dateien im directory rekursiv aus

Löschen von Dateien

```
kaidu:linux$ git rm anewfile.txt
```

• löscht Datei (mit dem nächsten Commit)

```
kaidu:linux$ git rm --cached anewfile.txt
```

• löscht Datei aus dem Index/Versionierungssystem, behält sie aber auf der lokalen Festplatte

• Achtung: Einmal versionierte Dateien bleiben natürlich sowieso immer erhalten und können jederzeit wieder hergestellt werden!

Versionierung

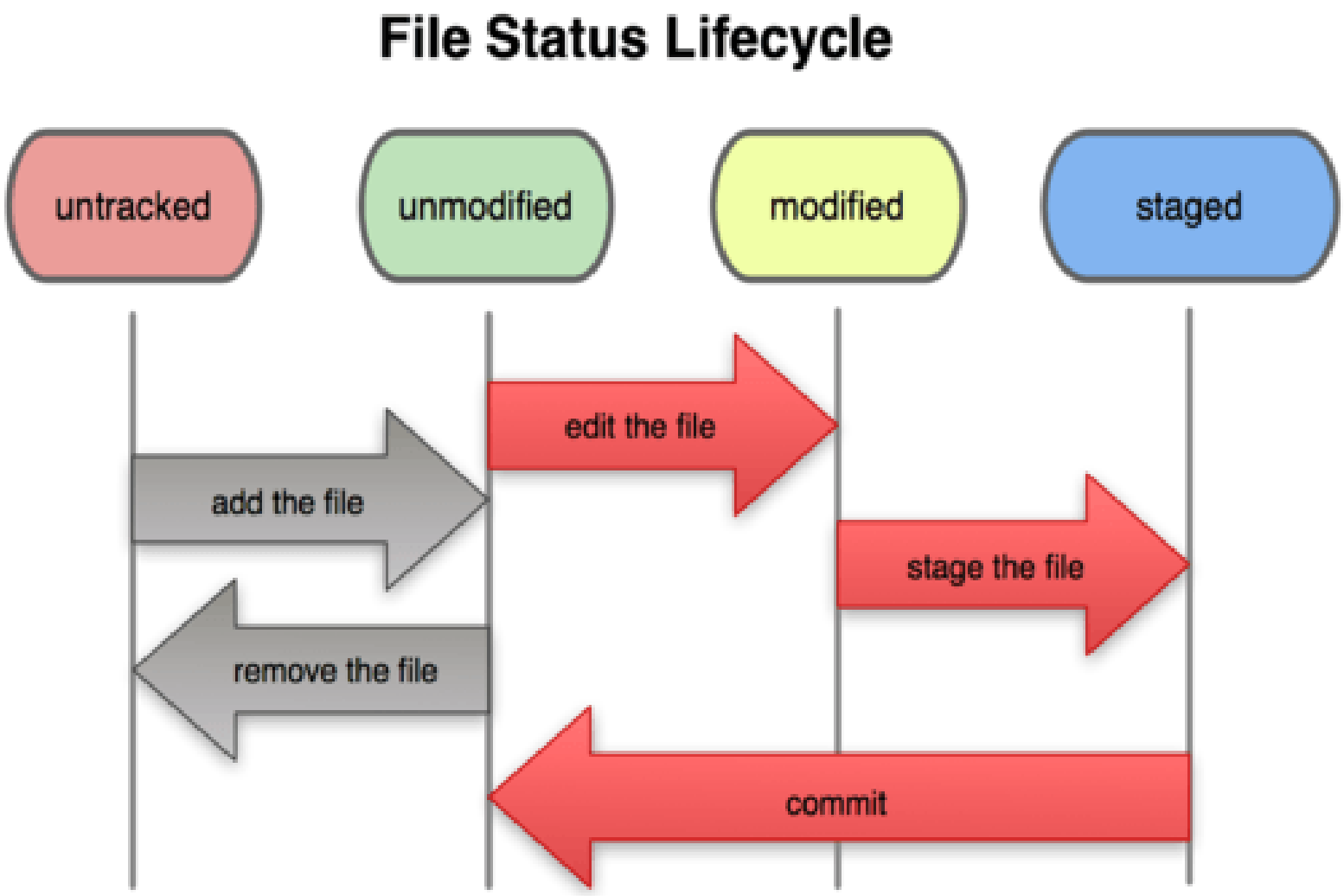
```
kaidu:linux$ git commit -m 'file added'
```

- speichert alle Änderungen (**staged** files) in einer neuen Version ab
- speichert eine Log-Message ab

```
kaidu:linux$ git commit -am 'files added'
```

- speichert alle Änderungen (**tracked** files) in einer neuen Version ab

Mögliche Zustände einer Datei im Repository



Logs und Status

```
kaidu:linux$ git status
```

- gibt an welche Dateien neu erstellt, modifiziert und gestaged sind
- gibt viele weitere Informationen (aktueller Branch etc.) an

```
kaidu:linux$ git log
```

