

# **Apache Subversion (SVN)**

Datamining und Sequenzanalyse

Marvin Meusel, Sascha Winter

19.10.2012

# ~~Apache Subversion (SVN)~~

Datamining und Sequenzanalyse

Marvin Meusel, Sascha Winter

19.10.2012

# git

## Datamining und Sequenzanalyse

Marvin Meusel, Bertram Vogel  
(Kai Dührkop)

24.10.2014

# 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](#) [anlegt](#), 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 wird und gesichert und können später wiederhergestellt werden. Versionsverwaltungssysteme werden typischerweise in der Software 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 fest 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 auf '''VCS''' (Version Control System) gebräuchlich.

# Beispiel: Wikipedia

## „Versionsverwaltung“ – Versionsgeschichte

[Logbücher dieser Seite anzeigen](#)

In der Versionsgeschichte suchen

bis Jahr:  und Monat:  Markierungs-Filter:

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](#))

- [\(Aktuell | Vorherige\)](#)  17:27, 21. Okt. 2014 Kghbln ([Diskussion](#) | [Beiträge](#)) .. (12.700 Bytes) (-15) .. *(Die letzte Textänderung von 79.221.2.219 wurde verworfen und die Seite [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 von 171.18.29.130)*
- [\(Aktuell | Vorherige\)](#)  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 von Mfb]
- [\(Aktuell | Vorherige\)](#)  09:40, 23. Mai 2014 MRosetree ([Diskussion](#) | [Beiträge](#)) **K** .. (12.687 Bytes) (-4) .. [\(→Beispiele: Verlinkung auf Subversion Artikel, nicht auf Weiterleitung\)](#)
- [\(Aktuell | Vorherige\)](#)  09:13, 8. Mai 2014 195.124.7.210 ([Diskussion](#)) .. (12.691 Bytes) (+42) .. *(Link auf azyklischer Graph von der Weiterleitung direkt auf die relevante Seite)*
- [\(Aktuell | Vorherige\)](#)  22:36, 6. Feb. 2014 Elektrolurch ([Diskussion](#) | [Beiträge](#)) **K** .. (12.649 Bytes) (+9) .. [\(→Verteilte Versionsverwaltung\)](#) [\(rückgängig\)](#) [automatisch gesichtet]
- [\(Aktuell | Vorherige\)](#)  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]

Version vom 29. Oktober 2013, 11:48 Uhr (Bearbeiten)

[Astorabe71](#) (Diskussion | Beiträge)

(→*Beispiele*)

← Zum vorherigen Versionsunterschied

[gesichtete Version]

Aktuelle Version vom 21. Oktober 2014, 17:27 Uhr (Bearbeiten) (rückgängig)

[Kghbln](#) (Diskussion | Beiträge)

(Die letzte Textänderung von 79.221.2.219 wurde verworfen und die Version 134544628 von YMS wiederhergestellt.)

(Markierung: HHVM)

(12 dazwischenliegende Versionen von 9 Benutzern werden nicht angezeigt)

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.

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:

== 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

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.

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:

== 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 **wieder** in **einem** 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

