git

Datamining und Sequenzanalyse

Markus Fleischauer, Kai Dührkop

Was ist Versionsverwaltung?

Beispiel: Wikipedia

„Versionsverwaltung“ – Bearbeiten

Deine Änderungen werden angezeigt, sobald sie gesichtet wurden. (Hilfe)

Du bearbeitest diese Seite unangemeldet. Wenn du deine Änderung speicherst, wird deine aktuelle IP-Adresse in der Version
Benutzerkonto anlegst, bleibt deine IP-Adresse verborgen.

Speichere hier bitte keine Textversuche ab. Dafür haben wir unsere Spielwiese.

F K ▸ Erweitert ▸ Sonderzeichen ▸ Hilfe

```
Eine '''Versionsverwaltung''' ist ein System, das zur Erfassung von Änderungen an Dokumenten oder Dateien verwendet wir
gesichert und können später wiederhergestellt werden. Versionsverwaltungssysteme werden typischerweise in der Softwaree
bei [[Büroanwendung]]en oder [[Content-Management-System]]en zum Einsatz.

Ein Beispiel ist auch die [[Wikipedia]], hier erzeugt die Software nach jeder Änderung eines Artikels eine neue Version
es sind keine Varianten vorgesehen. Da zu jedem Versionswechsel die grundlegenden Angaben wie Verfasser und Uhrzeit fes
Bedarf – beispielsweise bei versehentlichen Änderungen – kann man zu einer früheren Version zurückkehren.

Die Versionsverwaltung ist eine Form des [[Variantenmanagement]]s; dort sind verschiedene Sprachvarianten oder modal au
'''VCS''' (Version Control System) gebräuchlich.
```

Beispiel: Wikipedia

„Versionsverwaltung“ – Versionsgeschichte

Logbücher dieser Seite anzeigen

In der Versionsgeschichte suchen

bis Jahr: 2014 und Monat: alle Markierungs-Filter: Anwenden

Alte Versionen des Artikels (Hilfe):

- (Aktuell) = Unterschied zur aktuellen Version, (Vorherige) = Unterschied zur vorherigen Version
- Uhrzeit und Datum = Artikel zu dieser Zeit, Benutzername bzw. IP-Adresse des Bearbeiters, K = Kleine Änderung
- (123 Bytes) = Größe der Version; (+543)/(-792) = Änderung der Seitengröße in Bytes gegenüber der vorherigen Version
- Um Unterschiede zwischen zwei bestimmten Versionen zu sehen, die Radiobuttons markieren und auf „Gewählte Versionen vergleichen“ klicken

(neueste | älteste) Zeige (nächste 50 | vorherige 50) (20 | 50 | 100 | 250 | 500)

Gewählte Versionen vergleichen

- (Aktuell | Vorherige) 17:27, 21. Okt. 2014 Kghbln (Diskussion | Beiträge) . . (12.700 Bytes) (-15) . . *(Die letzte Textänderung von 79.221.2.219 wurde verworfen und die*
  [automatisch gesichtet]
- (Aktuell | Vorherige) 10:39, 7. Okt. 2014 79.221.2.219 (Diskussion) . . (12.715 Bytes) (+15) . . (rückgängig)
- (Aktuell | Vorherige) 16:05, 2. Okt. 2014 YMS (Diskussion | Beiträge) **K** . . (12.700 Bytes) (-2) . . *(Änderungen von 171.18.29.130 (Diskussion) auf die letzte Version vo*
- (Aktuell | Vorherige) 15:54, 2. Okt. 2014 171.18.29.130 (Diskussion) . . (12.702 Bytes) (+2) . . *(→Hauptaufgaben)* (rückgängig)
- (Aktuell | Vorherige) 13:29, 21. Jul. 2014 194.8.218.60 (Diskussion) . . (12.700 Bytes) (+13) . . *(→Weblinks: toten Link mit Alternative ersetzt)* (rückgängig) [gesichtet vo
- (Aktuell | Vorherige) 09:40, 23. Mai 2014 MRosetree (Diskussion | Beiträge) **K** . . (12.687 Bytes) (-4) . . *(→Beispiele: Verlinkung auf Subversion Artikel, nicht auf Weite*
- (Aktuell | Vorherige) 09:13, 8. Mai 2014 195.124.7.210 (Diskussion) . . (12.691 Bytes) (+42) . . *(Link auf azyklischer Graph von der Weiterleitung direkt auf die relevan*
- (Aktuell | Vorherige) 22:36, 6. Feb. 2014 Elektrolurch (Diskussion | Beiträge) **K** . . (12.649 Bytes) (+9) . . *(→Verteilte Versionsverwaltung)* (rückgängig) [automatisch gesi
- (Aktuell | Vorherige) 22:35, 6. Feb. 2014 Elektrolurch (Diskussion | Beiträge) **K** . . (12.640 Bytes) (+417) . . *(→Verteilte Versionsverwaltung: Hinweis auf übliche Praxis*
- (Aktuell | Vorherige) 13:43, 30. Jan. 2014 193.8.177.17 (Diskussion) . . (12.223 Bytes) (+39) . . (rückgängig) [gesichtet von Mfb]

Beispiel: Wikipedia

„Versionsverwaltung“ – Versionsunterschied

[gesichtete Version]	[gesichtete Version]
Version vom 29. Oktober 2013, 11:48 Uhr (Bearbeiten)	**Aktuelle Version vom 21. Oktober 2014, 17:27 Uhr (Bearbeiten) (rückgängig)**
Autorabe71 (Diskussion | Beiträge)	Kghbln (Diskussion | Beiträge)
(→Beispiele)	*(Die letzte Textänderung von 79.221.2.219 wurde verworfen und die Version 134544628 von YMS wiederhergestellt.)*
← Zum vorherigen Versionsunterschied	(Markierung: HHVM)

(12 dazwischenliegende Versionen von 9 Benutzern werden nicht angezeigt)

Zeile 1:	**Zeile 1:**
− Eine '''Versionsverwaltung''' ist ein System, das zur Erfassung von Änderungen an Dokumenten oder Dateien verwendet wird. Alle Versionen werden in einem Archiv mit [[Zeitstempel]] und Benutzerkennung gesichert und können später wiederhergestellt werden. Versionsverwaltungssysteme werden typischerweise in der Softwareentwicklung eingesetzt um [[Quelltext]]e zu verwalten. Versionsverwaltung kommt auch bei [[Büroanwendung]]en oder [[Content-Management-System]]en zum Einsatz.	+ Eine '''Versionsverwaltung''' ist ein System, das zur Erfassung von Änderungen an Dokumenten oder Dateien verwendet wird. Alle Versionen werden in einem Archiv mit [[Zeitstempel]] und Benutzerkennung gesichert und können später wiederhergestellt werden. Versionsverwaltungssysteme werden typischerweise in der Softwareentwicklung eingesetzt, um [[Quelltext]]e zu verwalten. Versionsverwaltung kommt auch bei [[Büroanwendung]]en oder [[Content-Management-System]]en zum Einsatz.
Ein Beispiel ist auch die [[Wikipedia]], hier erzeugt die Software nach jeder Änderung eines Artikels eine neue Version. Alle Versionen bilden in Wikipedia eine Kette, in der die letzte Version gültig ist; es sind keine Varianten vorgesehen. Da zu jedem Versionswechsel die grundlegenden Angaben wie Verfasser und Uhrzeit festgehalten werden, kann genau nachvollzogen werden, wer wann was geändert hat. Bei Bedarf – beispielsweise bei versehentlichen Änderungen – kann man zu einer früheren Version zurückkehren.	Ein Beispiel ist auch die [[Wikipedia]], hier erzeugt die Software nach jeder Änderung eines Artikels eine neue Version. Alle Versionen bilden in Wikipedia eine Kette, in der die letzte Version gültig ist; es sind keine Varianten vorgesehen. Da zu jedem Versionswechsel die grundlegenden Angaben wie Verfasser und Uhrzeit festgehalten werden, kann genau nachvollzogen werden, wer wann was geändert hat. Bei Bedarf – beispielsweise bei versehentlichen Änderungen – kann man zu einer früheren Version zurückkehren.
Zeile 13:	**Zeile 13:**
== Terminologie ==	== Terminologie ==
{{Anker|Branch}}Ein "Branch", zu Deutsch Zweig, ist eine Abspaltung von einer anderen Version, so dass unterschiedliche Versionen parallel weiterentwickelt werden können. Änderungen können dabei von einem Branch auch in einem anderen wieder einfließen, das wird dann als	{{Anker|Branch}}Ein "Branch", zu Deutsch Zweig, ist eine Abspaltung von einer anderen Version, so dass unterschiedliche Versionen parallel weiterentwickelt werden können. Änderungen können dabei von einem Branch auch wieder in einen anderen einfließen, das wird dann als

Was ist Versionsverwaltung?

- ermöglicht mehreren Personen gleichzeitig und unabhängig voneinander an einem Dokument zu arbeiten

- Gleichzeitige Änderungen werden zusammengeführt (**merge**)

- führt eine Versionsgeschichte, in der alle Änderungen am Dokument aufgeführt sind.

- Frühere Dokumentversionen können eingesehen, verglichen und wiederhergestellt werden

Was ist Versionsverwaltung?

Gemeinsam an Sourcecode arbeiten: Fileserver?

N F S

NETWORK FILE SYSTEM

Was ist Versionsverwaltung?

Gemeinsam an Sourcecode arbeiten: Fileserver?