- zeigt die letzten Commits mit ihren Logmessages

Synchronisieren mit Server

```
kaidu:linux$ git push origin master
```

- überträgt all eure Commits (im Branch 'master') zum Server (names 'origin')
- muss nicht unbedingt nach jedem commit aufgerufen werden, aber doch einmal am Tag ;)

```
kaidu:linux$ git pull origin master
```

- überträgt alle Commits auf dem Server auf euren Rechner
- merged die Commits im Server mit euren neuen Commits
- meldet mögliche Konflikte

Konflikt

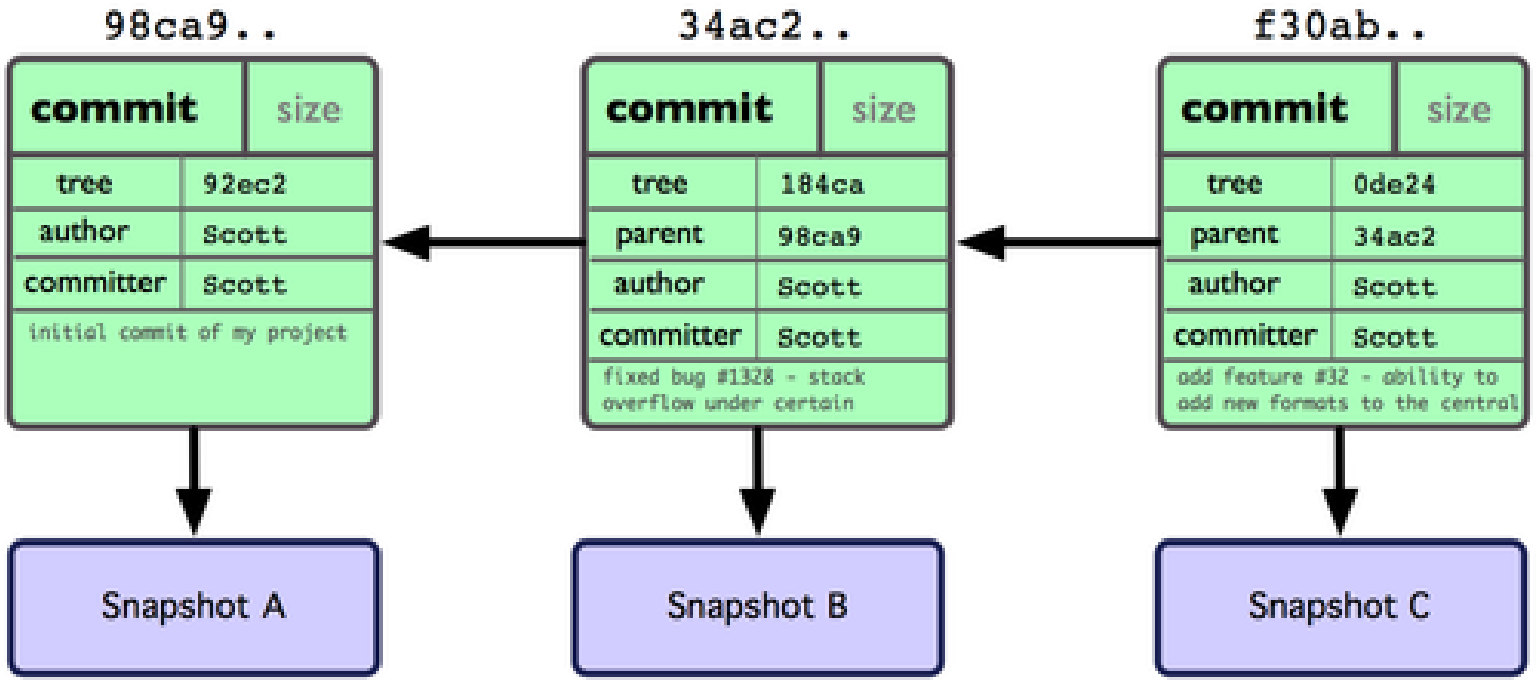
```
kaidu:linux$ git mergetool
```

- git löst Konflikte in den meisten Fällen selbstständig auf, nämlich dann wenn die Änderungen an unterschiedlichen Stellen stattfinden
- wenn aber in der selben Datei in direkter Nähe Änderungen stattfinden, muss der Konflikt manuell gelöst werden
- git **mergetool** ruft einen grafischen Editor für Konfliktlösung auf
- beide Versionen einsehen, sich für eine gemeinsame Version entscheiden