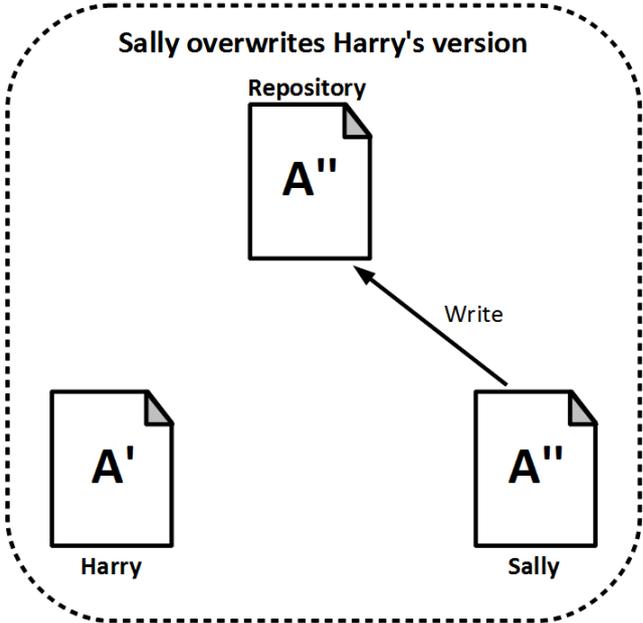
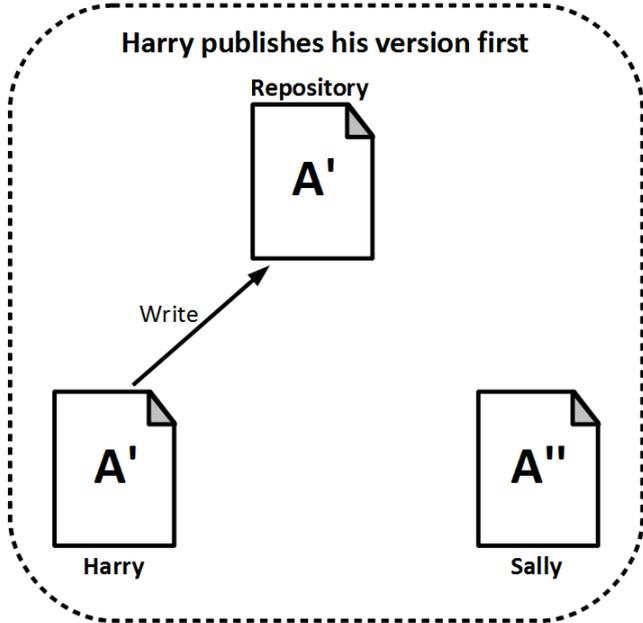
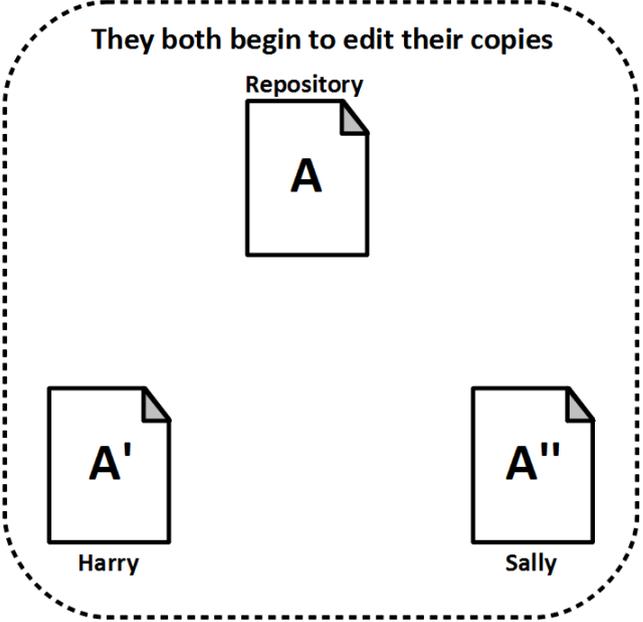
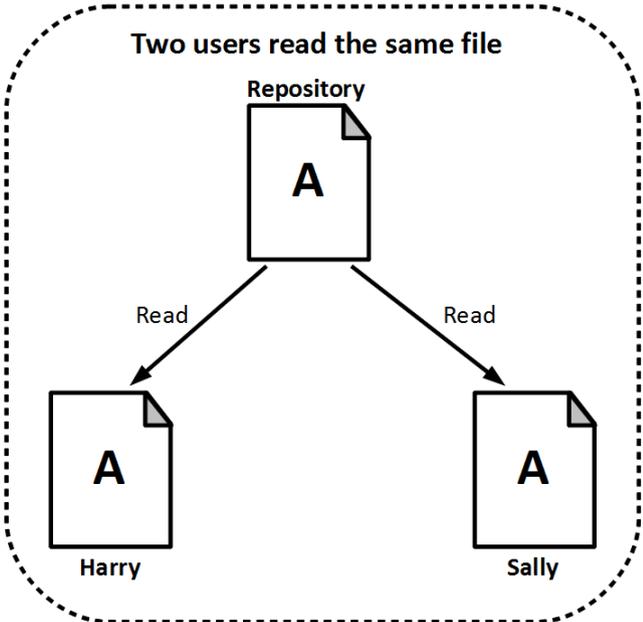
# Was ist Versionsverwaltung?