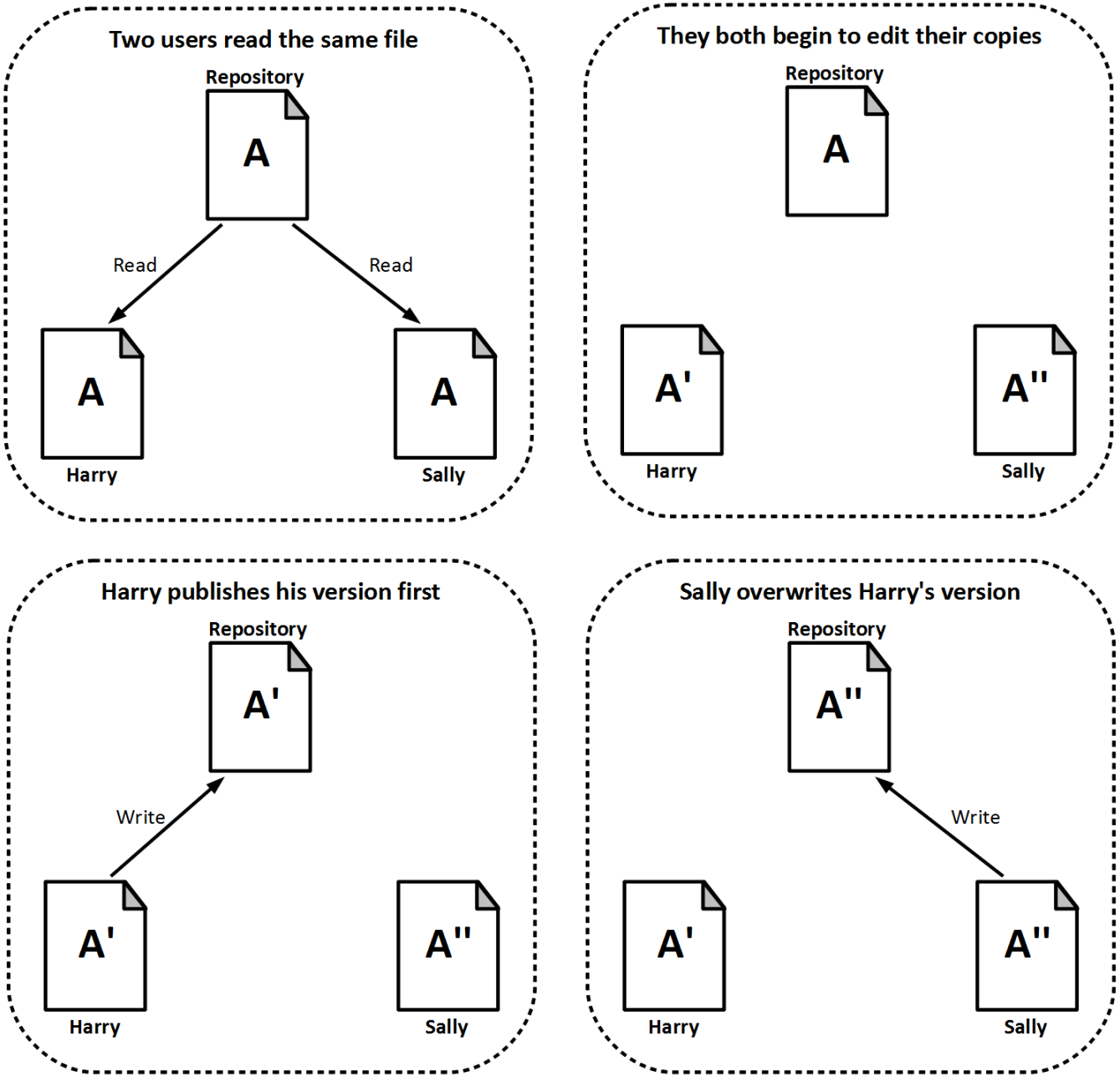
samba

ssh

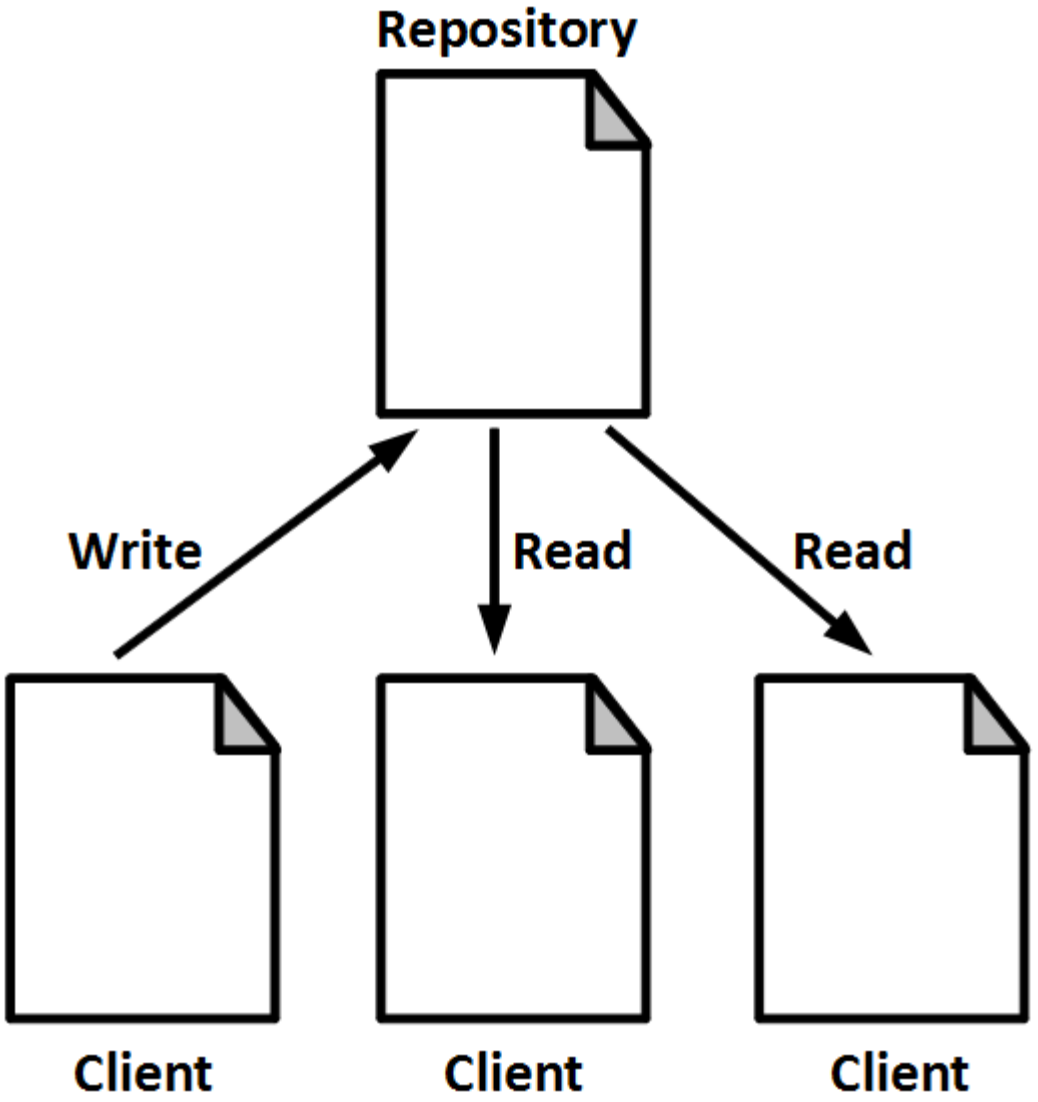
NFS

NETWORK FILE SYSTEM

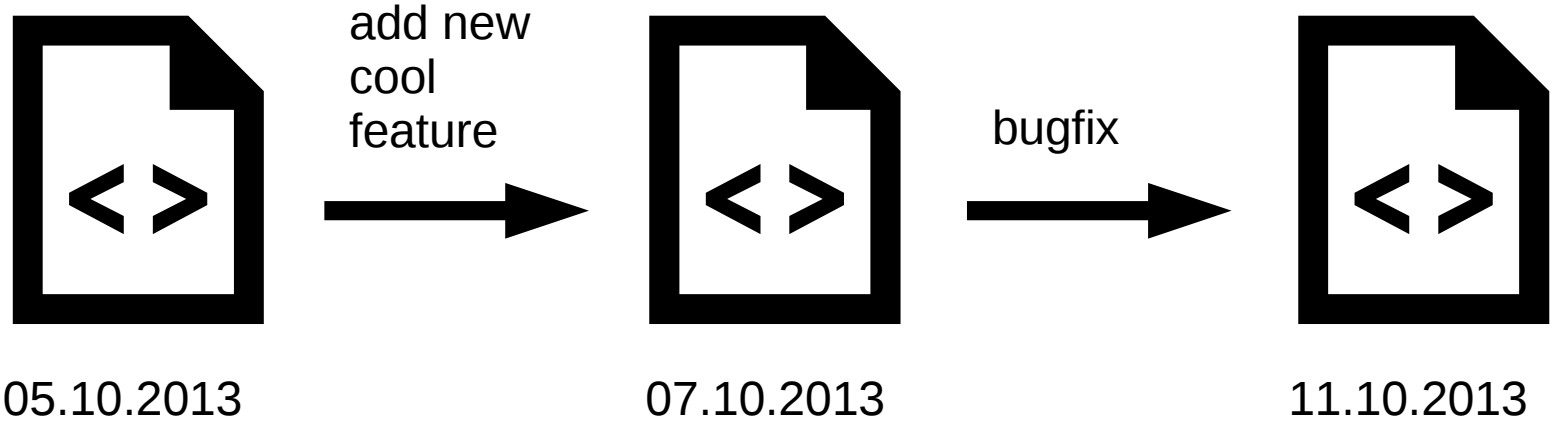
Das Problem verteilter Dateizugriffe



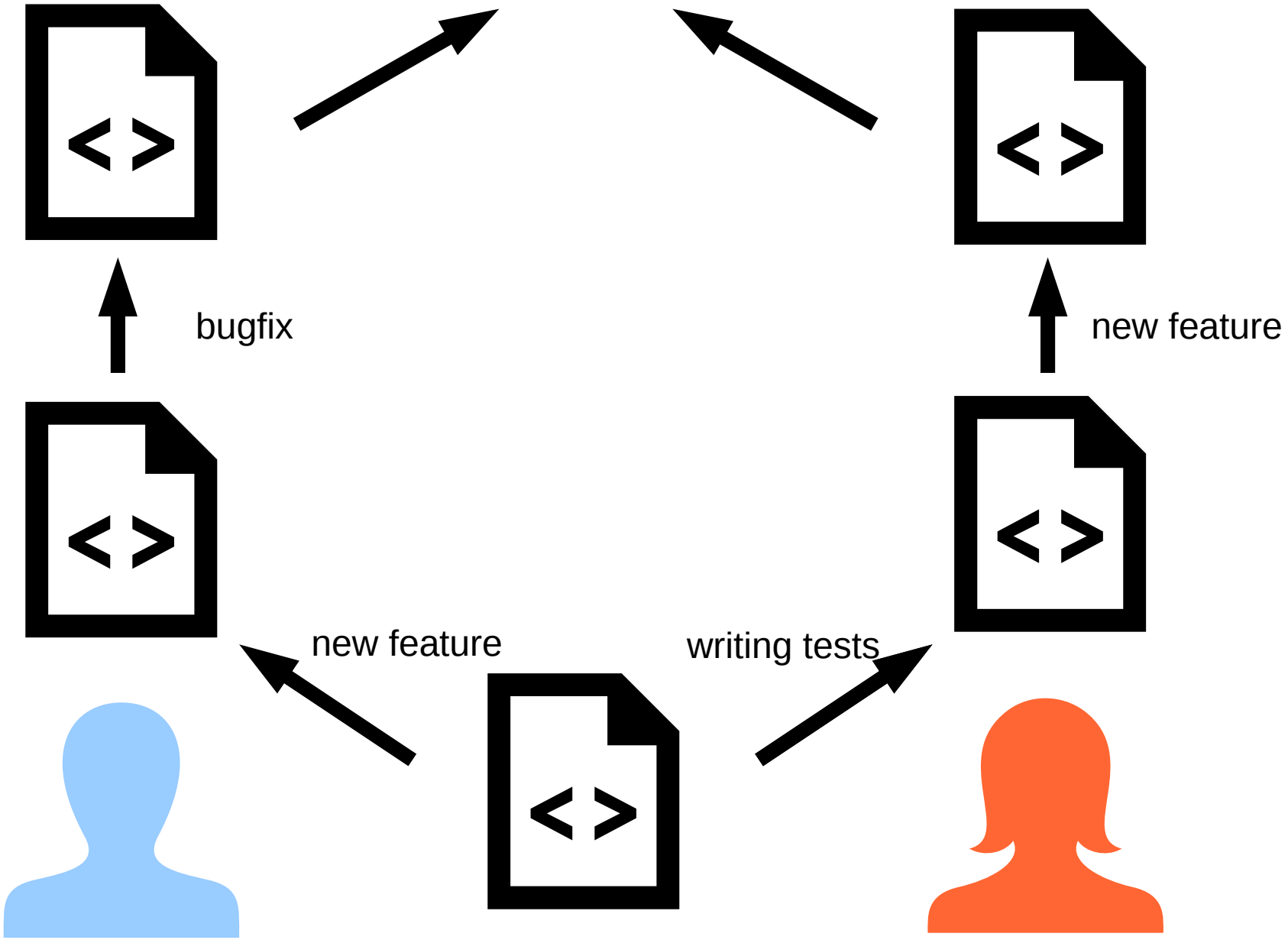
Versionsverwaltung



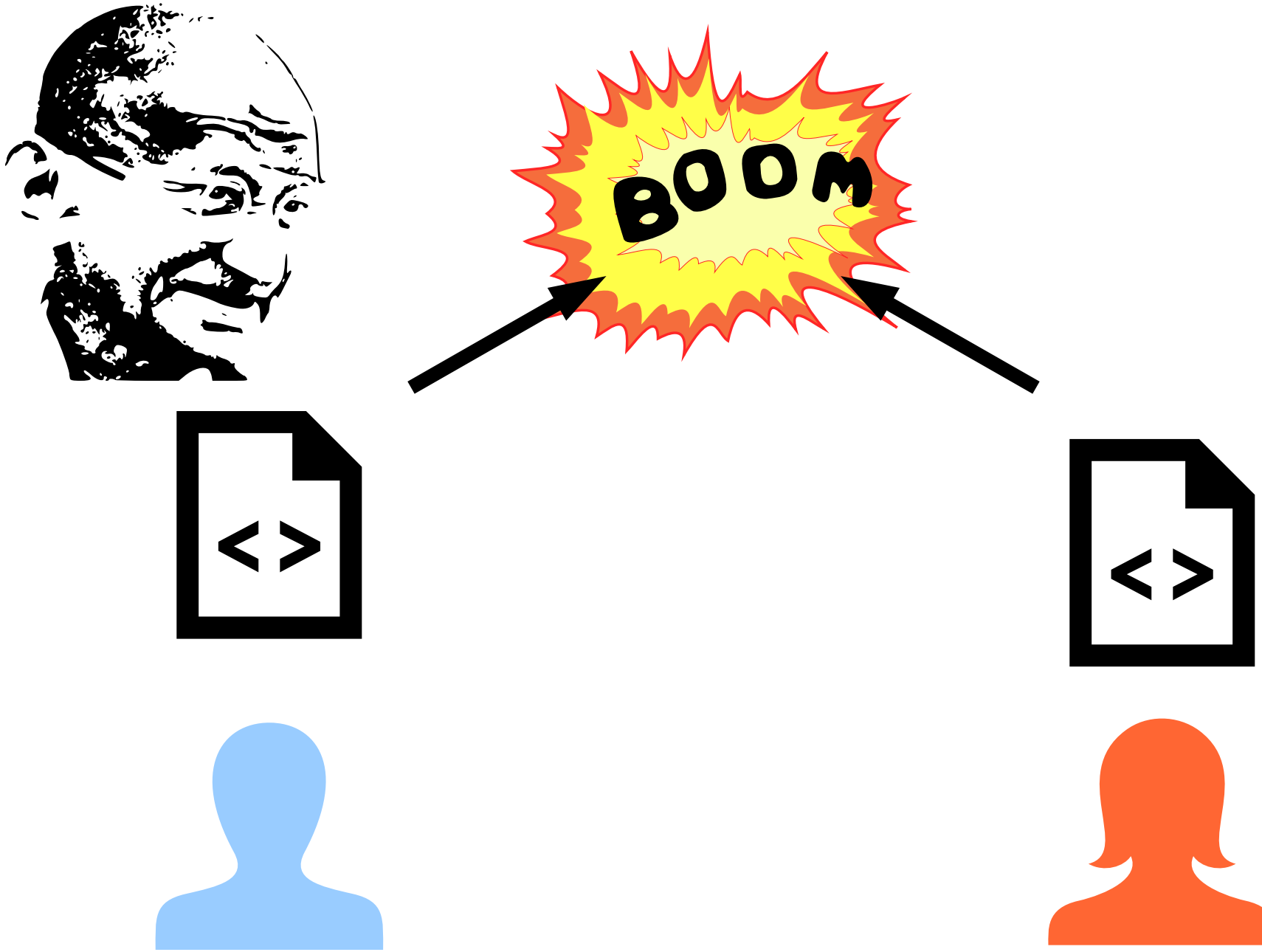
Versionierung...über die Zeit



Versionierung...über mehrere Entwickler

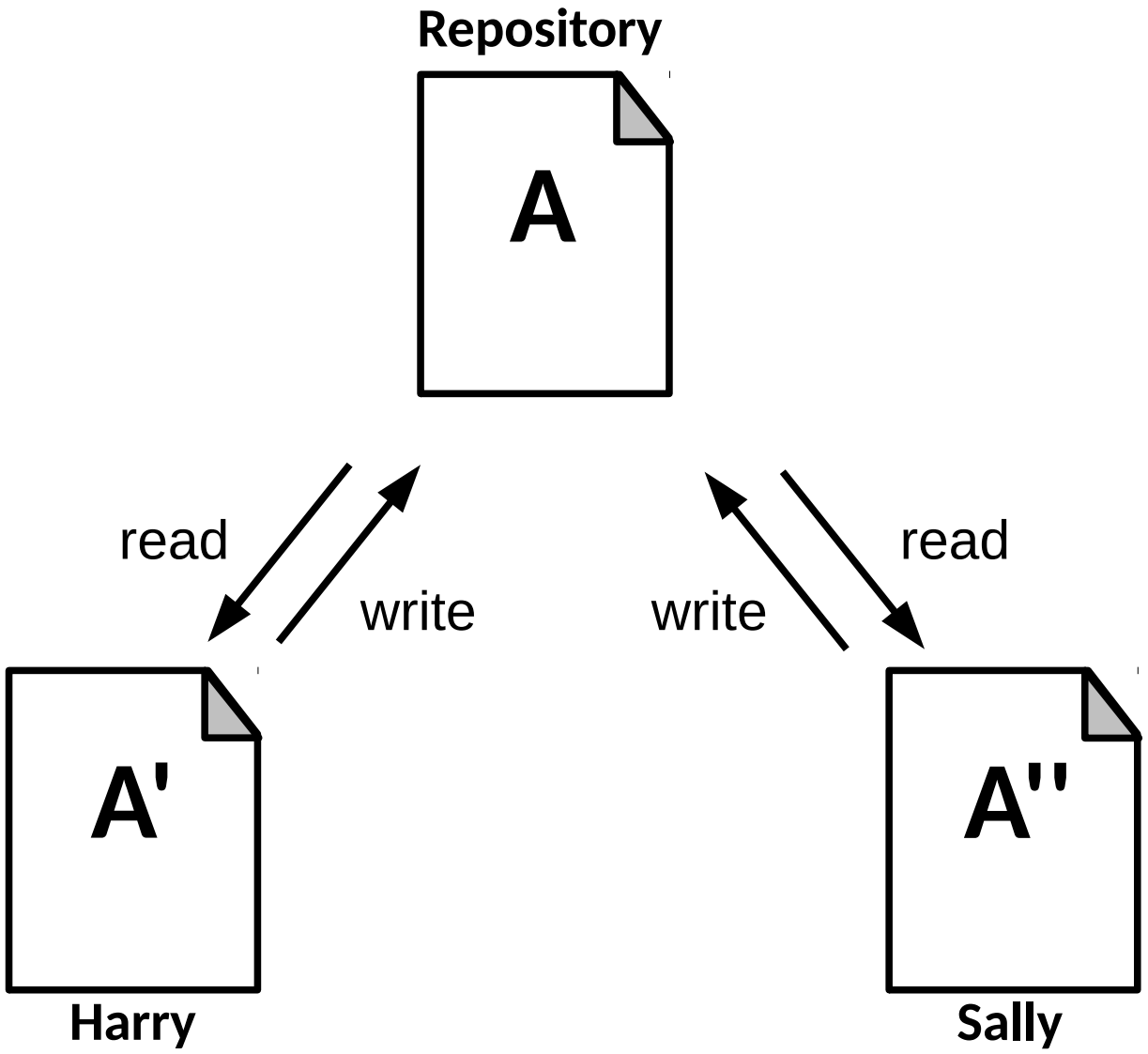


Konfliktlösung

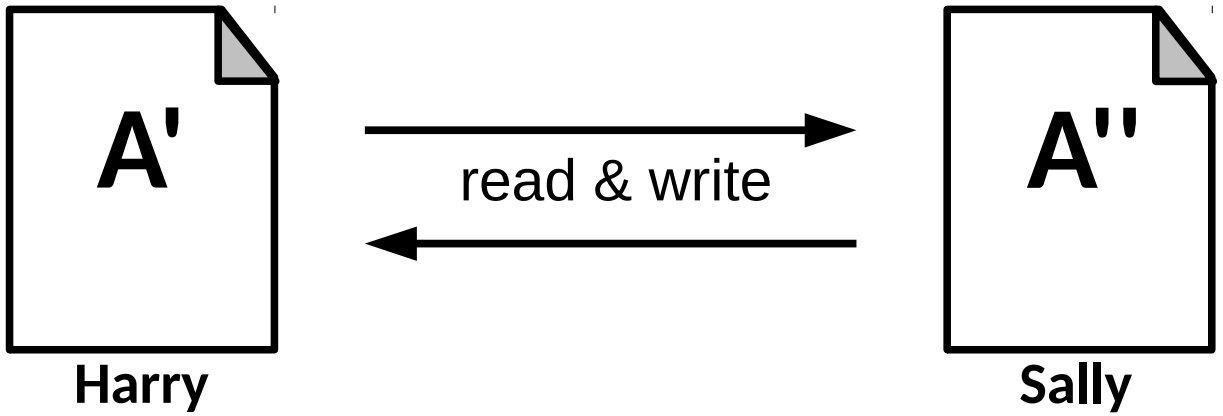


Versionskontrollsysteme

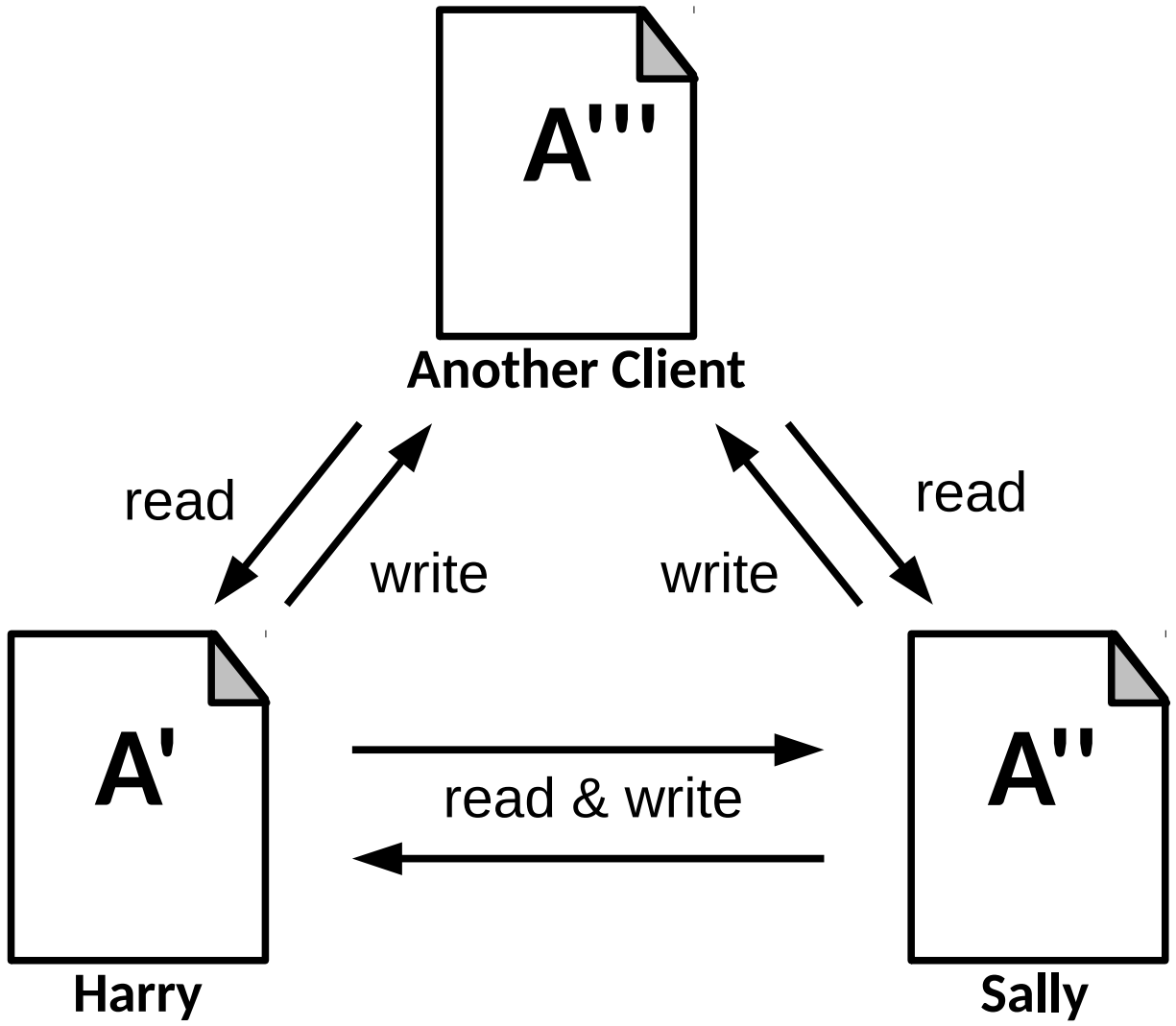
SVN - eine zentrale Versionsverwaltung



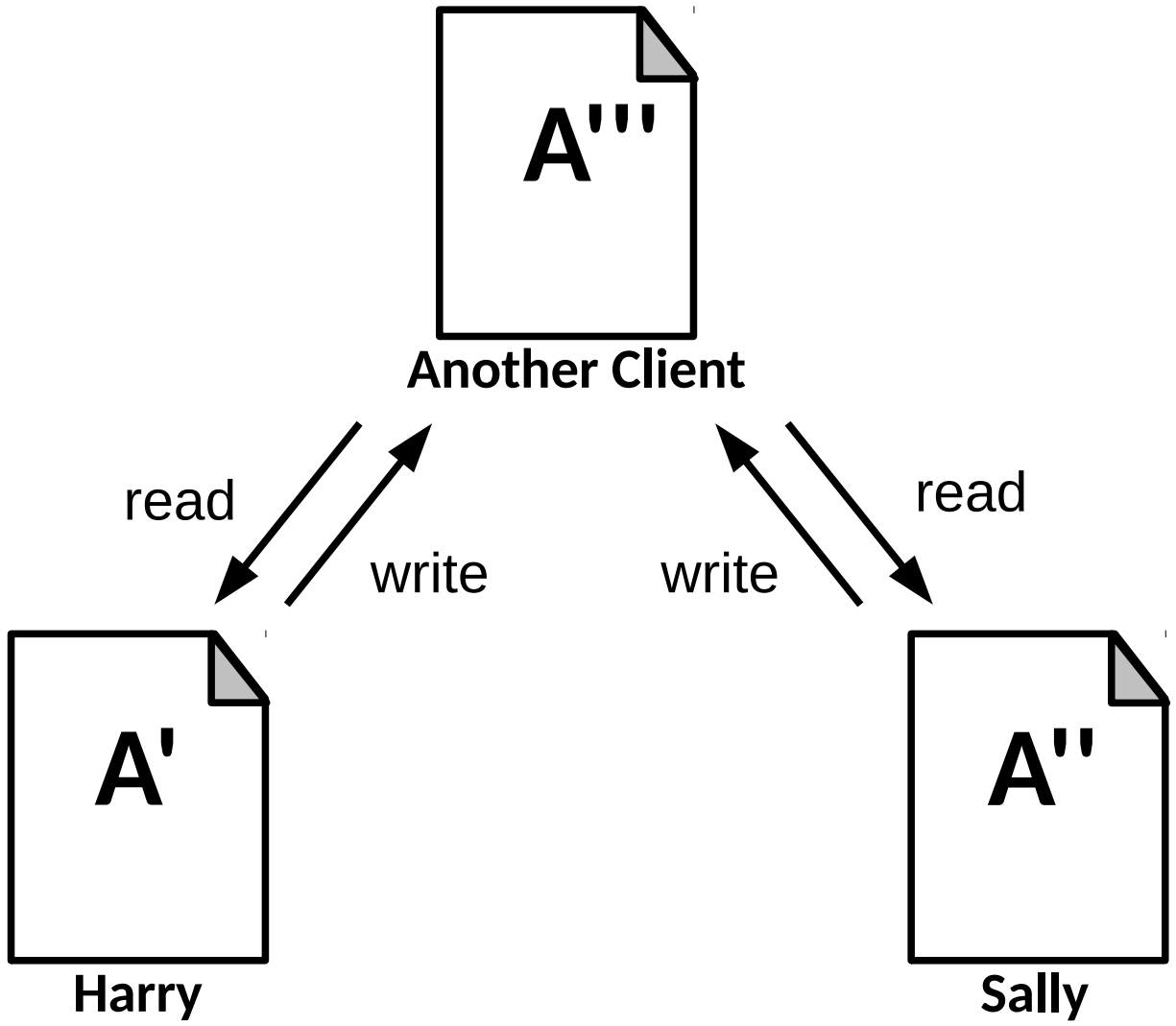
git - eine dezentrale Versionsverwaltung



git - eine dezentrale Versionsverwaltung



git - eine dezentrale Versionsverwaltung



git - eine dezentrale Versionsverwaltung

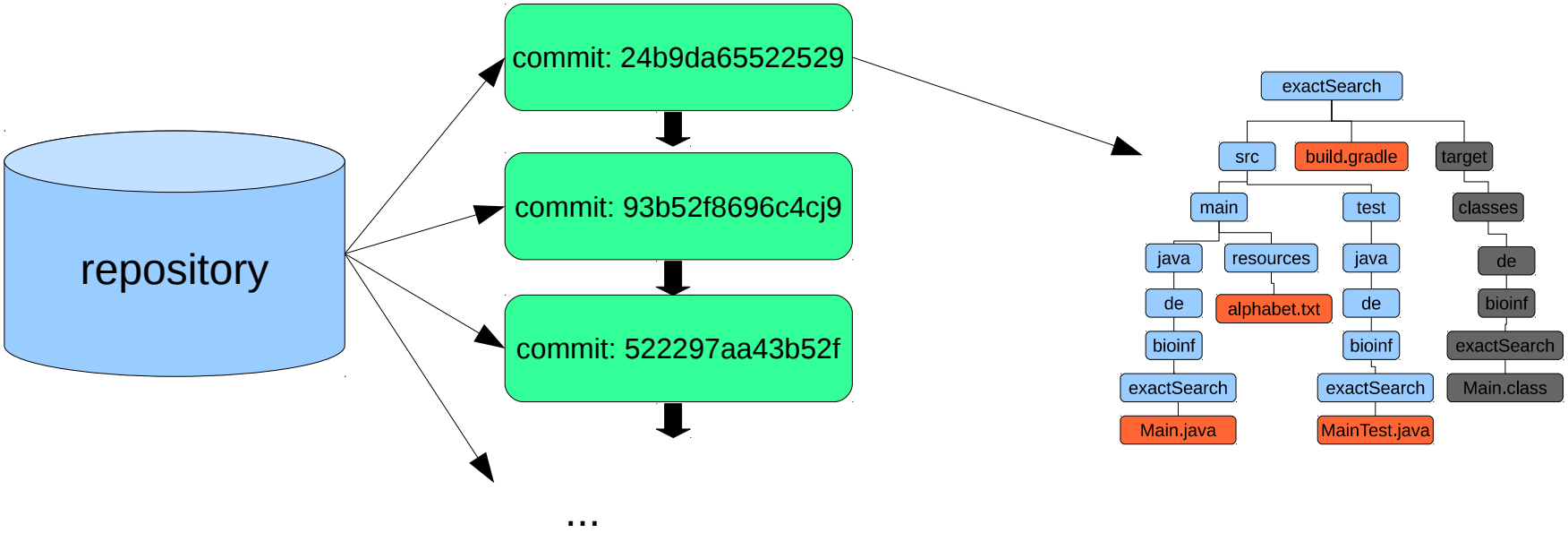
- Alle Dokumente einer Versionsverwaltung liegen in einem **Repository**

- Jeder Nutzer darf seine eigene Kopie eines Repositories haben

- Nutzer können untereinander Änderungen an ihren Dokumenten austauschen