Workflow

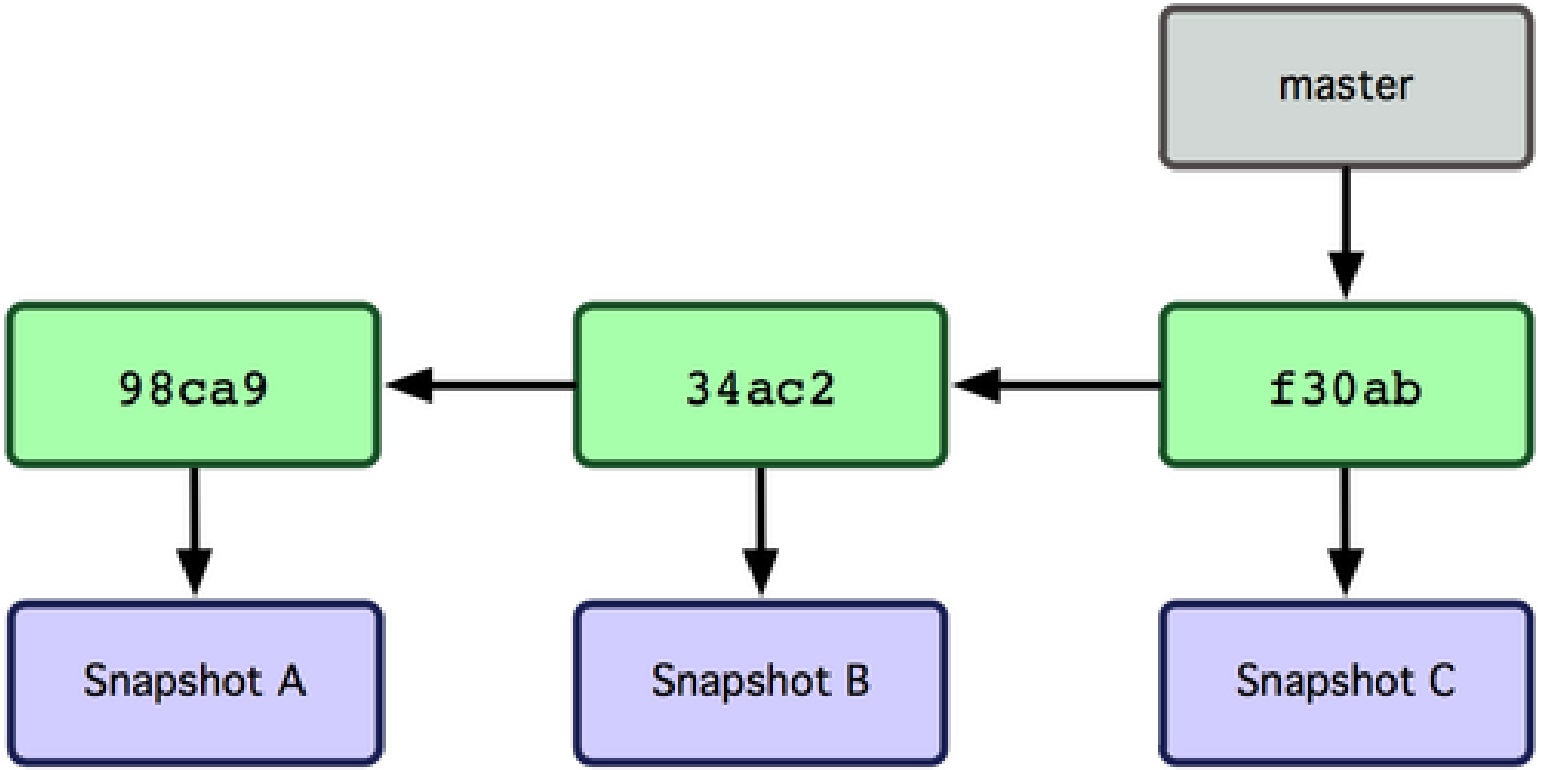
```
kaidu:linux$ git pull # Hole aktuelle Version
kaidu:linux$ change some files...
kaidu:linux$ git commit -am 'my changes'
kaidu:linux$ git pull # Prüfe auf Konflikte
kaidu:linux$ git push # Sende an Server
```