**Gemeinsam an Sourcecode arbeiten: Fileserver?**

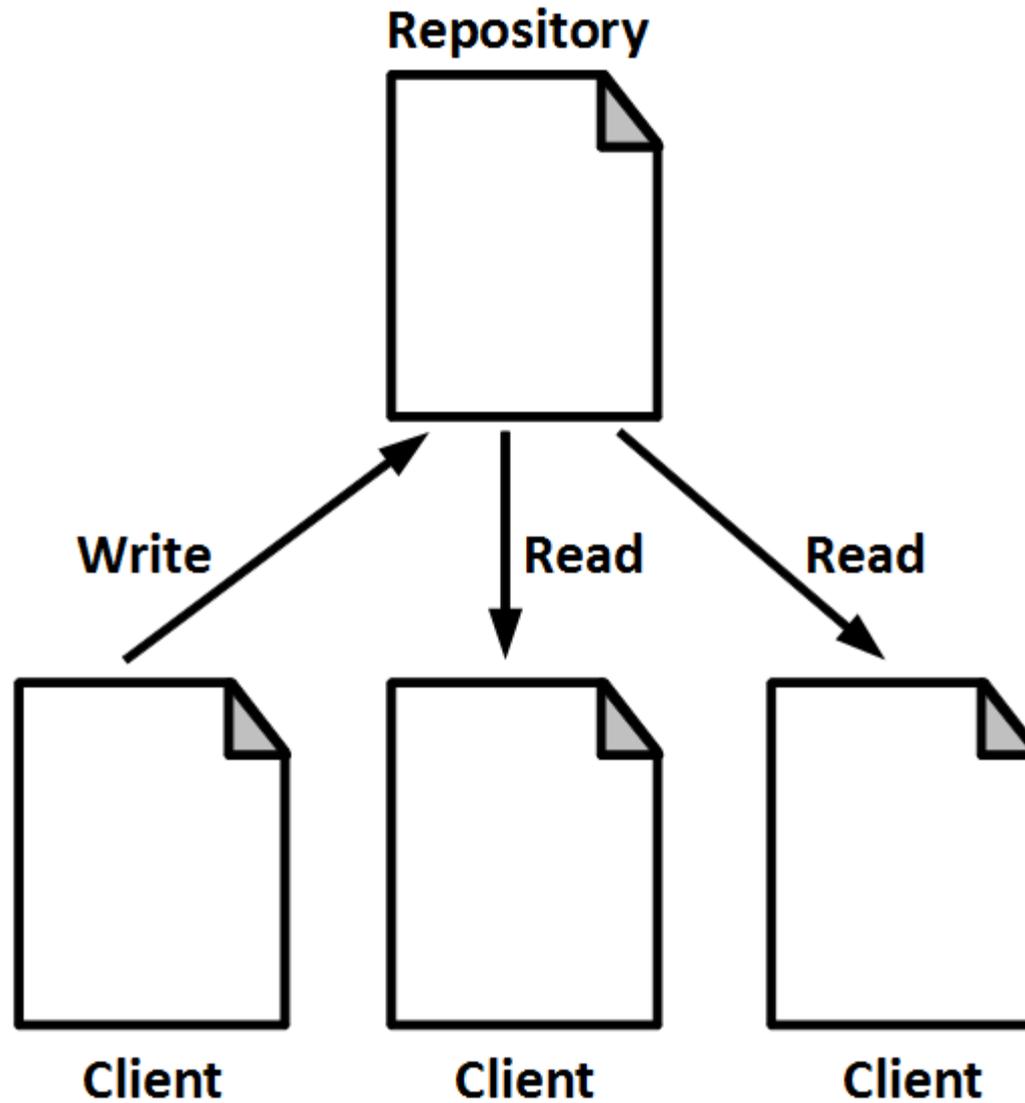


**N F S**  
NETWORK FILE SYSTEM

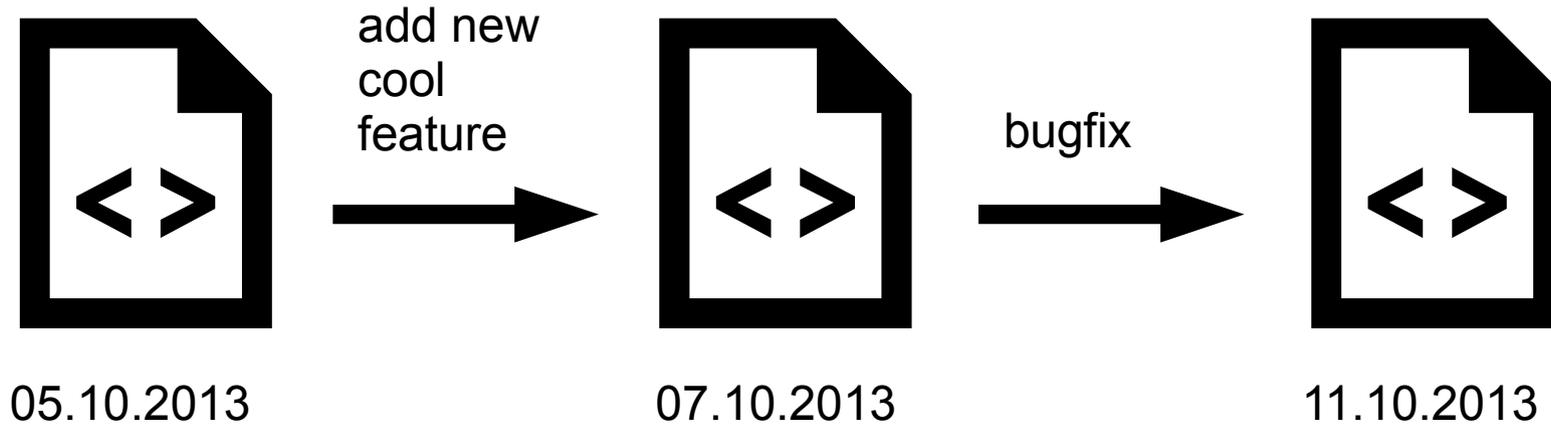