- In der Praxis ist es sinnvoll, ein (oder mehrere) zentrales Repository zu haben, in das jeder seine Änderungen einpflegt

Grundlegende Begriffe



- ein **repository** ist ein versioniertes Projekt
- enthält alle commits

- ein **commit** ist eine bestimmte Version des Projekts
- nutzt einen SHA-1 Hash-Key als Identifier

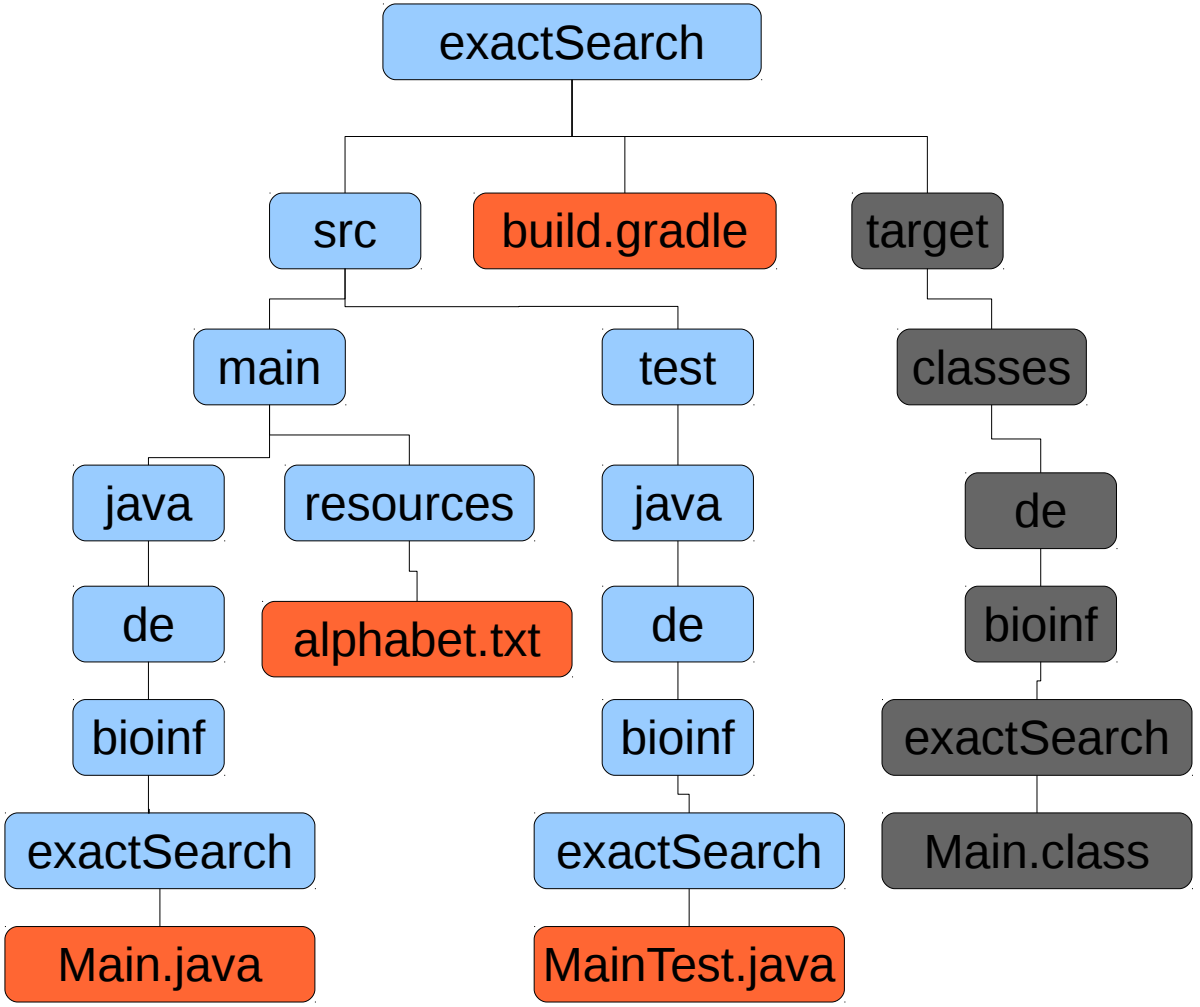
- jeder commit hat einen snapshot
- ein **snapshot** ist ein Verzeichnisbaum mit versionierten Dateien

Working directory

- Ein commit darf „aktiv“ (checked out) sein

- Check out bedeutet: der dazugehörige Snapshot wird aus der git-Datenbank extrahiert und in euer Verzeichnis kopiert

- Auf diesem „working directory“ könnt ihr arbeiten: Dateien hinzufügen, löschen, bearbeiten



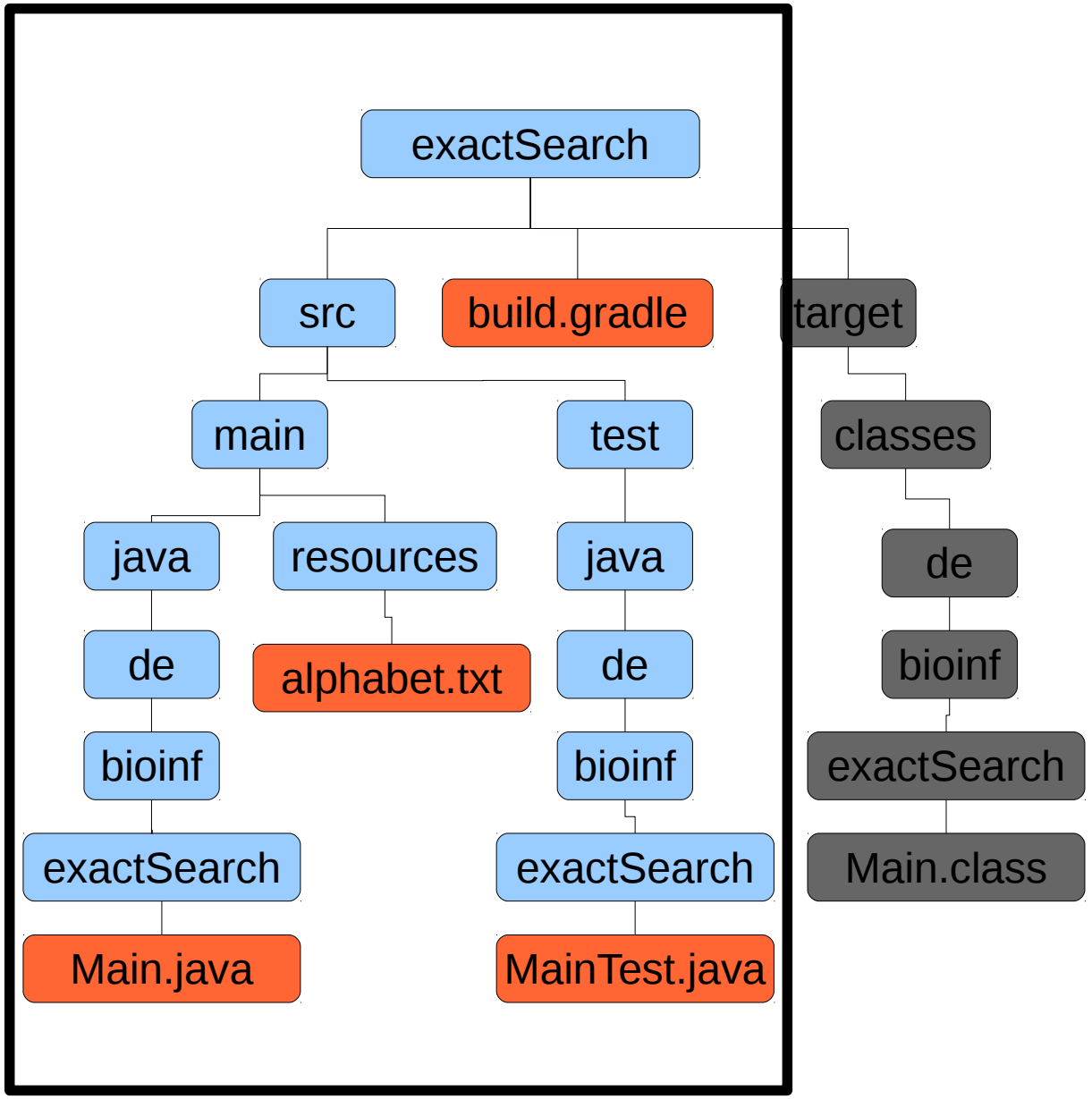
Grundlegende Begriffe

Snapshot:

- die Gesamtheit aller versionierten Dateien innerhalb einer Version

- Nur Dateien werden versioniert. Verzeichnispfade existieren lediglich als Eigenschaften von Dateien

- Entsprechend ist es nicht möglich leere Verzeichnisse zu versionieren



Grundlegende Begriffe

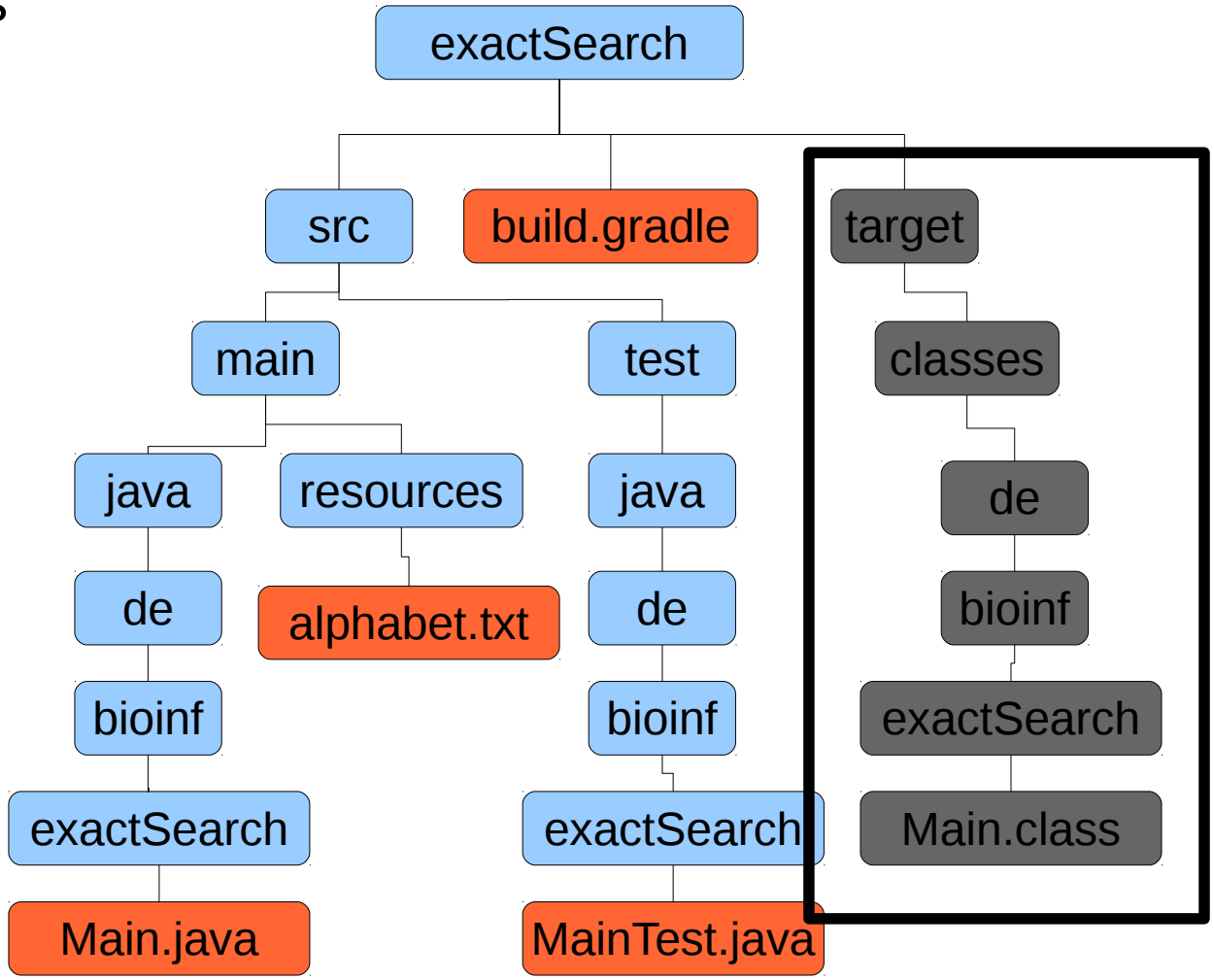
Untracked files:

• Ihr dürft auch Dateien in das Working directory kopieren, die nicht versioniert werden sollen

• Solche Dateien nennt man „untracked“

• alles was **Text** ist und **manuell** geändert wird sollte versioniert werden

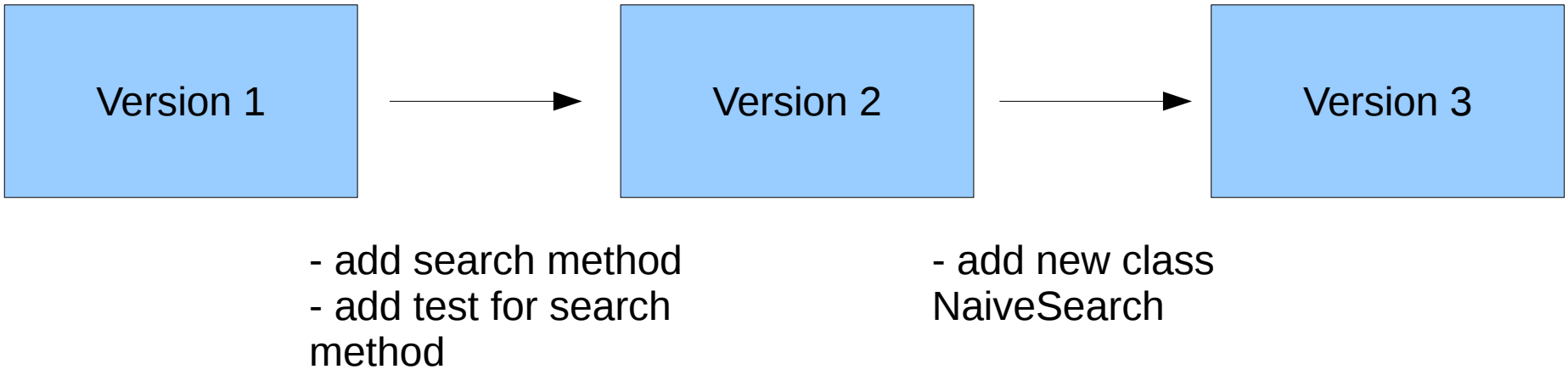
• alles was **binary** data ist (Bilder, class files, jar files ...) oder **automatisch** generiert wird (.iml files eurer IDE, javadoc, test reports) gehört NICHT ins repository



Grundlegende Begriffe

Commit:

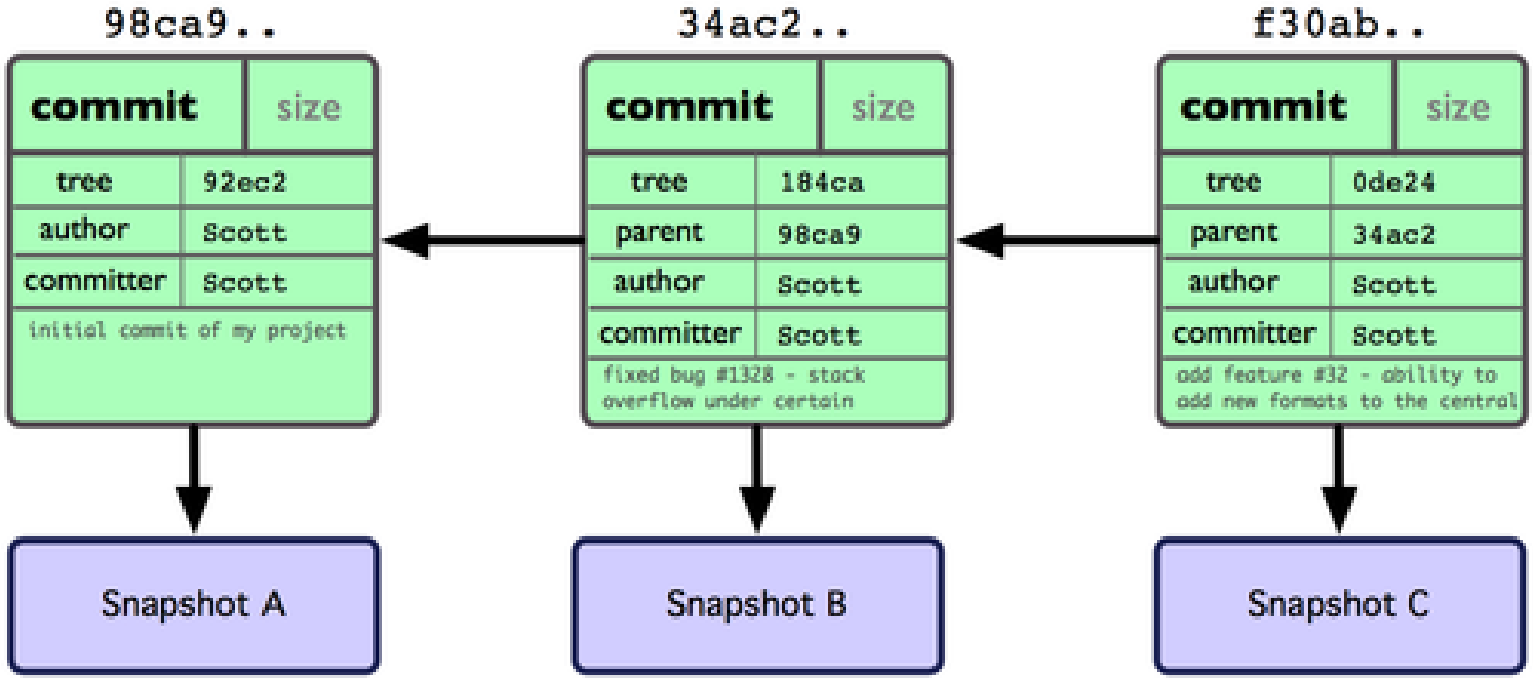
- Eine Version eures repositories



Grundlegende Begriffe

Commit:

- Eine Version eures repositories
- Enthält den Snapshot, sowie Metainformationen (Autor, Zeitstempel, Hash, **Log-Message** ...)



Grundlegende Begriffe

Commit:

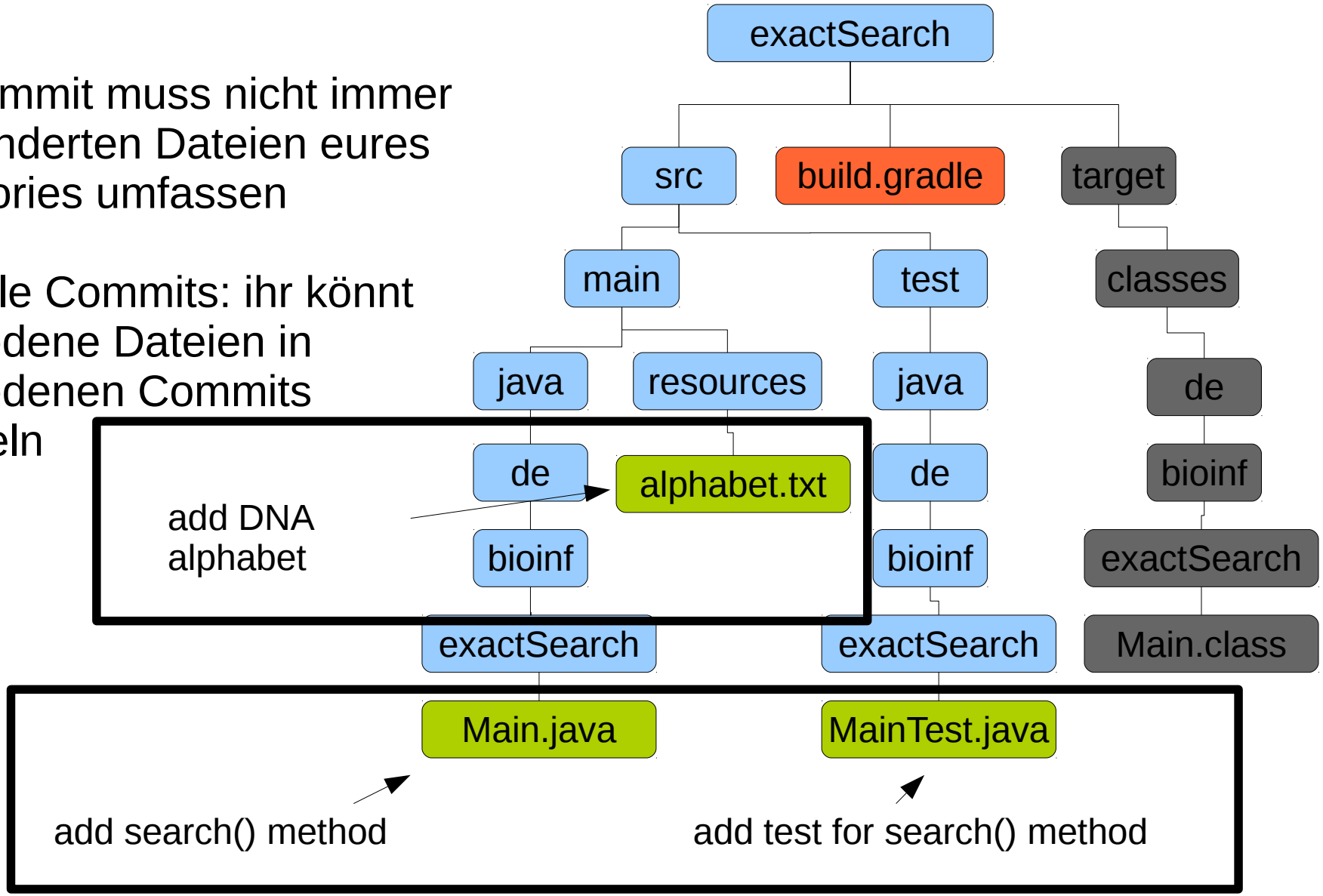
- IHR entscheidet wann ihr einen neuen Commit anlegt
- jeder Commit erzeugt einen neuen Snapshot
- ihr könnt jederzeit einen früheren Snapshot betrachten oder wiederherstellen
- Insbesondere könnt ihr einen früheren commit auschecken um Änderungen rückgängig zu machen
- Änderungen innerhalb eines Commits sind dagegen nicht mehr einsehbar

Grundlegende Begriffe

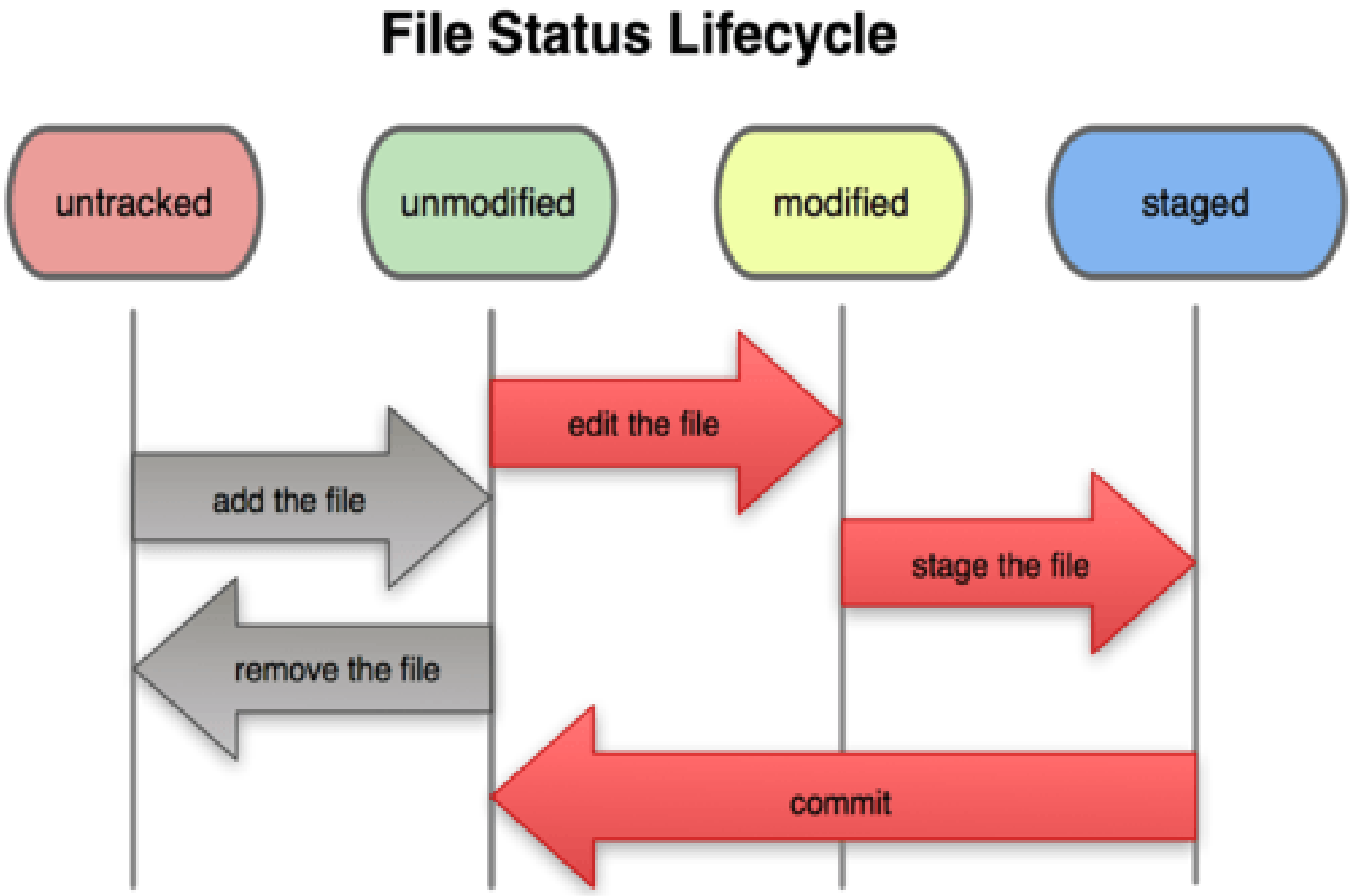
Commit:

- Ein Commit muss nicht immer alle geänderten Dateien eures Repositories umfassen

- parallele Commits: ihr könnt verschiedene Dateien in verschiedenen Commits abhandeln



Mögliche Zustände einer Datei im Repository



Grundlegende Begriffe

Commit:

• ein Commit sollte immer nur Änderungen umfassen die **logisch** zusammengehören

• Jeder Bugfix sollte z.B. ein eigener Commit sein

• ein Commit beachtet nur Änderungen an Dateien die **staged** sind

• für parallele Commits: Einfach nacheinander Dateien stagen, commiten, dann wiederholen (geht mit IDE sehr einfach und bequem)

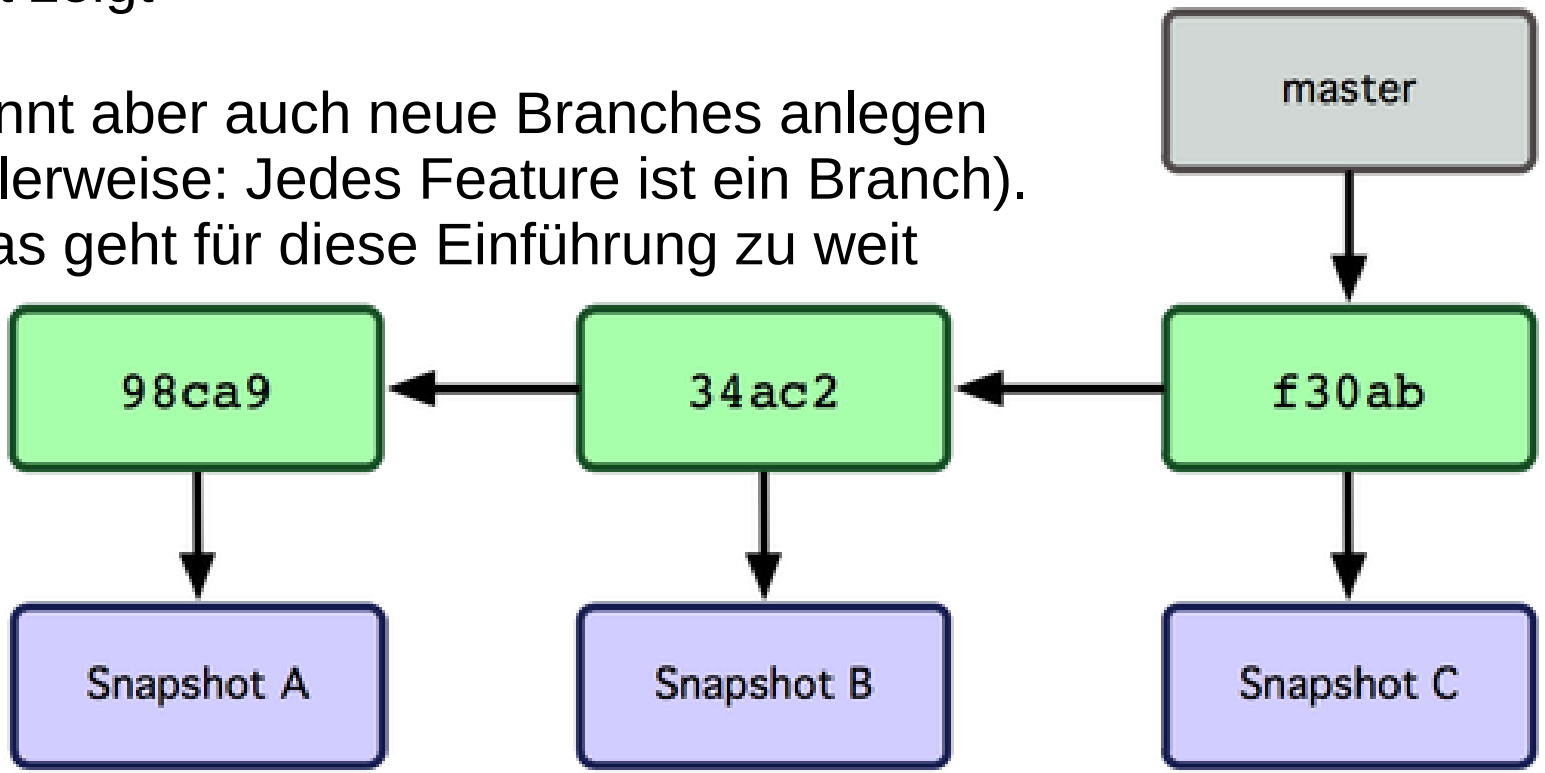
Grundlegende Begriffe

Branch:

• ein Branch ist ein Zeiger auf einen Commit

• standardmäßig gibt es einen **master** Branch der, soweit ihr nichts anderes verfügt, auf den letzten Commit zeigt

• ihr könnt aber auch neue Branches anlegen (normalerweise: Jedes Feature ist ein Branch). Aber das geht für diese Einführung zu weit



Grundlegende Begriffe

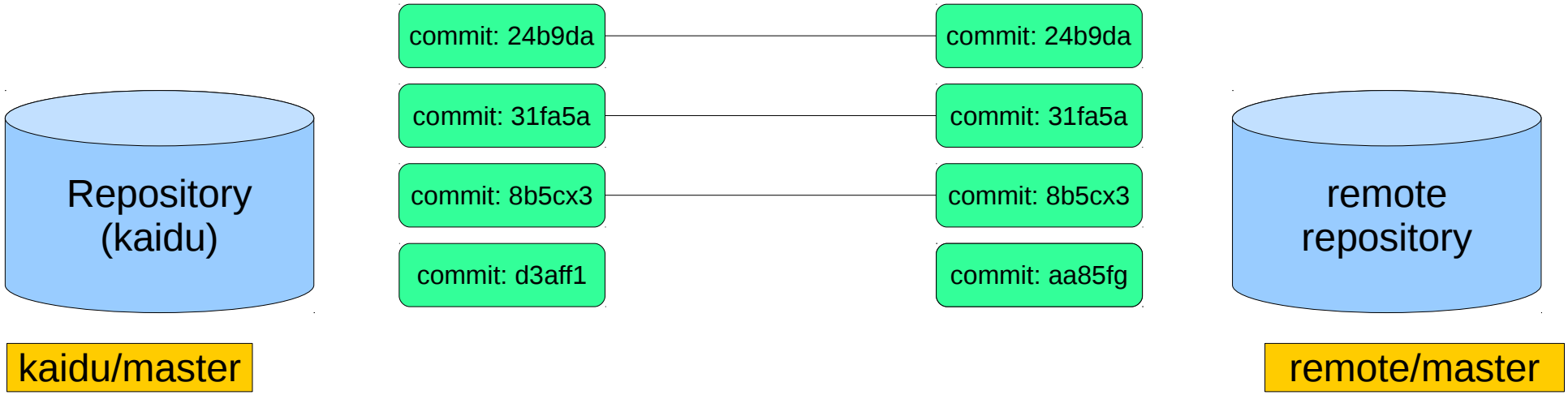
Remote Repository:

• da git dezentral ist, kann es mehrere Kopien eines Repositories geben

• jede Kopie des Repositories ist ein **remote** Repository

• für uns wichtig: Unser zentrales Repository ist ein remote Repository, unser lokales ist unser Arbeitsrepository

Remote repository

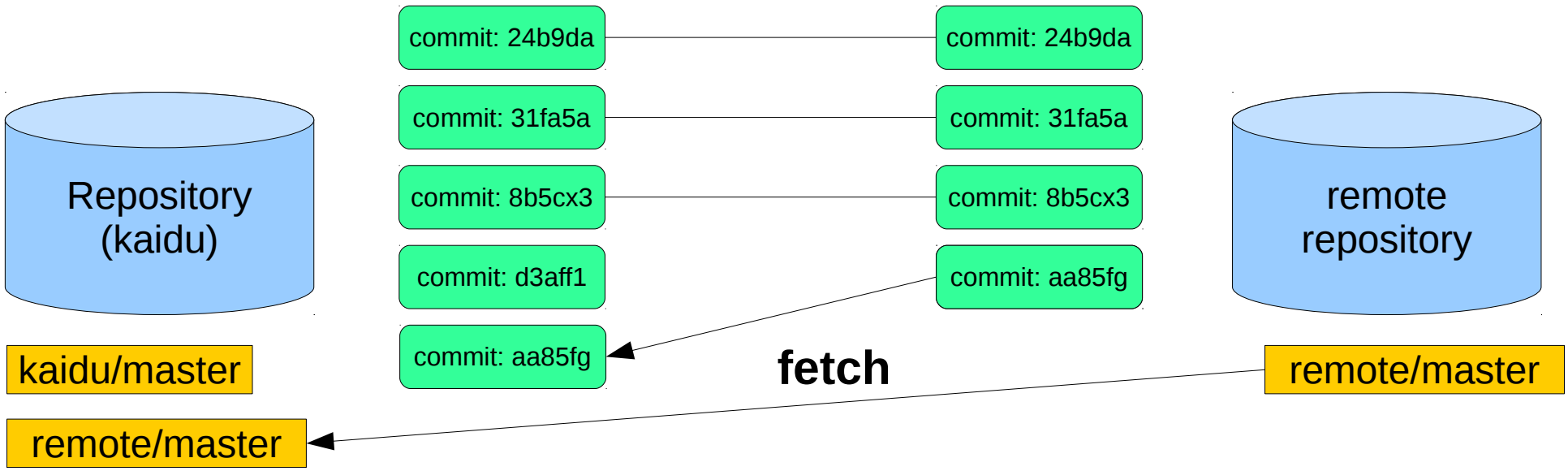


- Jedes Repository hat seine eigenen Commits und Branches

- Wir synchronisieren unser repository mit dem remote repository, in dem wir alle Commits und Branches rüberkopieren/runterladen **(fetch)**

- git erkennt automatisch, wenn zwei Commits identisch sind. Bei Branches hingegen unterscheidet git immer zwischen den lokalen branches und den remote branches

Remote repository

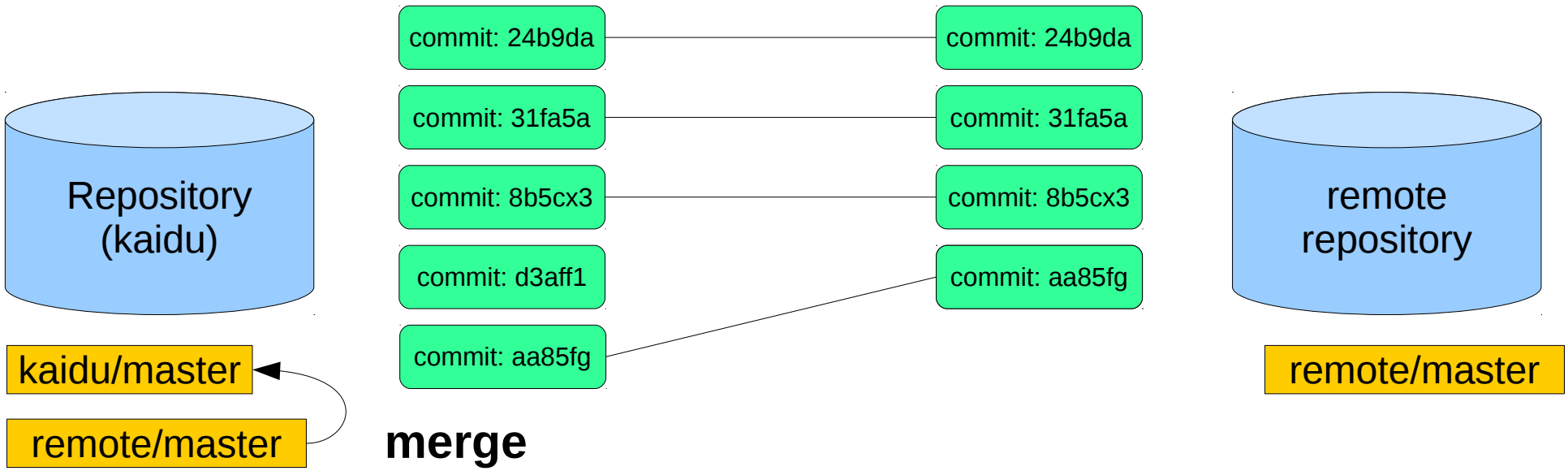


- Jedes Repository hat seine eigenen Commits und Branches

- Wir synchronisieren unser repository mit dem remote repository, in dem wir alle Commits und Branches rüberkopieren/runterladen **(fetch)**

- git erkennt automatisch, wenn zwei Commits identisch sind. Bei Branches hingegen unterscheidet git immer zwischen den lokalen branches und den remote branches

Remote repository

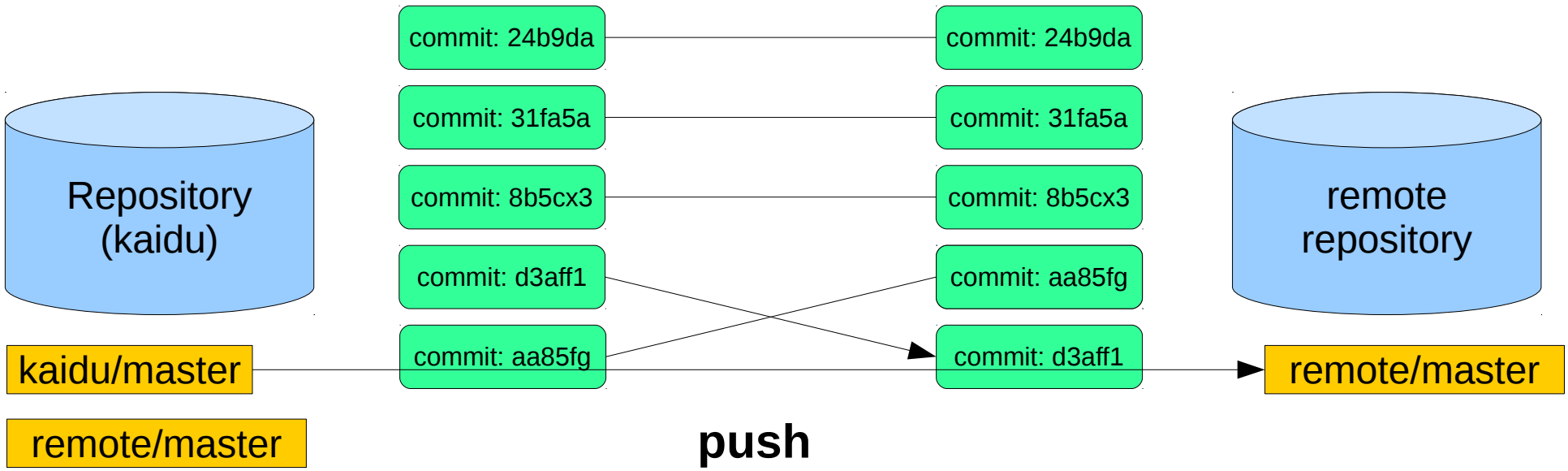


- Das Zusammenführen zweier branches nennt man merge

- Nach dem fetch kann der remote branch mit dem eigenen zusammengeführt werden

- Da fetch und merge so oft gebraucht werden, gibt es den **pull** Befehl, der beides nacheinander ausführt

Remote repository



- Andersherum können lokale Commits und Branches in das remote repository kopiert werden

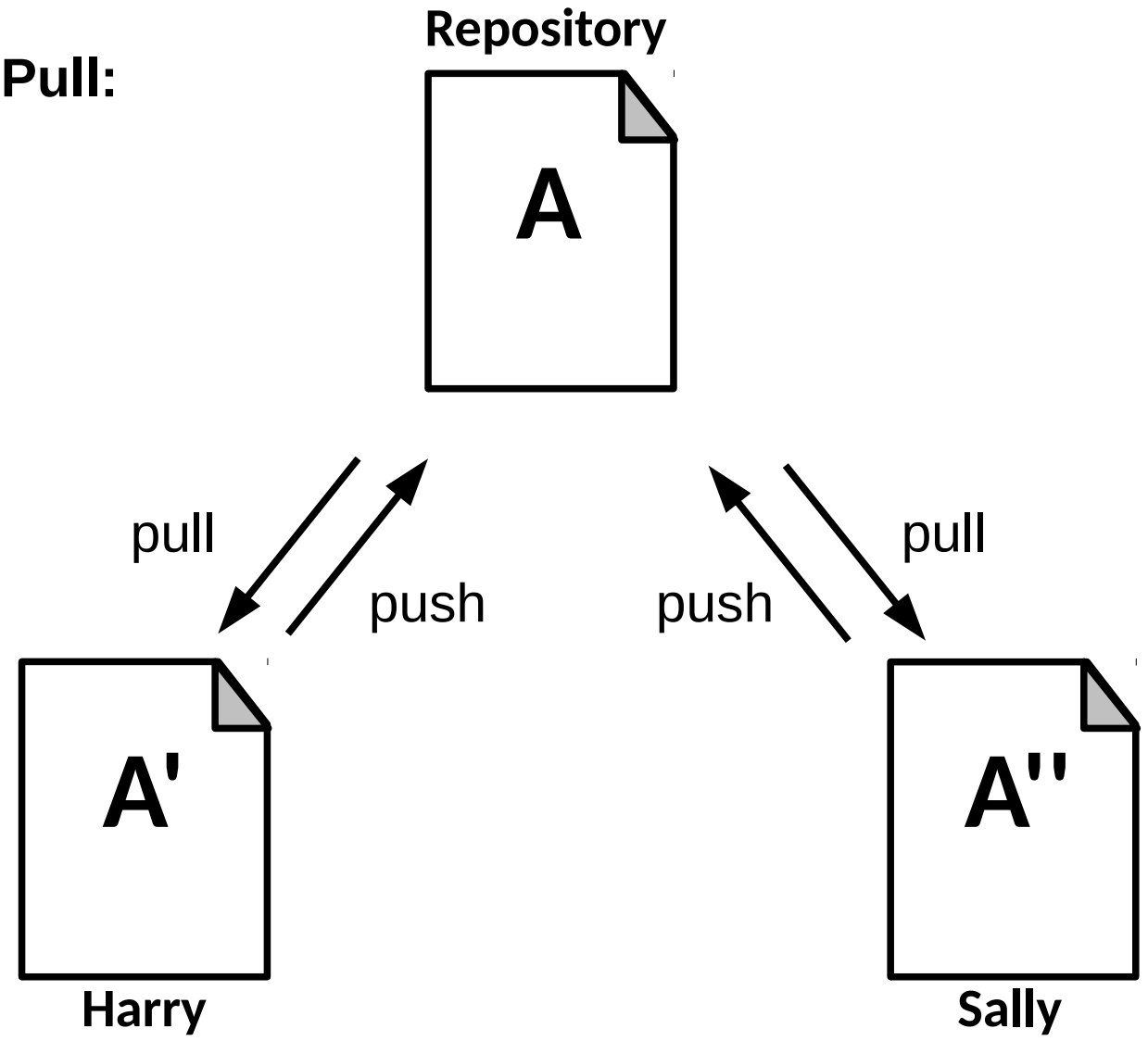
- Branches werden dabei auf remote Seite geupdatet

Remote repository

Push und Pull:

- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch
- wird alles verständlicher wenn wir es an Beispielen zeigen

Push und Pull:



Grundlegende Begriffe

- Repositories (lokal, remote)
- Snapshots
- Commits
- Branches

git auf der Konsole

- jeder sollte die grundlegenden Konsolenbefehle für git kennen
- in der Anwendung könnt ihr git in der Regel von eurer IDE aus bedienen
- manches geht aber über die Konsole einfacher

Einmalig: Identifizieren

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

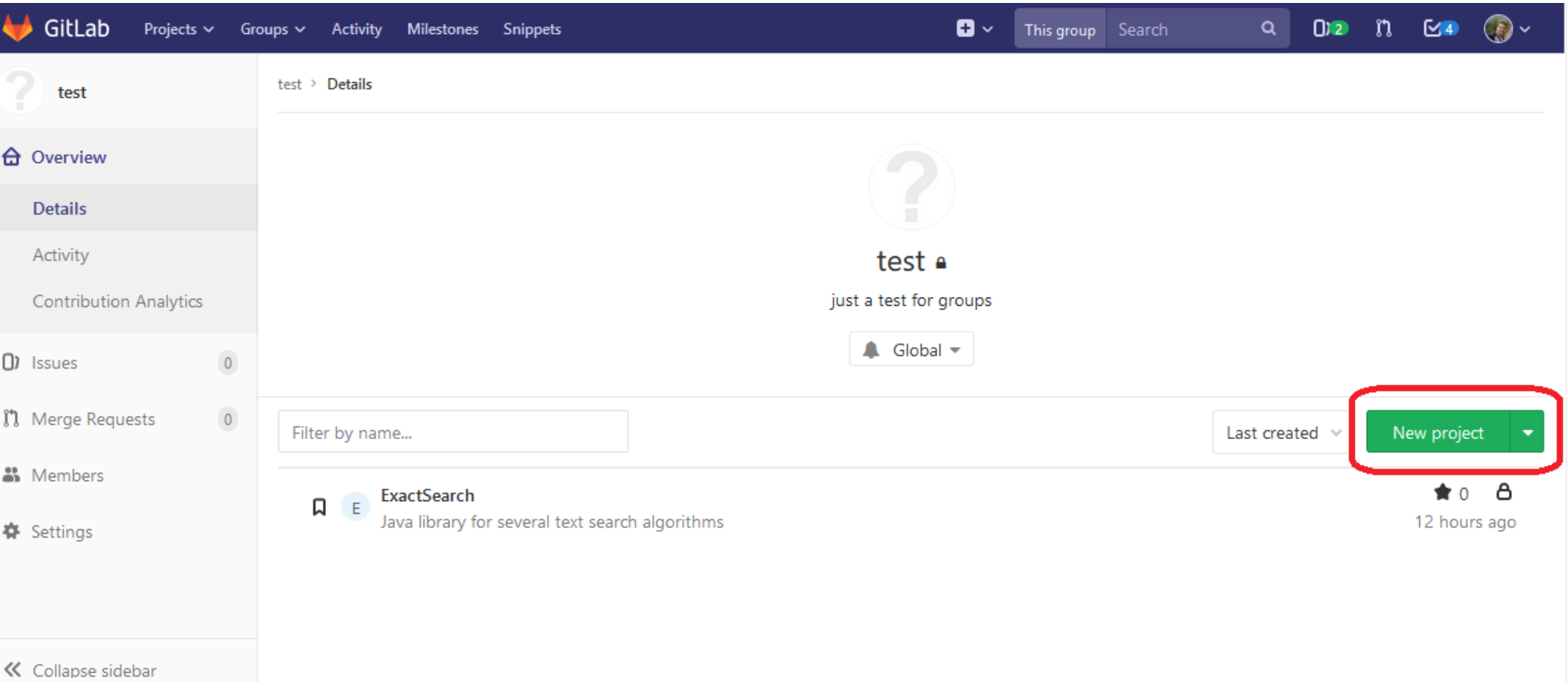
Anlegen eines git Repositories

```
kaidu:biodm08$ git init exactSearch
Erzeugt ein leeres git Repository im Ordner
exactSearch
```