Exkurs: Branches



Exkurs: Branches

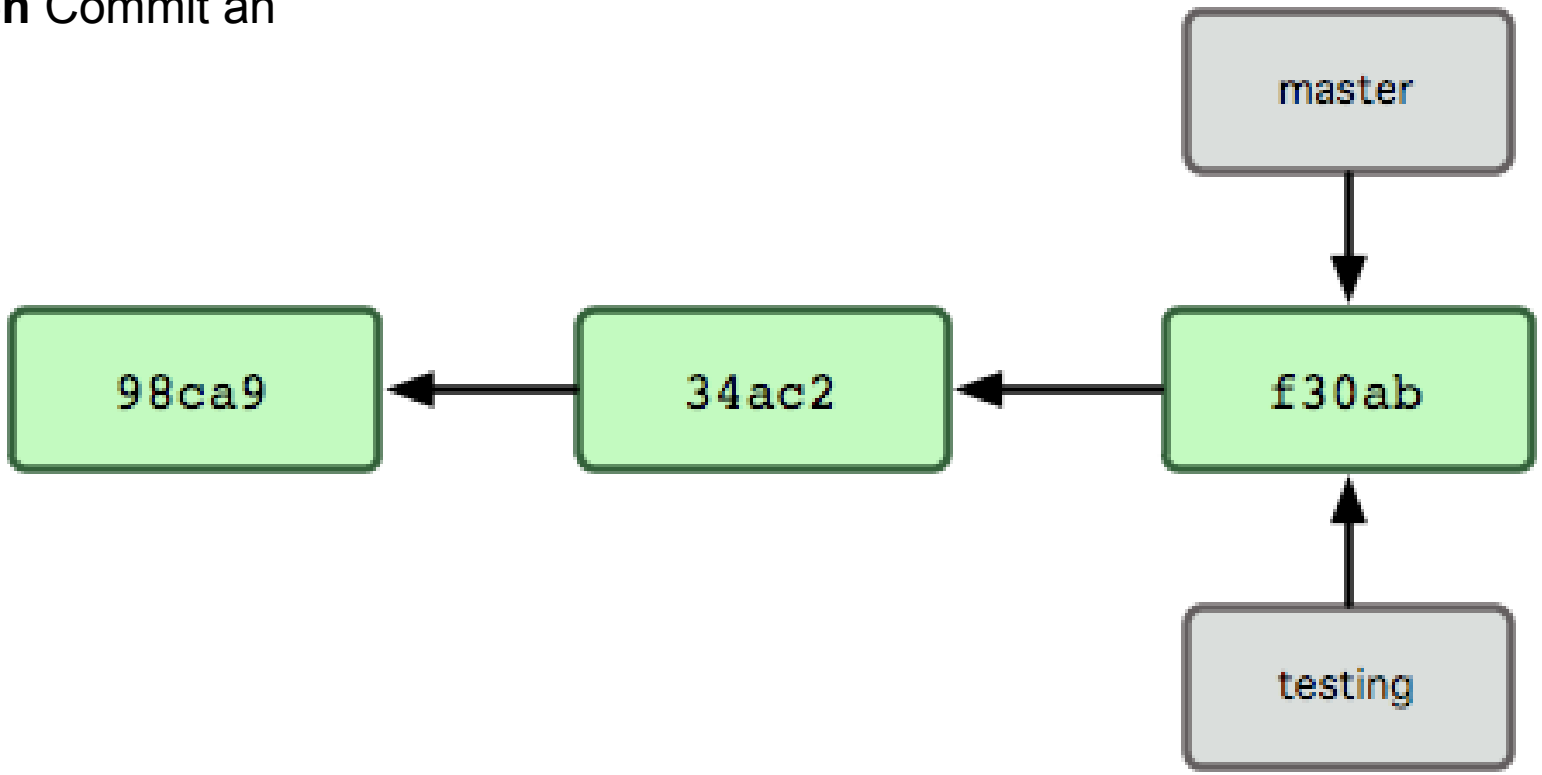
- ein Branch ist ein **Zeiger** auf einen Commit