# Das Problem verteilter Dateizugriffe



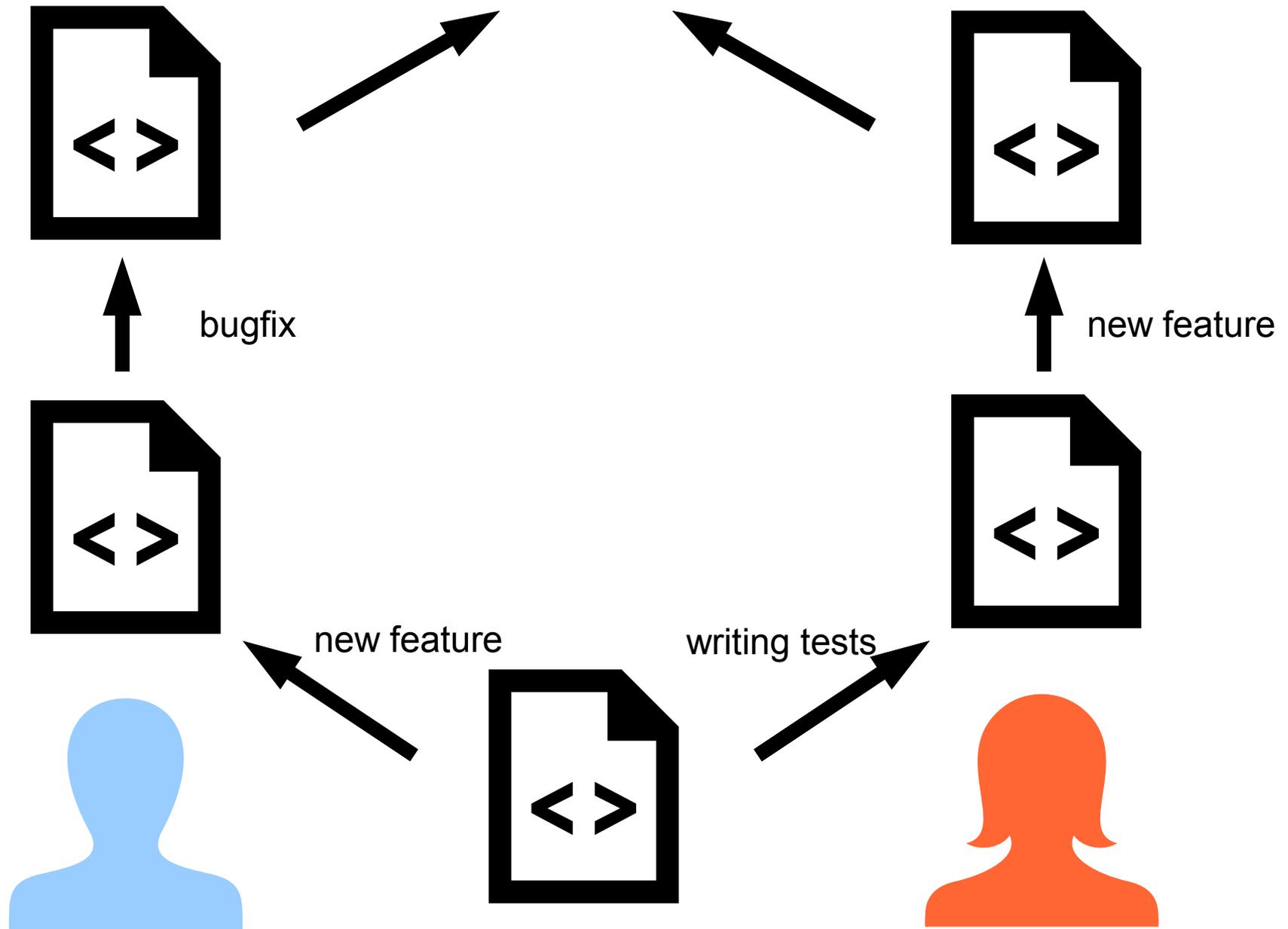
# Versionsverwaltung



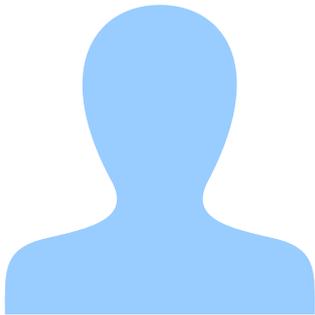
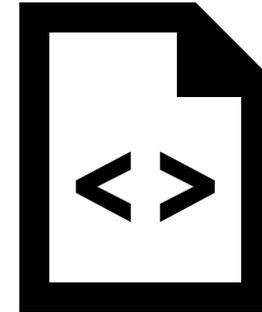
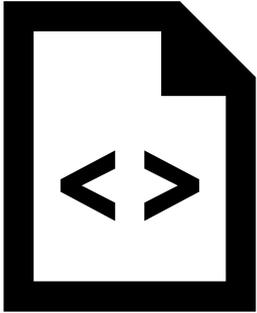
# Versionierung... über die Zeit



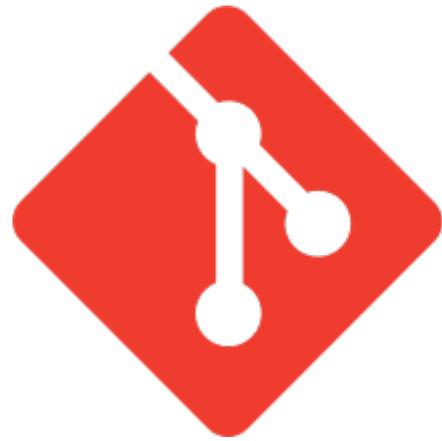
# Versionierung... über mehrere Entwickler