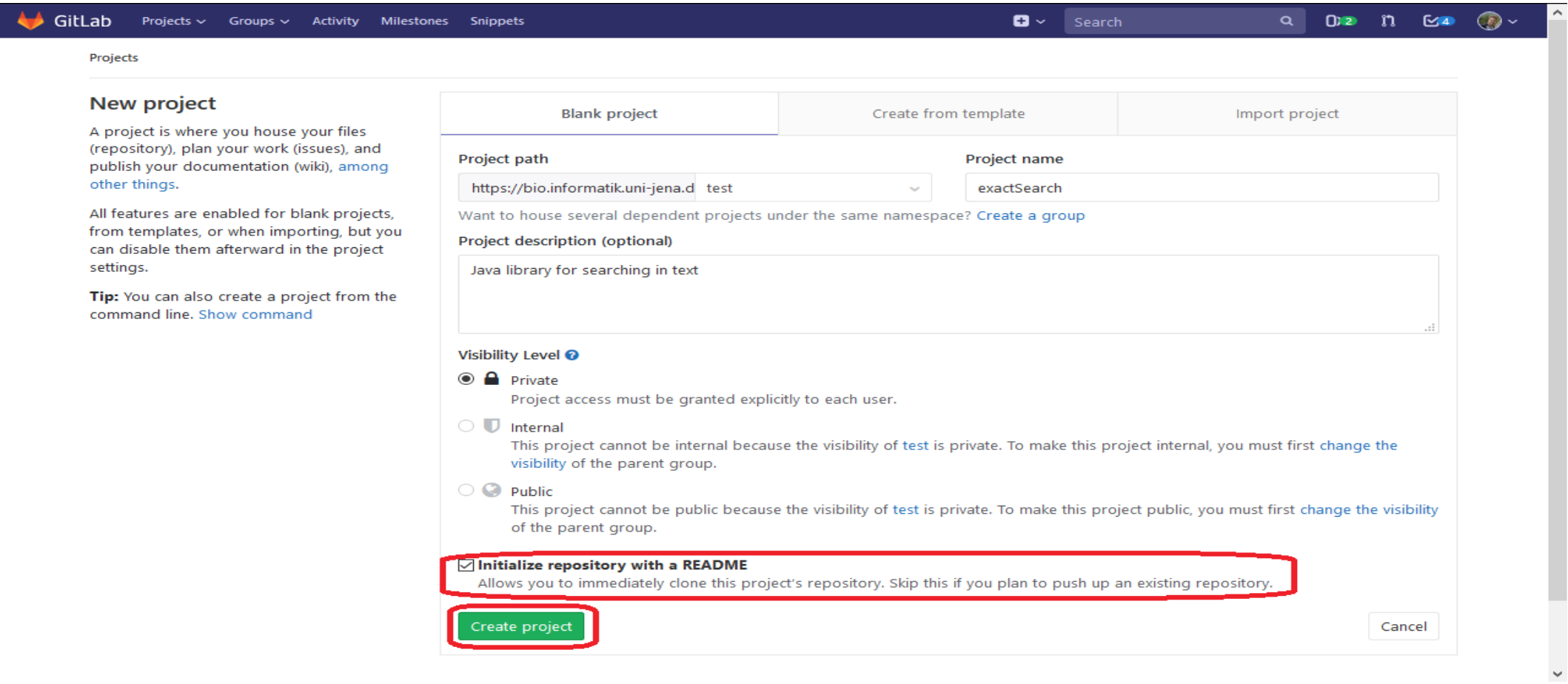
• Mit git init legt ihr ein neues git Repository an. Nützlich, wenn ihr mal ein Versionskontrollsystem z.B. für eure Master-Arbeit braucht

• **Für dieses Praktikum ist es einfacher, ihr legt zuerst das zentrale git repository an, und erzeugt dann eine lokale Kopie**

Anlegen eines git Repositories



Anlegen eines git Repositories



Kopieren eines Repositories

```
kaidu:linux$ git clone https://bio.informatik.uni-jena.de/gitlab/test/exactSearch.git
Cloning into 'exactSearch'
```

- **clone** legt eine lokale Kopie eines externen Repositories an
- das Remote Repository, von dem geklont wird, bekommt automatisch den Namen **origin**