Exkurs: Branches

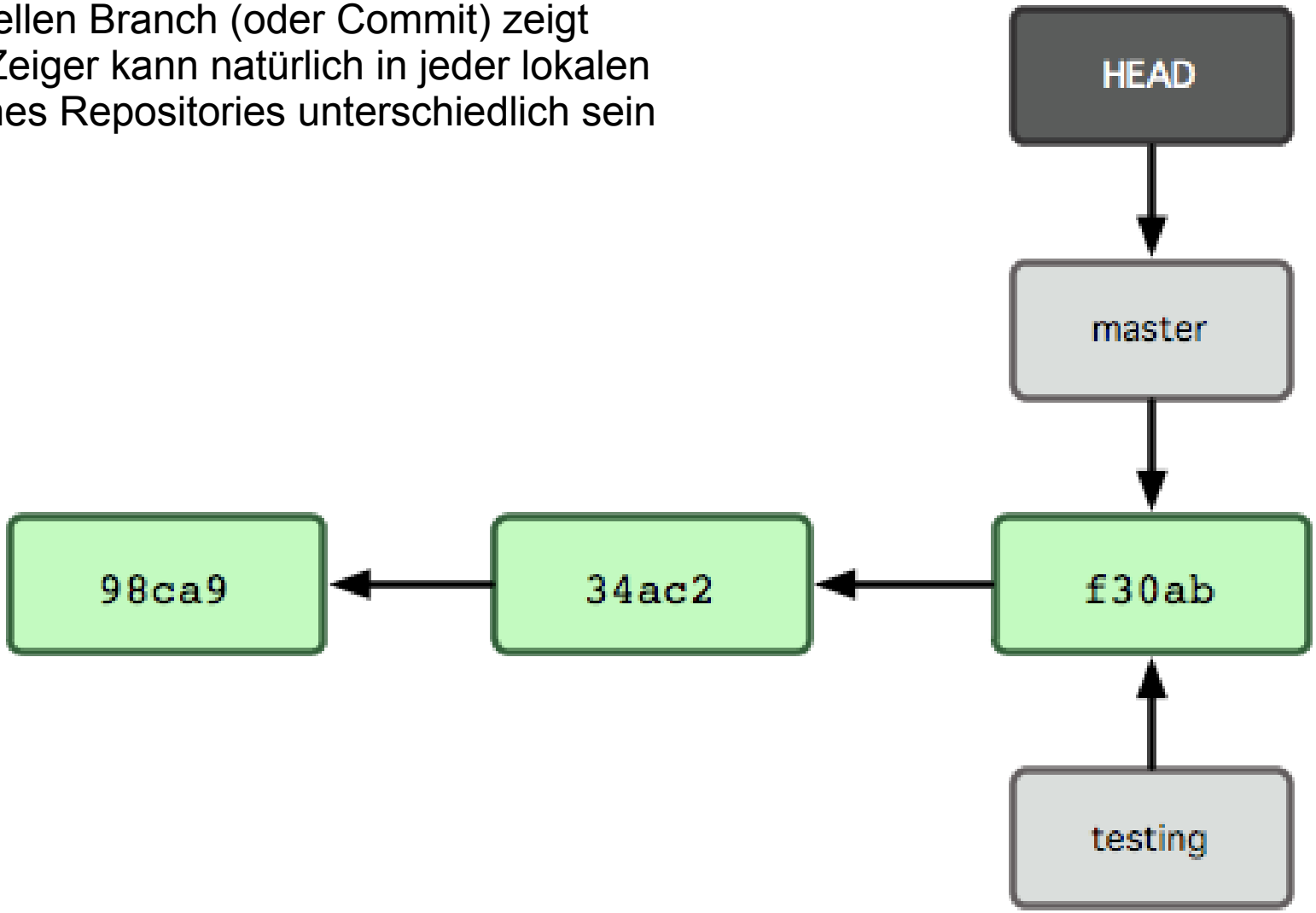
```
kaidu:linux$ git branch testing
```

• der **branch** Befehl legt einen neuen Branch im **aktuellen** Commit an

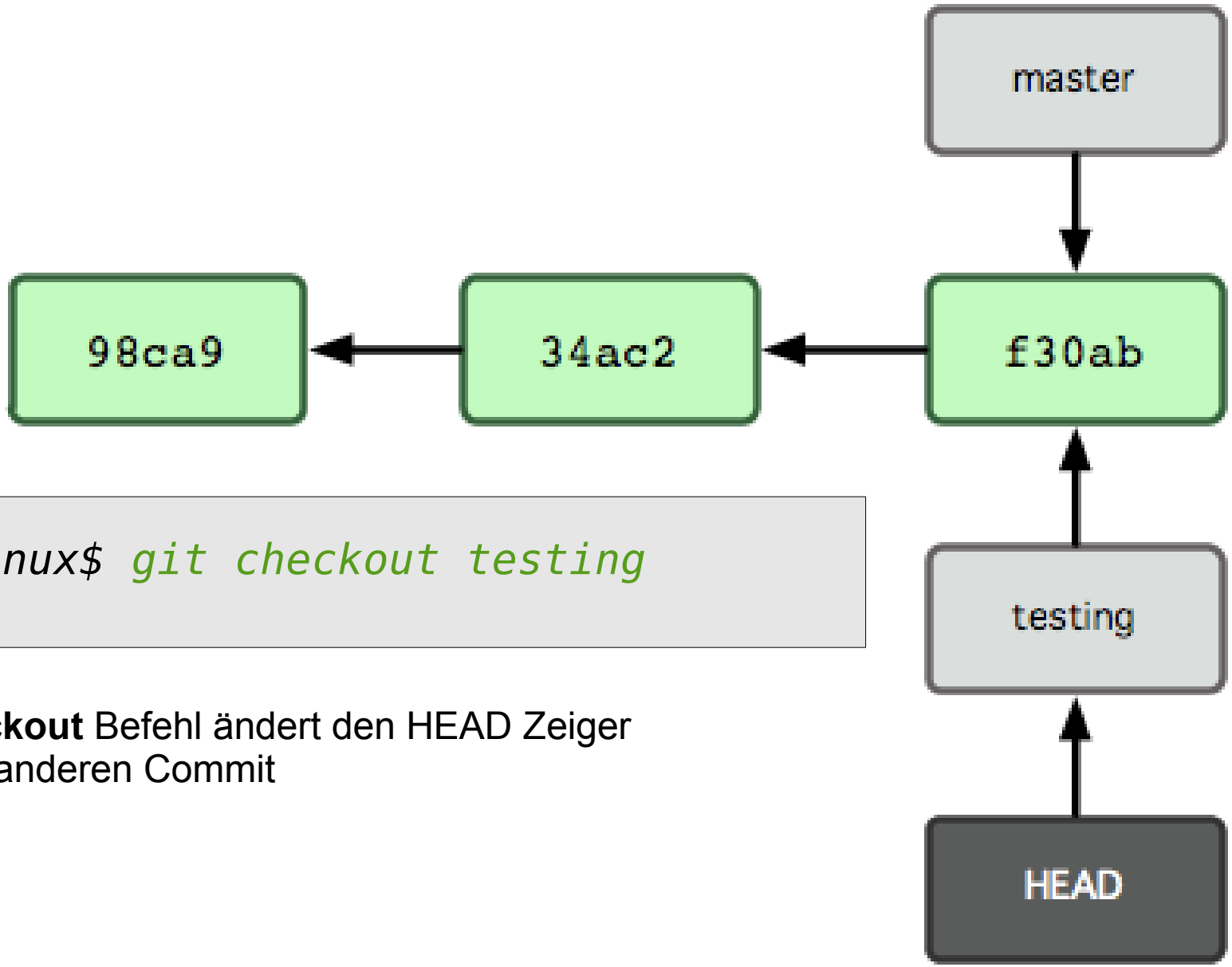


Exkurs: Branches