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

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

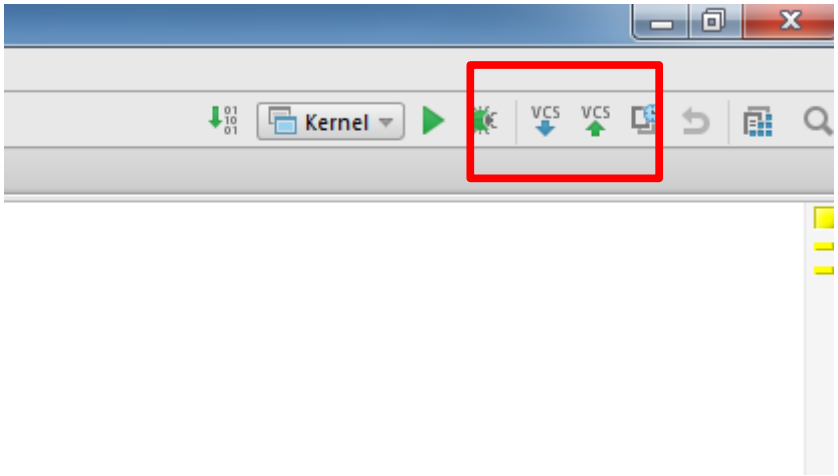
```
kaidu:linux$ git mergetool
```

- 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
```

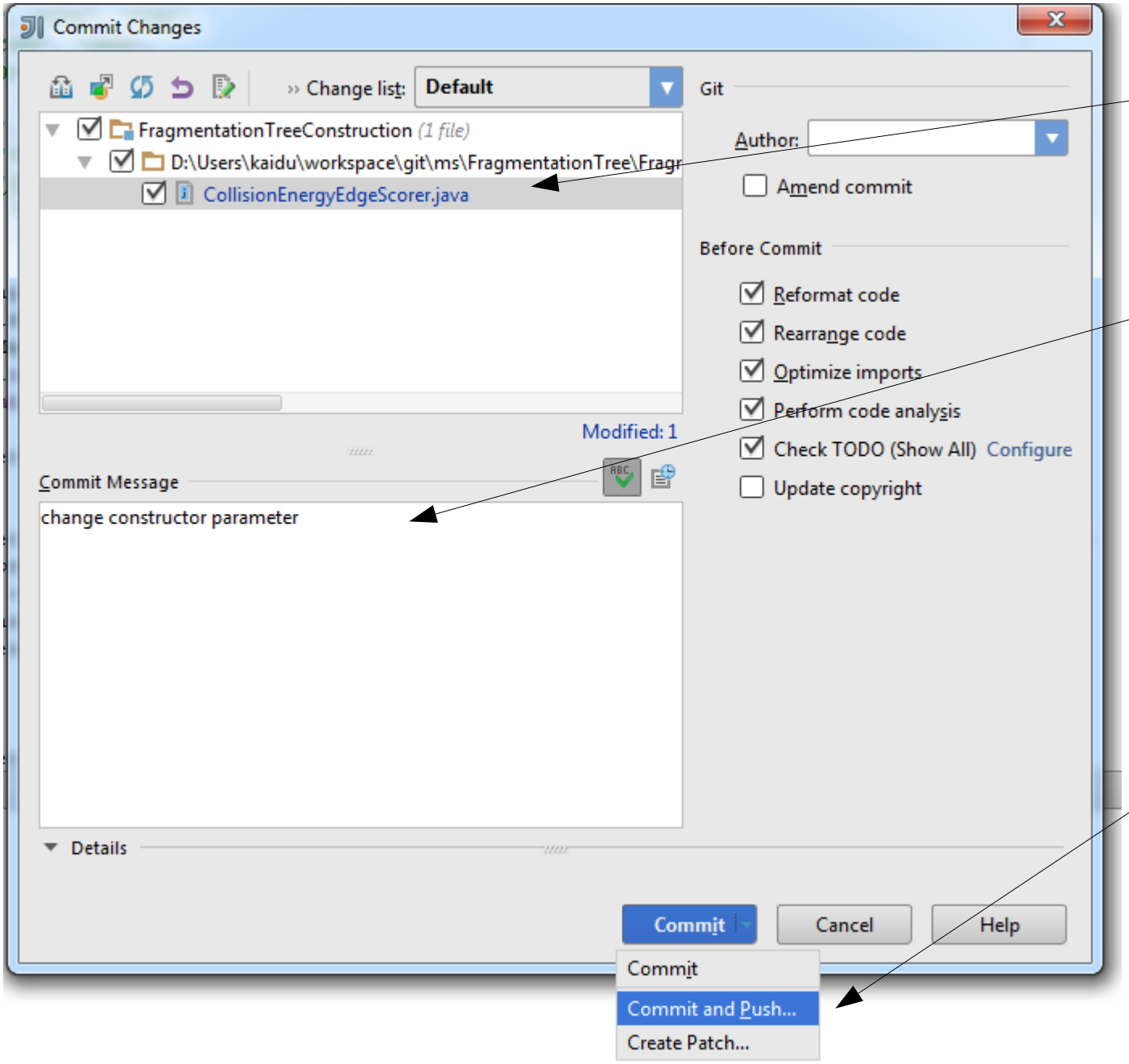
git und Intellij



Lade Änderungen vom Server runter (pull)

Commite eigene Änderungen (commit)
Sende geänderte Versionen an Server (push)

git und Intellij

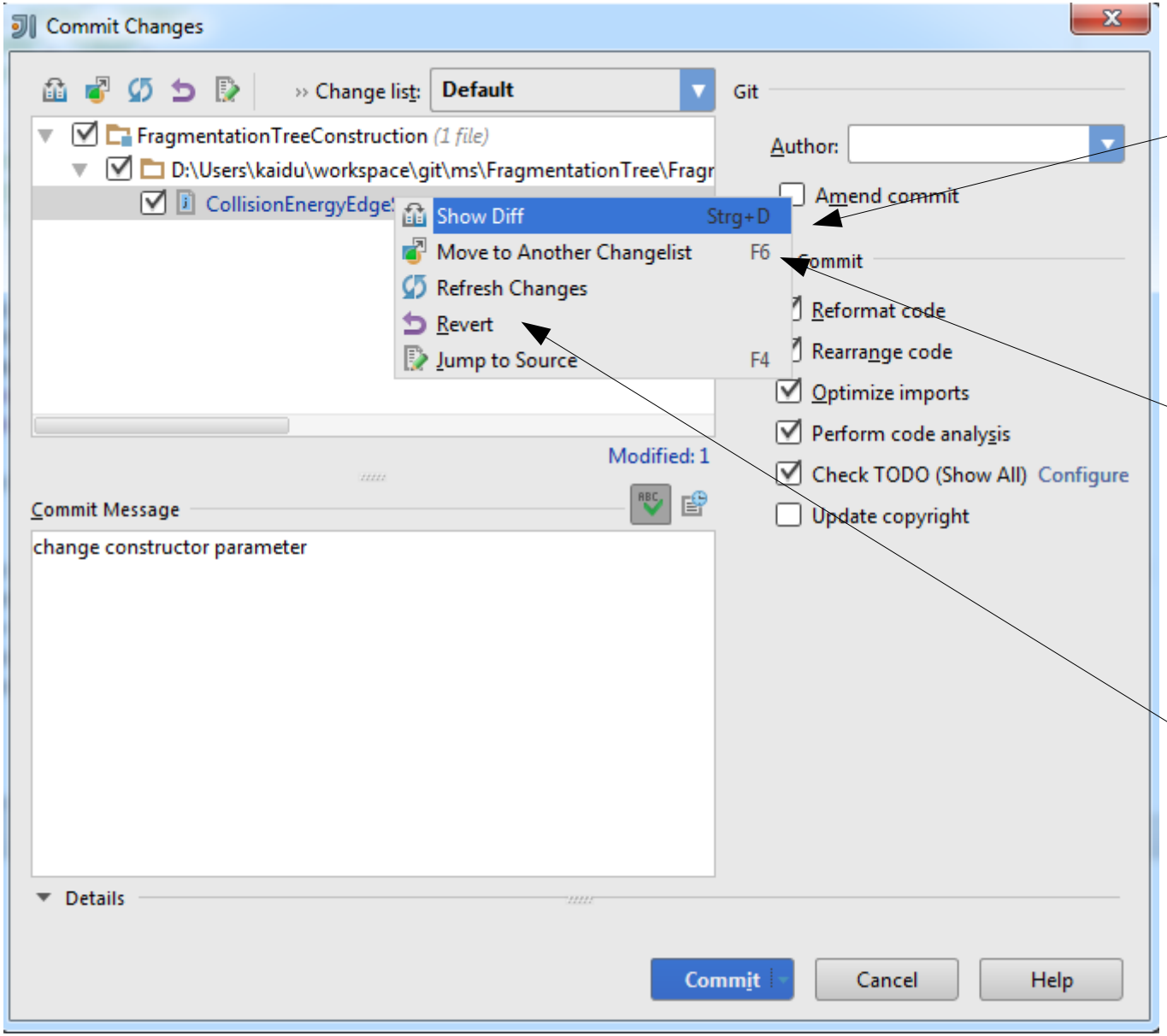


Checkbox aller geänderter Files die commited werden sollen

Commit Message (für log file)

„Commit and Push“ führt beide Operationen nacheinander aus

git und Intellij

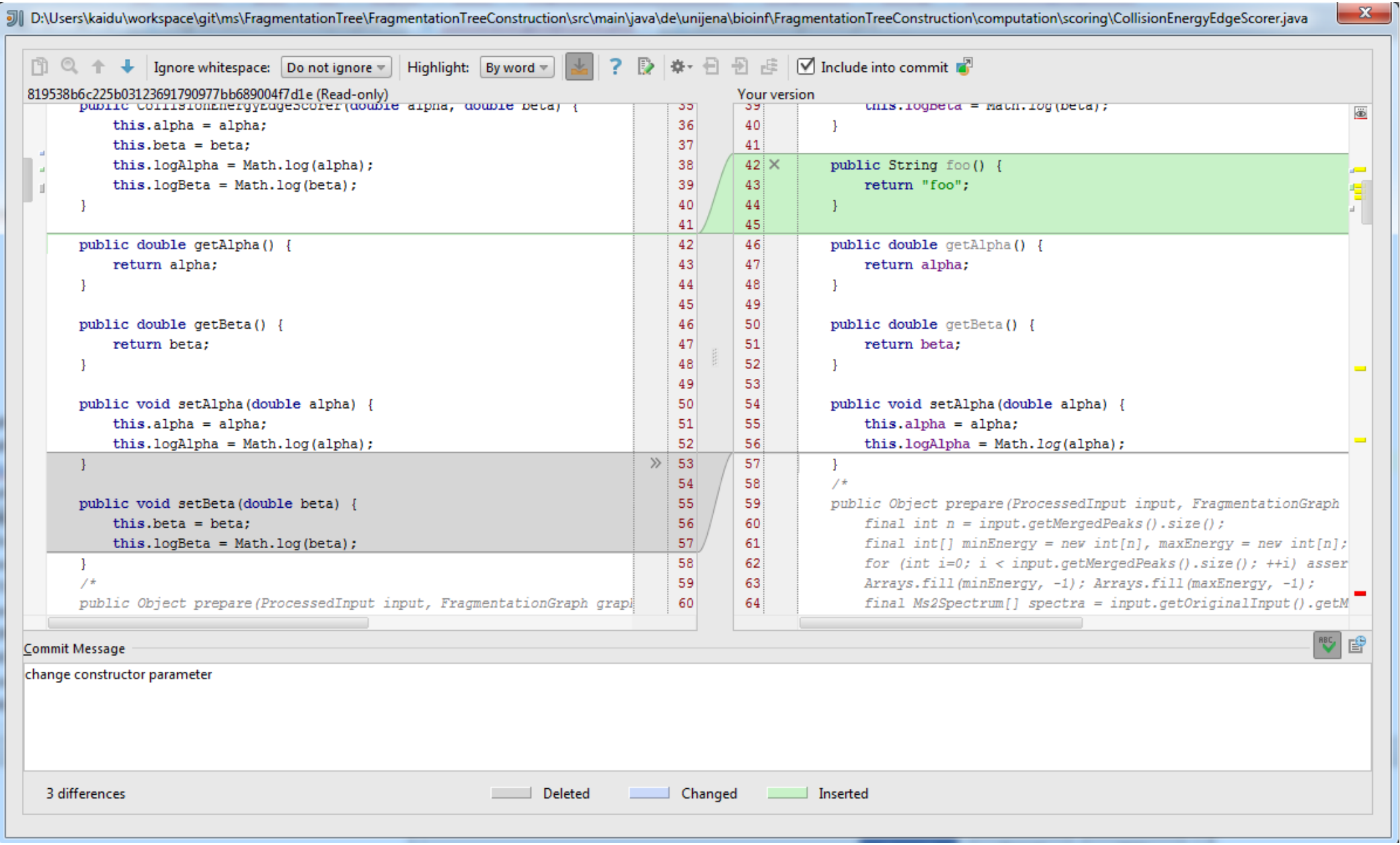


„Show Diff“ zeigt die Änderungen an, die in der Datei gemacht worden sind

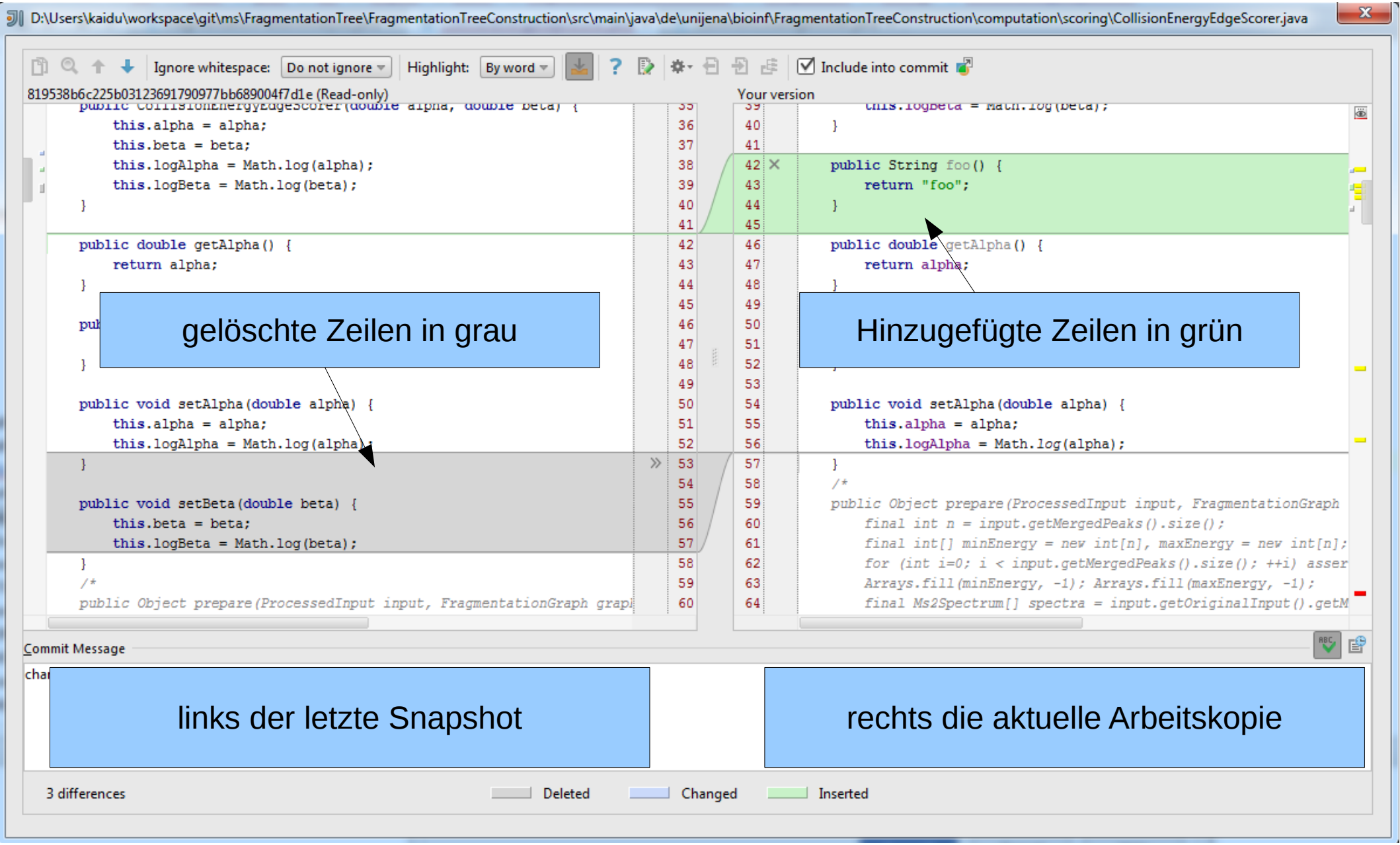
Geänderte Dateien können in Changelists gruppiert werden (nützlich für parallele commits, aber nicht notwendig)

macht lokale Änderungen an dieser Datei rückgängig

git und Intellij

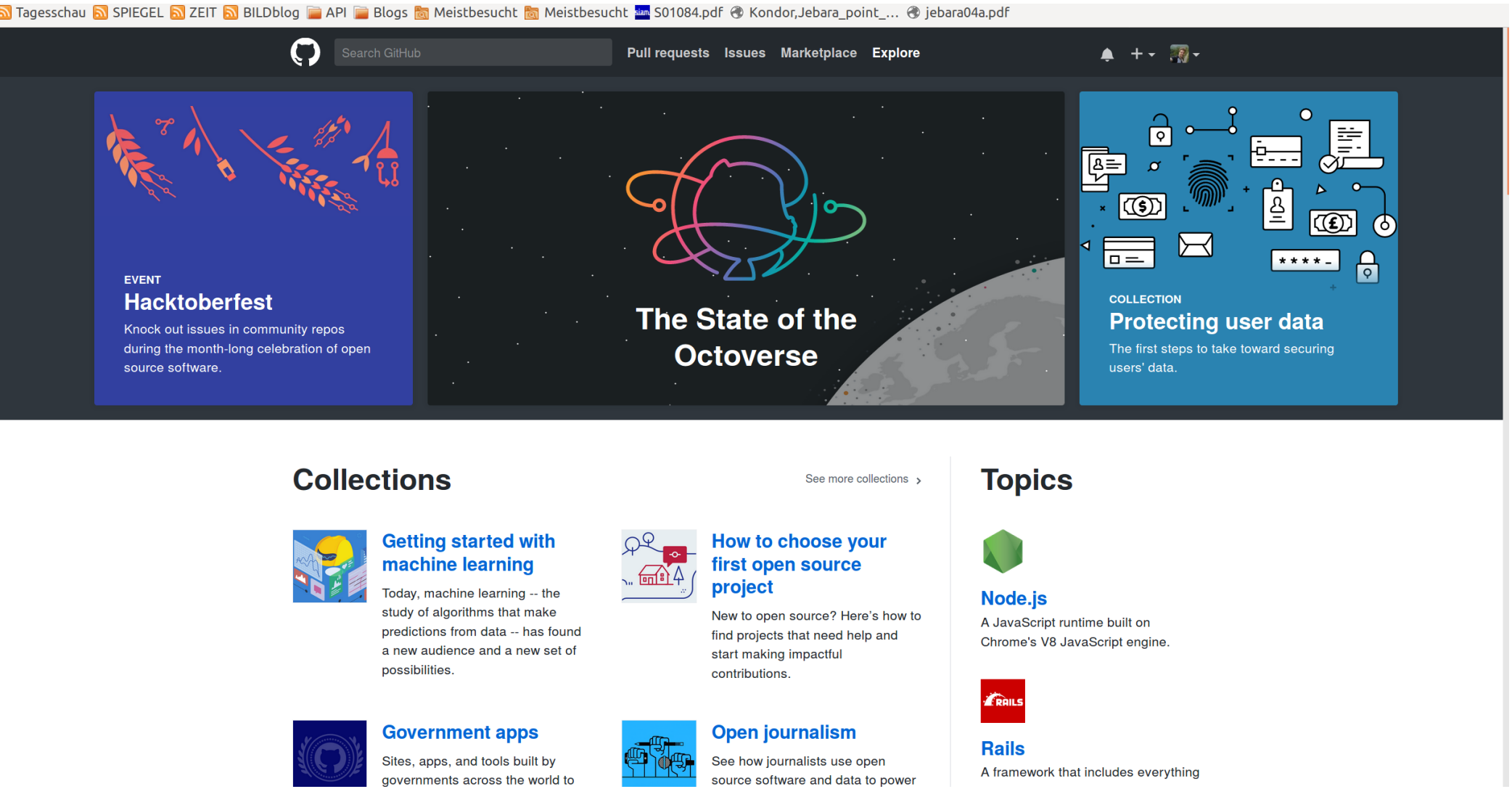


git und Intellij



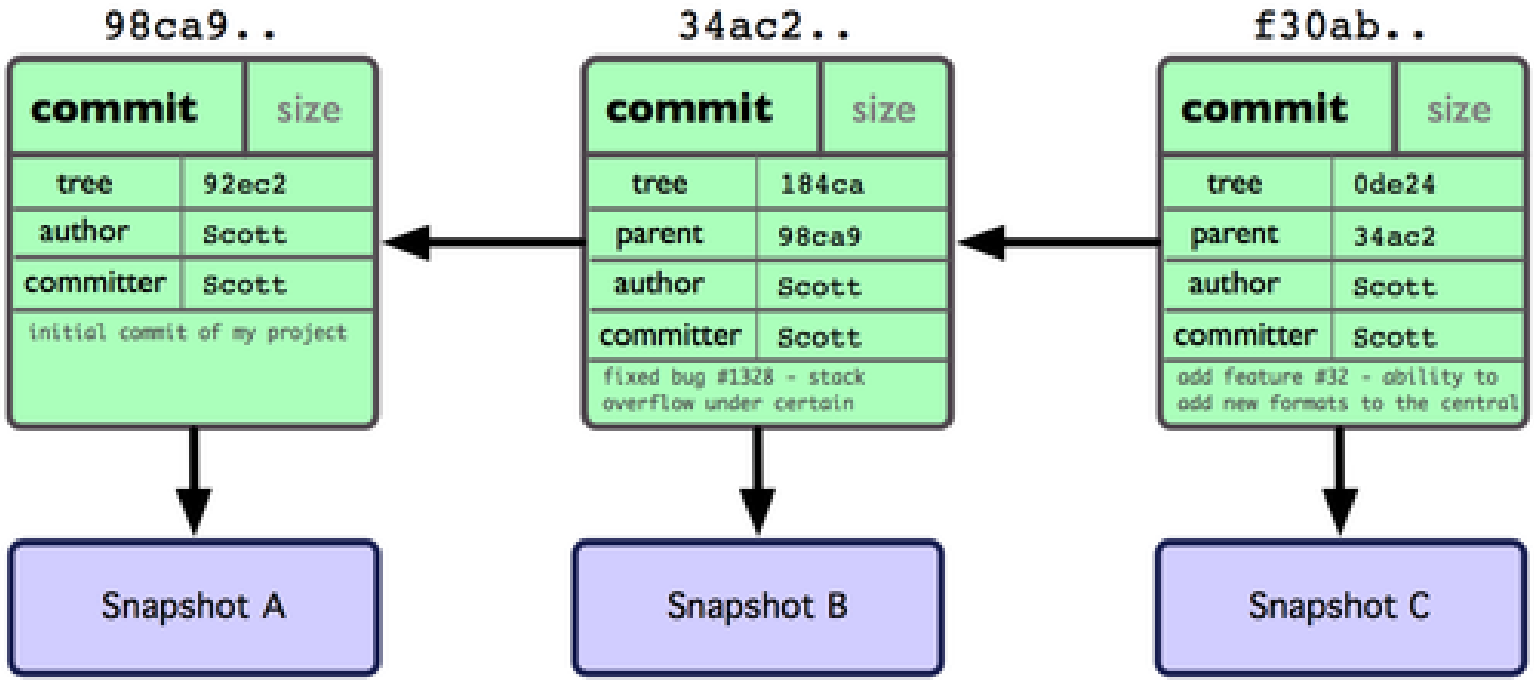
Tipps für git

- 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



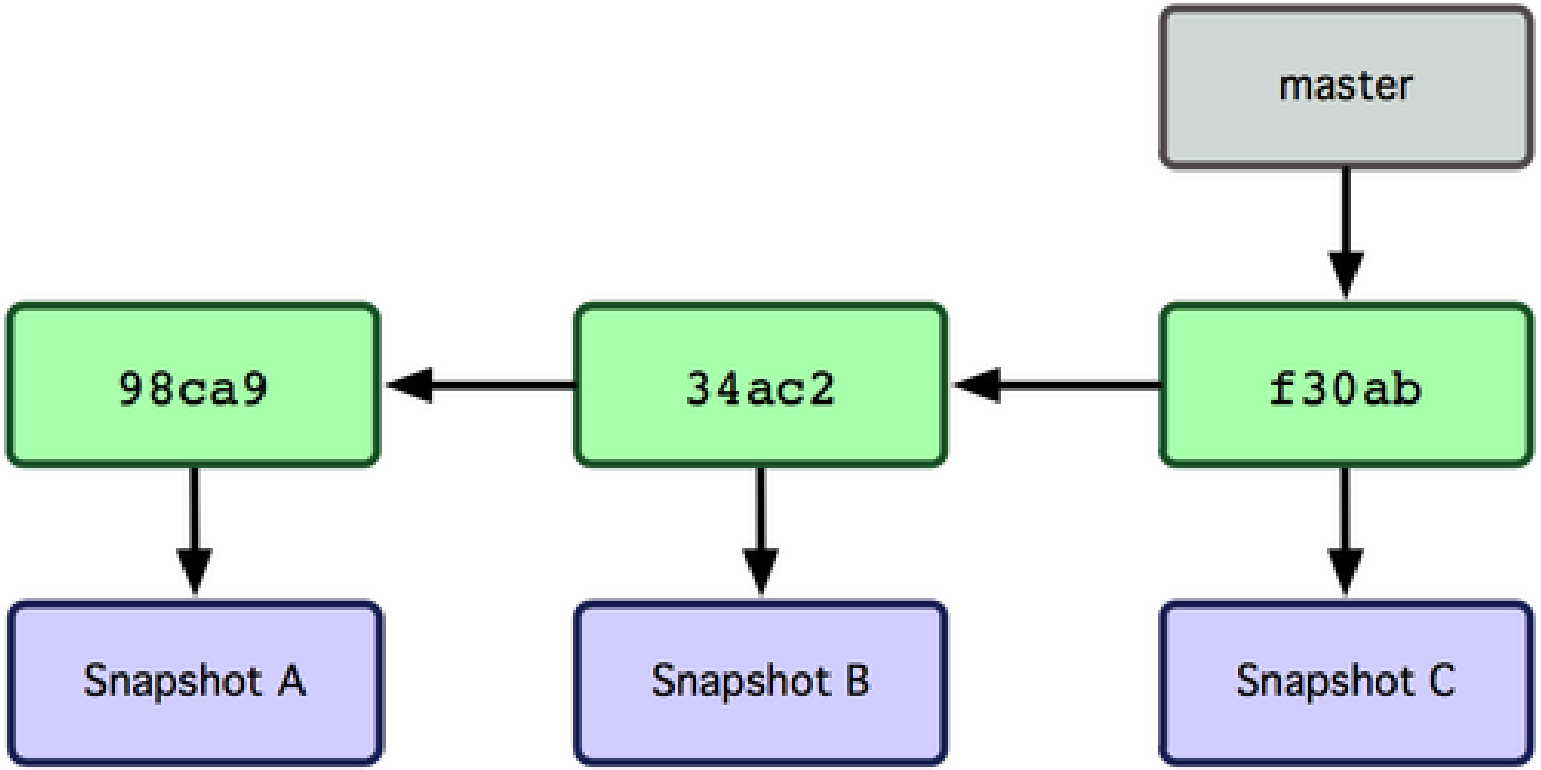
https://github.com

Exkurs: Branches



Exkurs: Branches

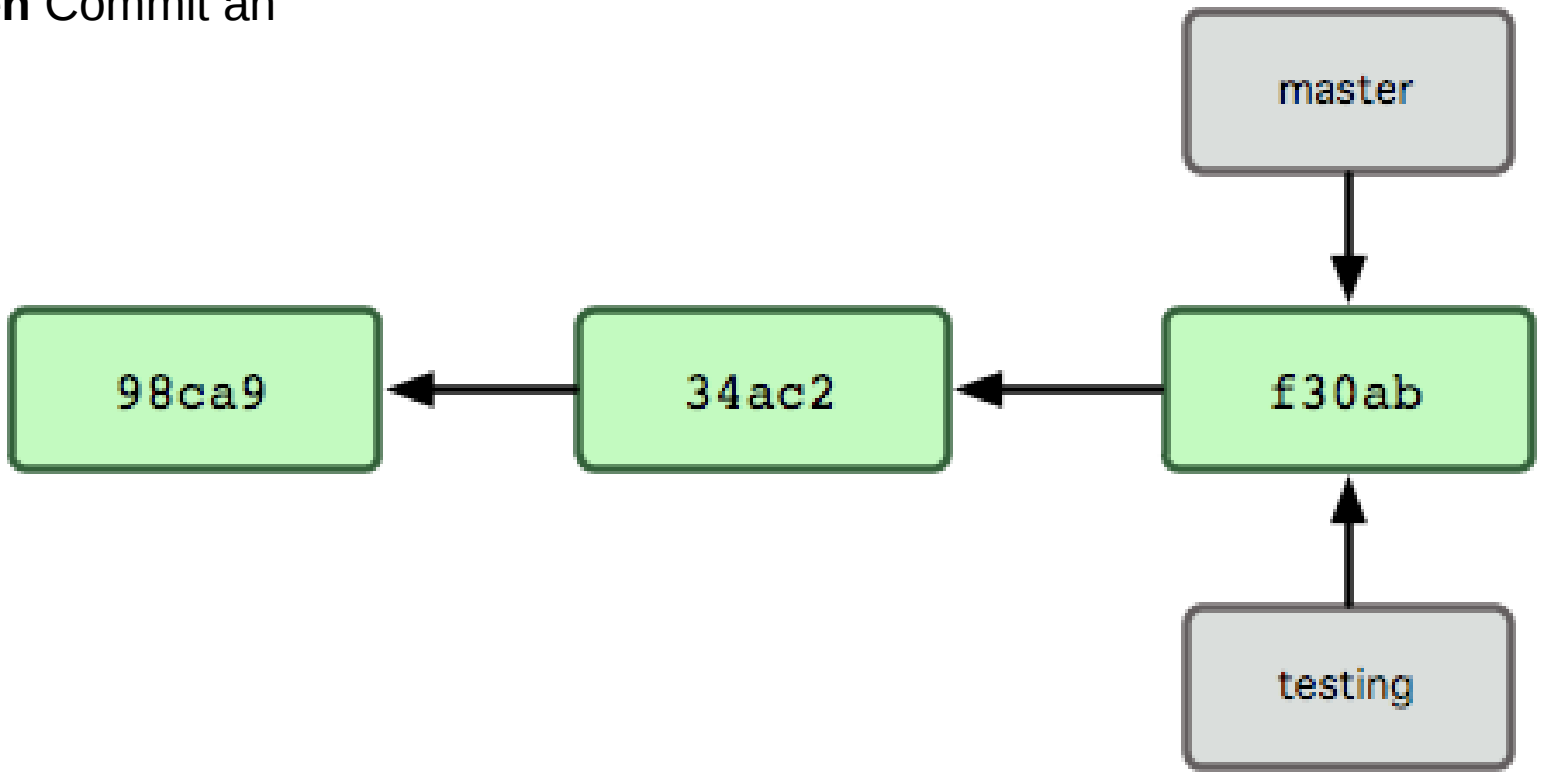
- ein Branch ist ein **Zeiger** auf einen Commit



Exkurs: Branches

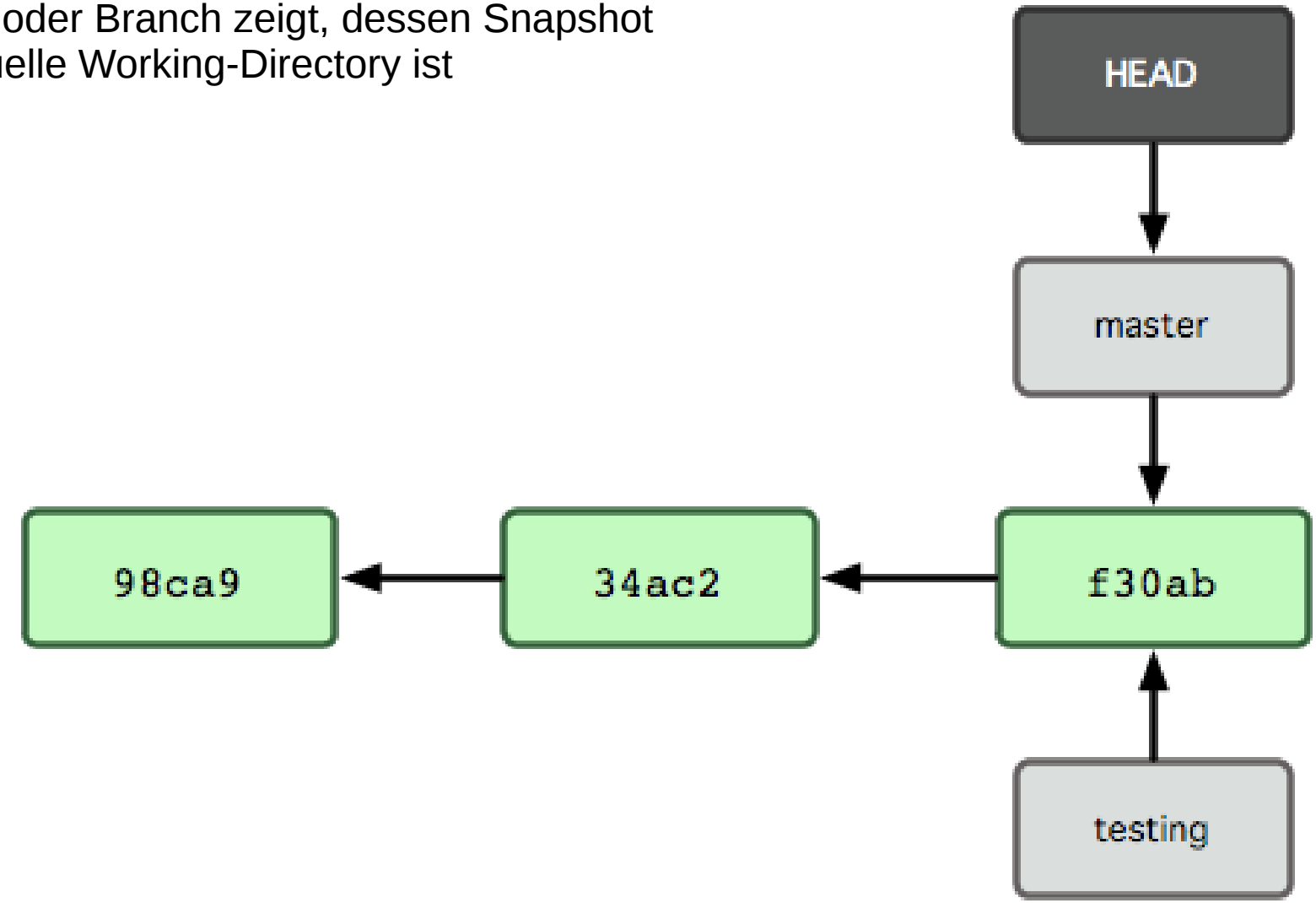
```
kaidu:linux$ git branch testing
```

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

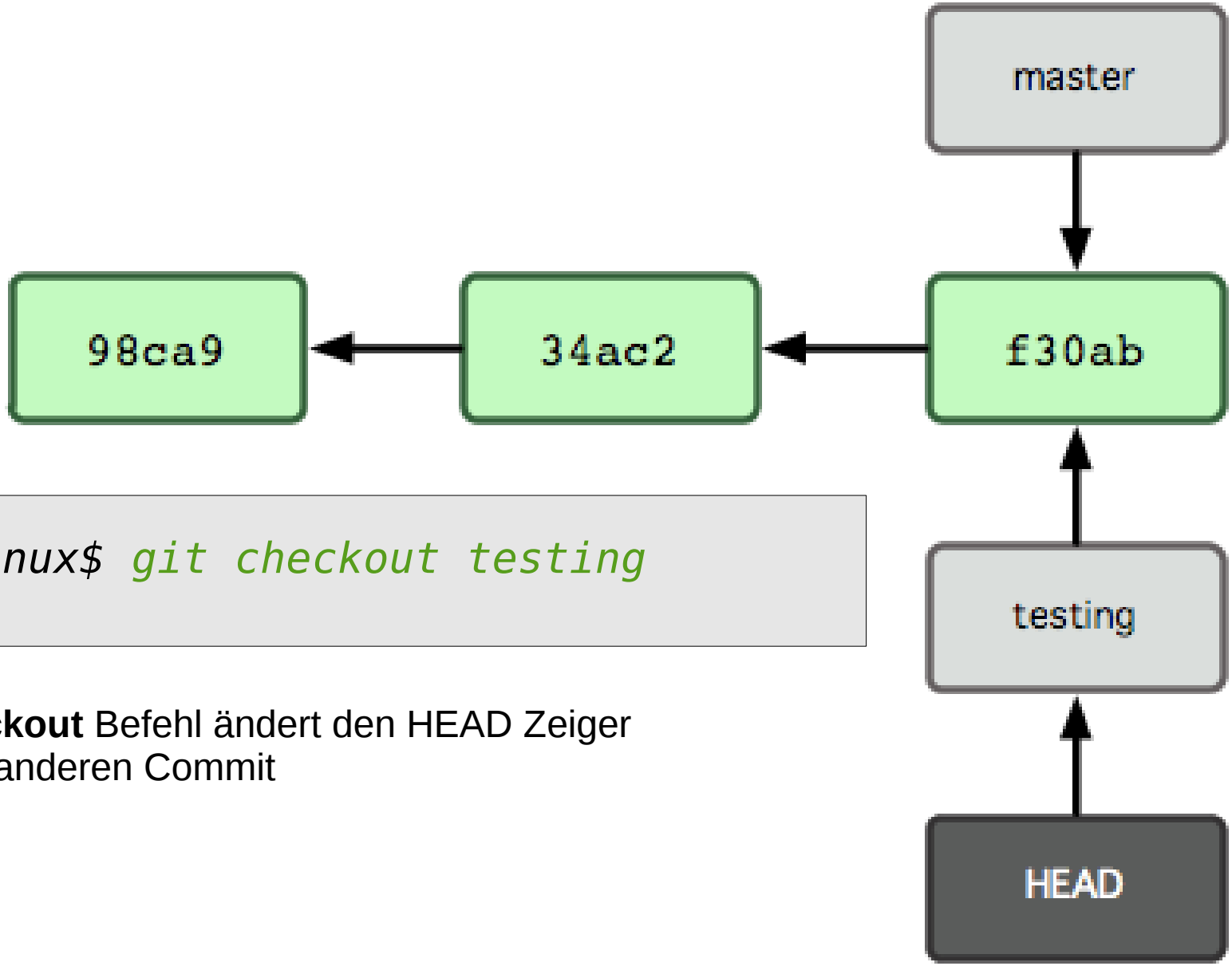


Exkurs: Branches

• **HEAD** ist ein spezieller Zeiger der auf den Commit oder Branch zeigt, dessen Snapshot das aktuelle Working-Directory ist



Exkurs: Branches



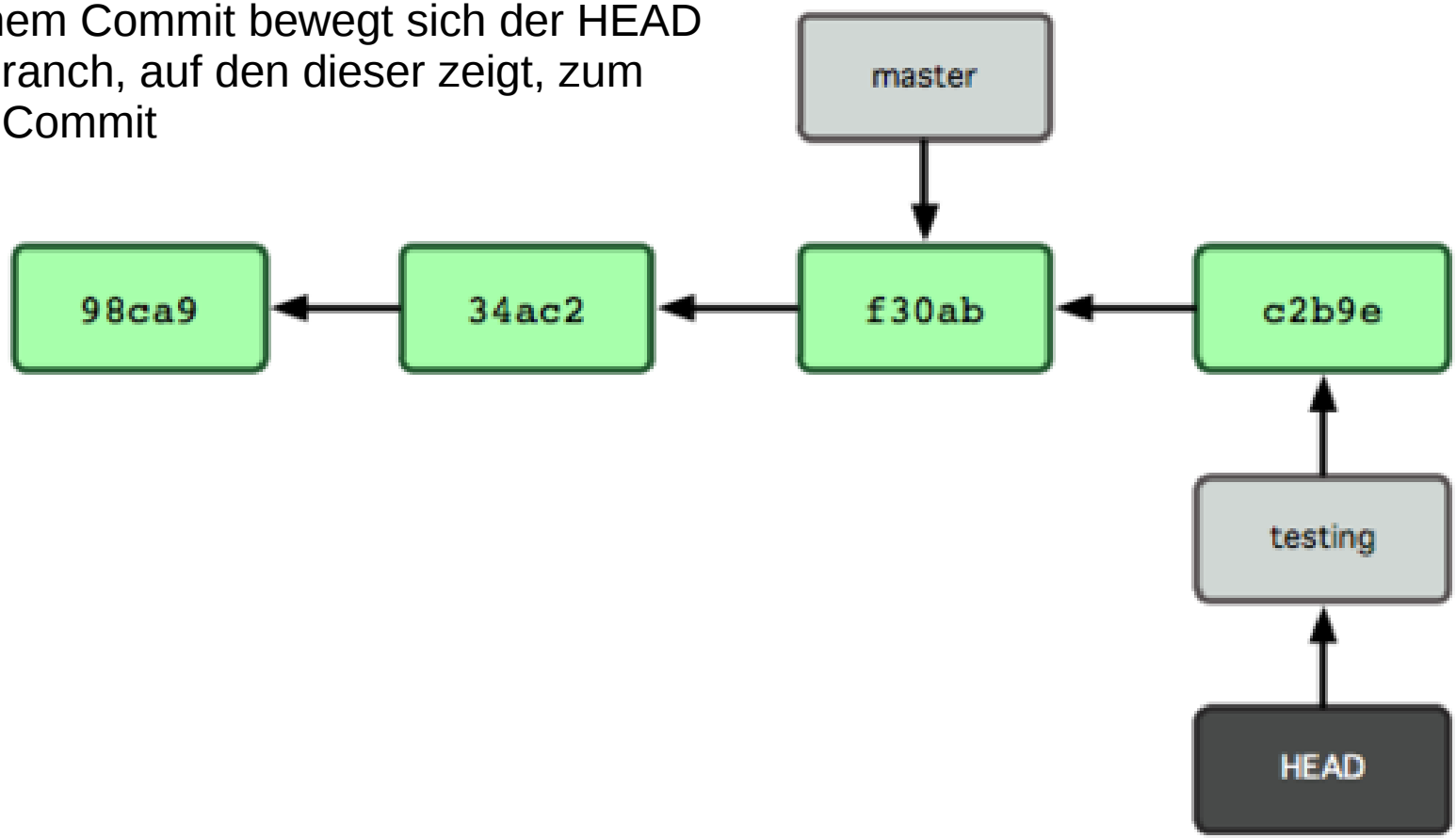

```
kaidu:linux$ git checkout testing
```

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

Exkurs: Branches

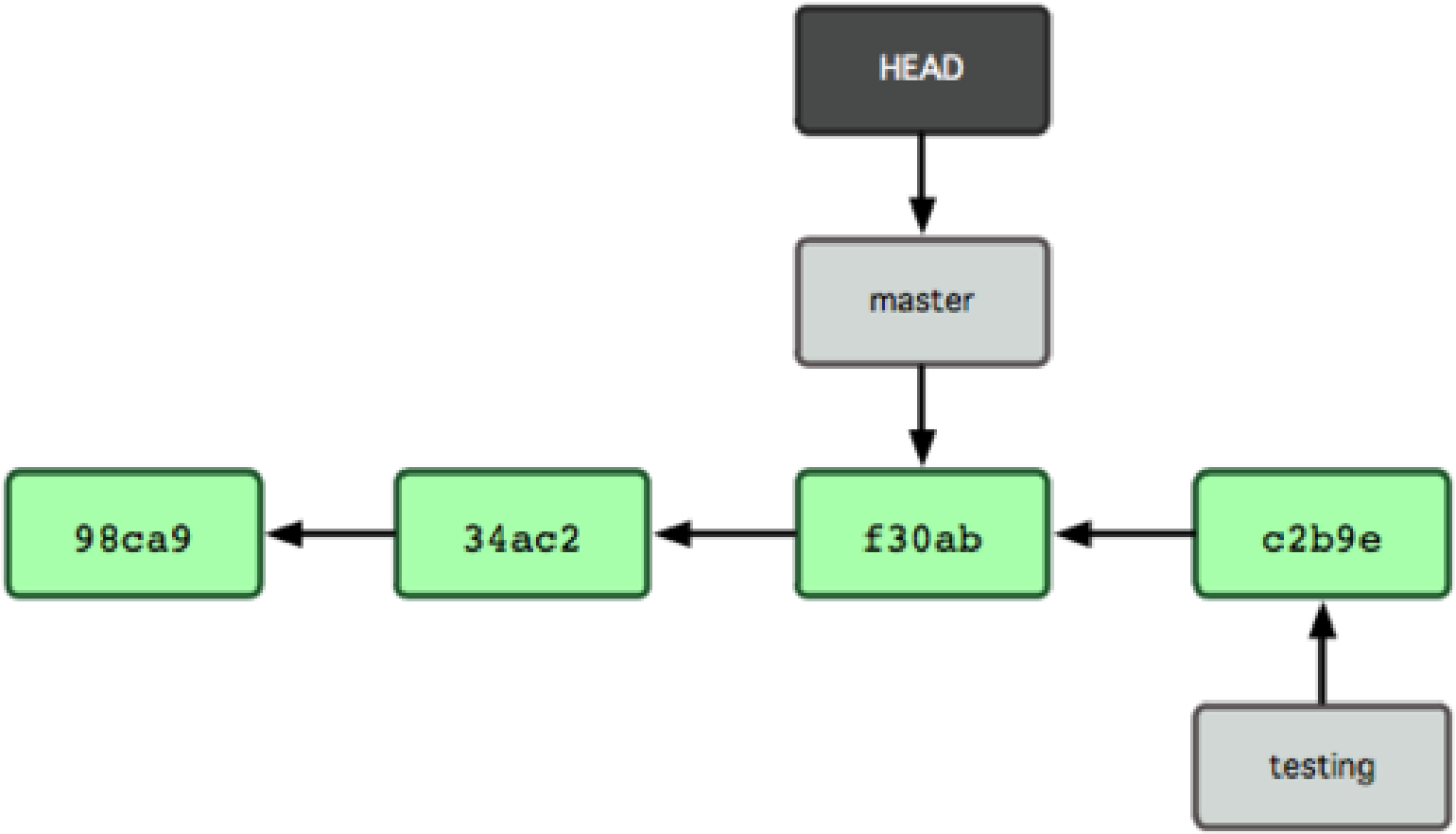
```
kaidu:linux$ git commit -am '...'
```

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



Exkurs: Branches

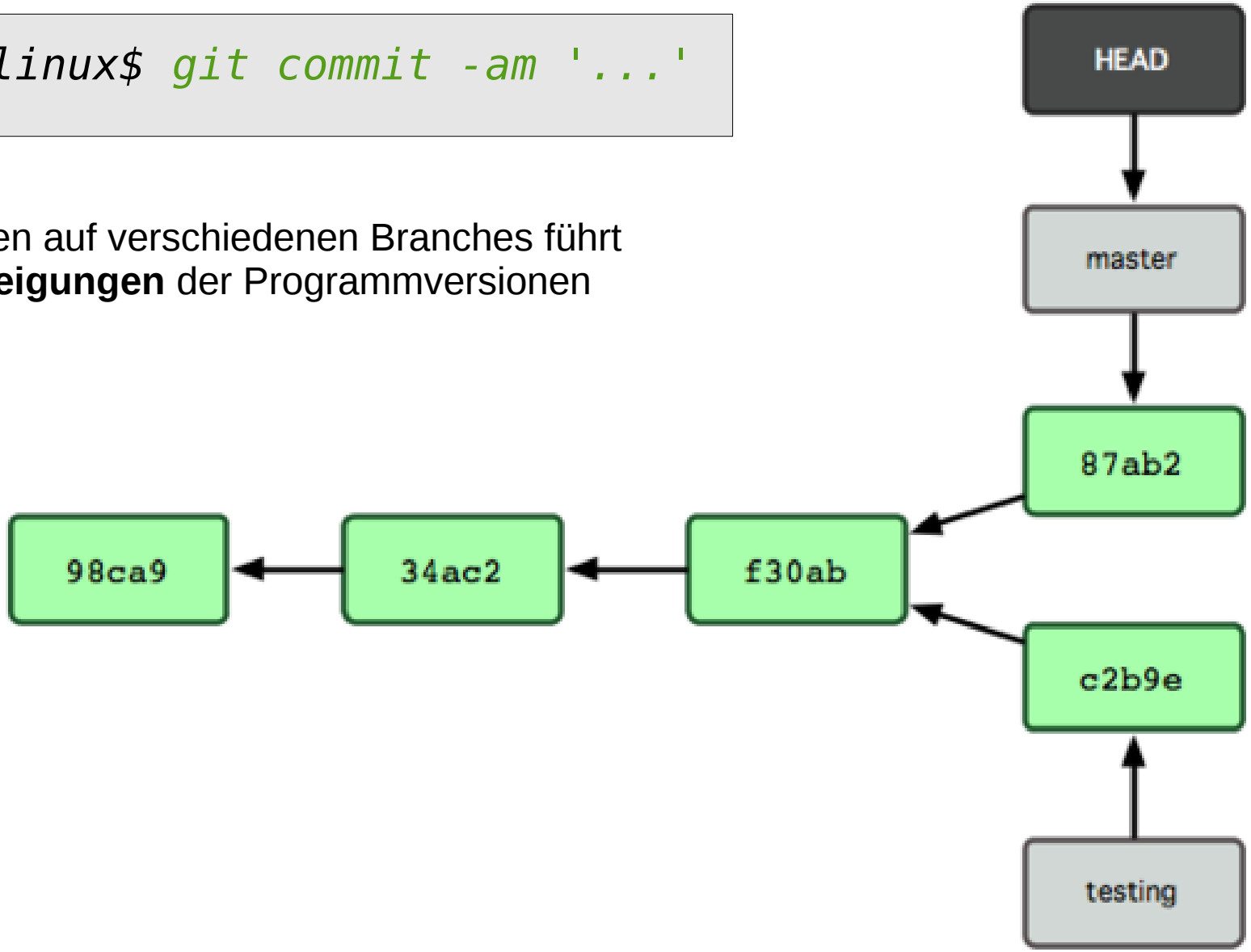
```
kaidu:linux$ git checkout master
```



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