- **HEAD** ist ein spezieller Zeiger der immer auf den aktuellen Branch (oder Commit) zeigt
- dieser Zeiger kann natürlich in jeder lokalen Kopie eines Repositories unterschiedlich sein



Exkurs: Branches



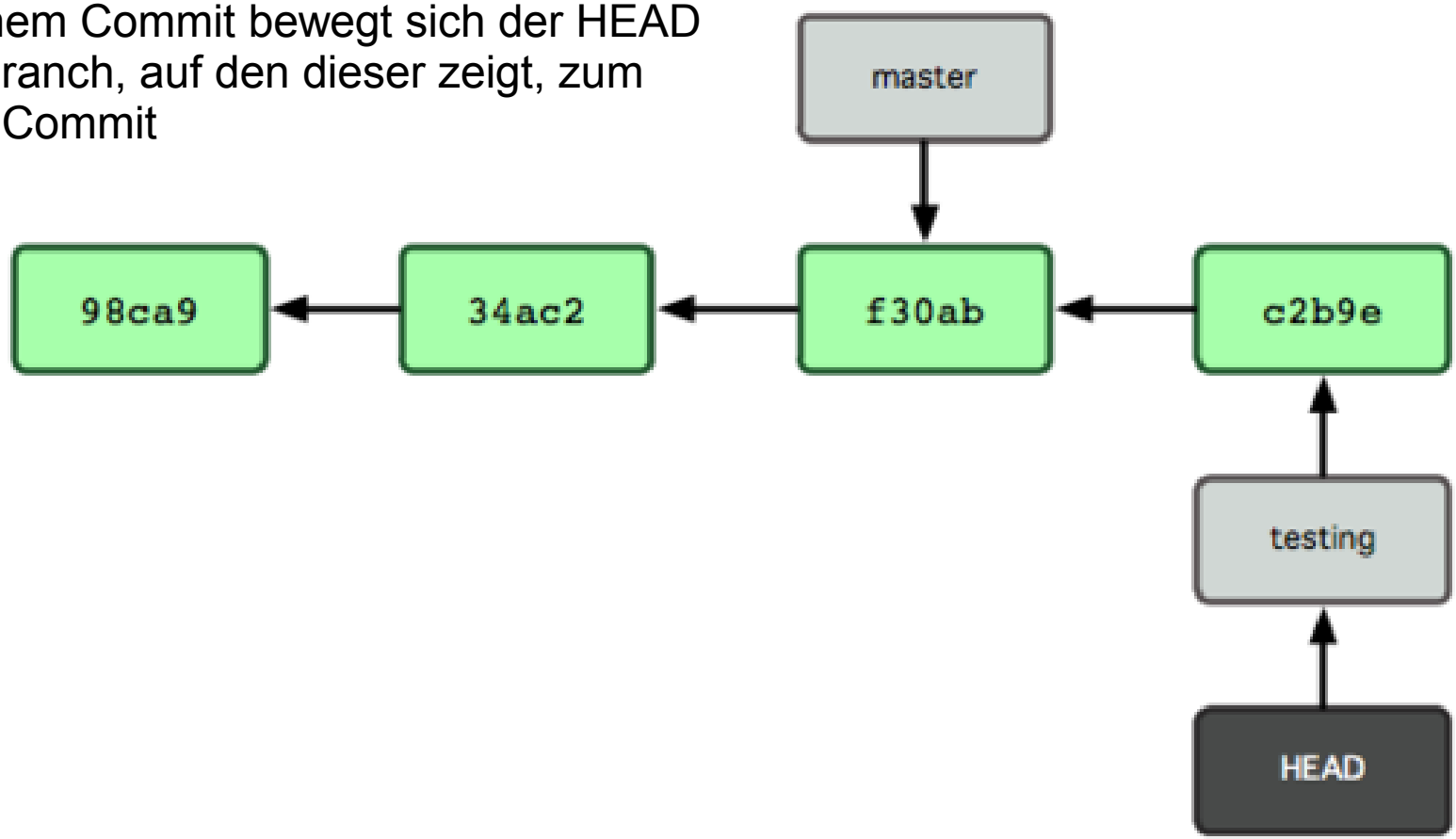

```
kaidu:linux$ git checkout testing
```

- der **checkout** Befehl ändert den HEAD Zeiger auf einen anderen Commit

Exkurs: Branches

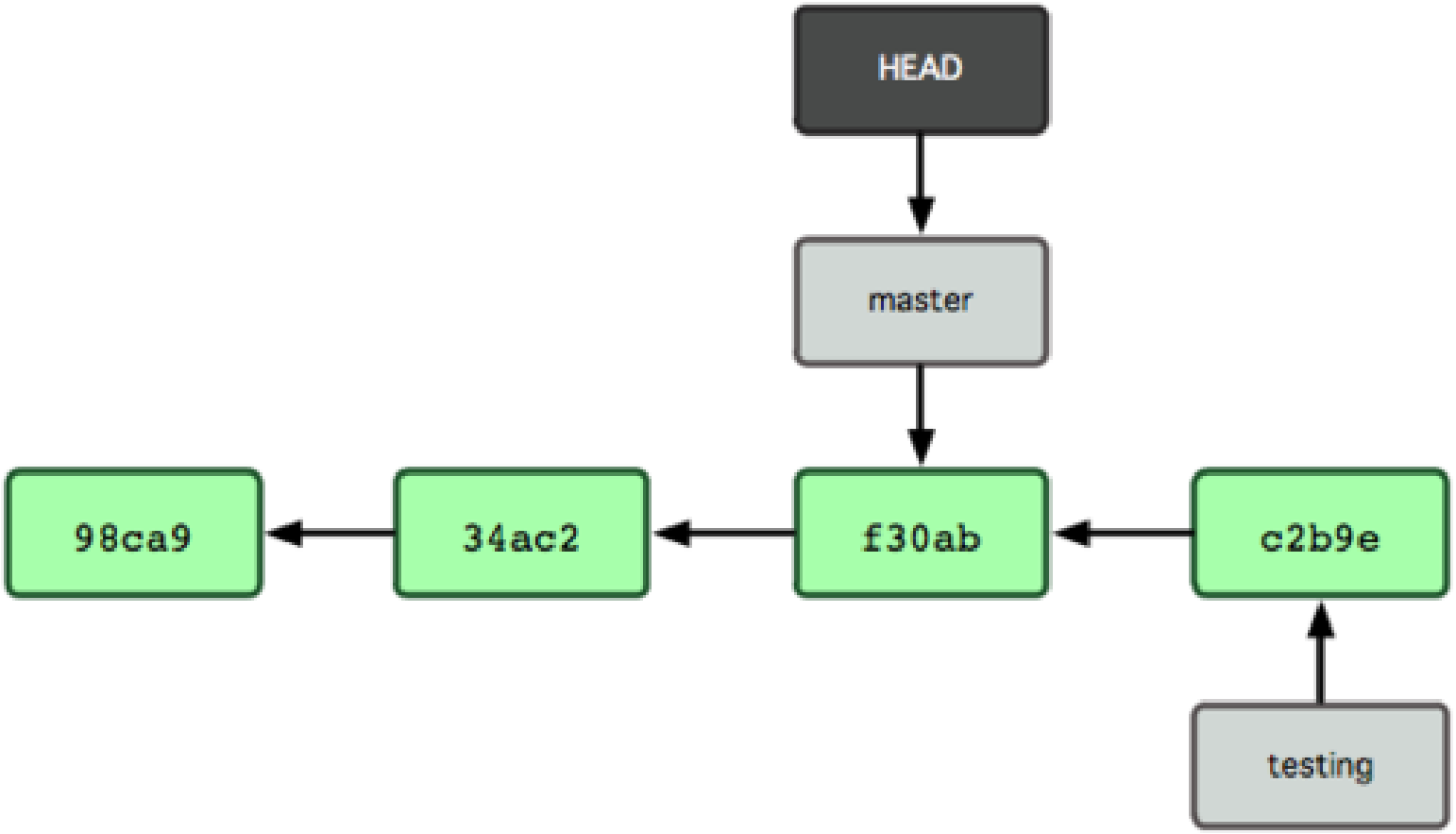
```
kaidu:linux$ git commit -am '...'
```

- nach einem Commit bewegt sich der HEAD und der Branch, auf den dieser zeigt, zum nächsten Commit



Exkurs: Branches

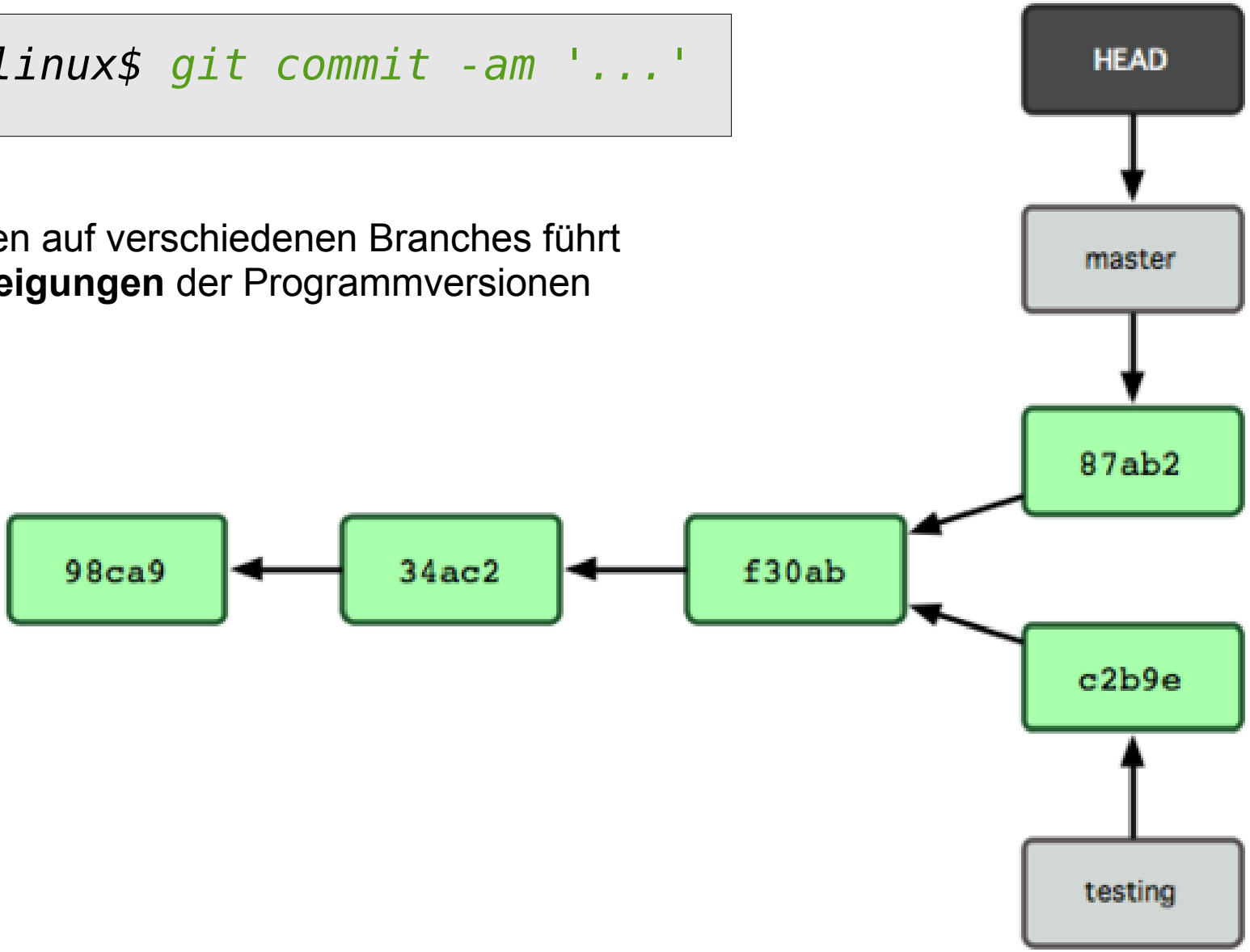
```
kaidu:linux$ git checkout master
```



Exkurs: Branches

```
kaidu:linux$ git commit -am '...'
```

- Commiten auf verschiedenen Branches führt zu **Verzweigungen** der Programmversionen



Exkurs: Branches

```
kaidu:linux$ git branch bugfix
kaidu:linux$ git checkout bugfix
kaidu:linux$ change some files...
kaidu:linux$ git commit -am 'fix a bug in ...'
kaidu:linux$ git checkout master
kaidu:linux$ git merge -d bugfix
```

- Typische Vorgehensweise beim Branchen:
- Änderungen eines bestimmten Features in einem eigenen Branch entwickeln
- Später Änderungen in den Hauptbranch einpflegen: **mergen**!
- Mergen ist exakt dasselbe, was beim **pull** Befehl passiert. Auch hier können Konflikte auftreten, die genauso gelöst werden

Exkurs: Tagging

```
kaidu:linux$ git tag -a v1.0 -m 'release 1.0'
```

- Ein Tag ist prinzipiell ein **konstanter Branch**
- wird zum setzen einer bestimmten feststehenden Versionen genutzt

(Beispiel: Die Version, die ihr uns später abgebt, könnte den Tag 'final' haben)

- Kann danach ähnlich wie ein Branch behandelt werden

```
kaidu:linux$ git checkout v1.0
kaidu:linux$ git tag # listet alle Tags
kaidu:linux$ git push origin v1.0
```

- Achtung: Tag muss explizit per push an den Server übertragen werden

Tipps für git

- http://git-scm.com/ <--- lesen!
- Vermeide Binaries im Repository
  - Ausnahme: Icons für GUIs usw.
- Verwende einfache Pfadnamen
- Achtung: Windows kann nicht

  zwischen Groß/Kleinschreibung unterscheiden)
- Verwende getrennte Ordner für den Sourcecode und die Ausgabe
  - Die Ausgabe gehört nicht ins Repository