# Konfliktlösung



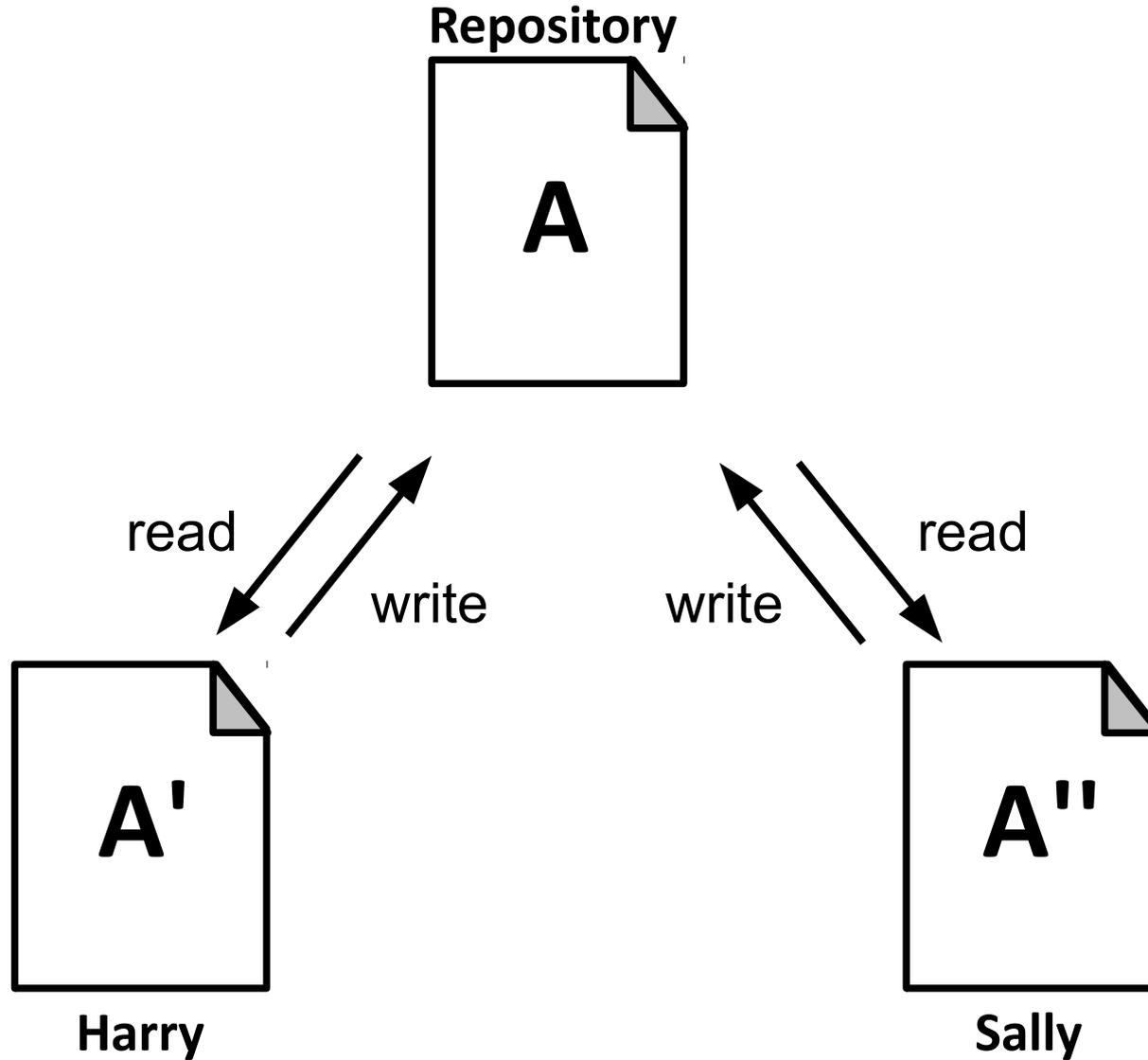
# Versionskontrollsysteme



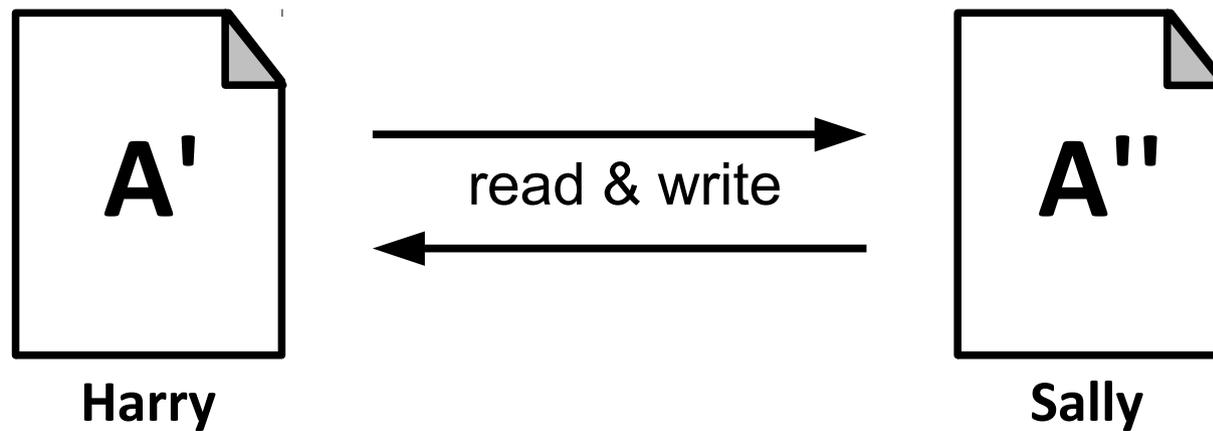
**git**



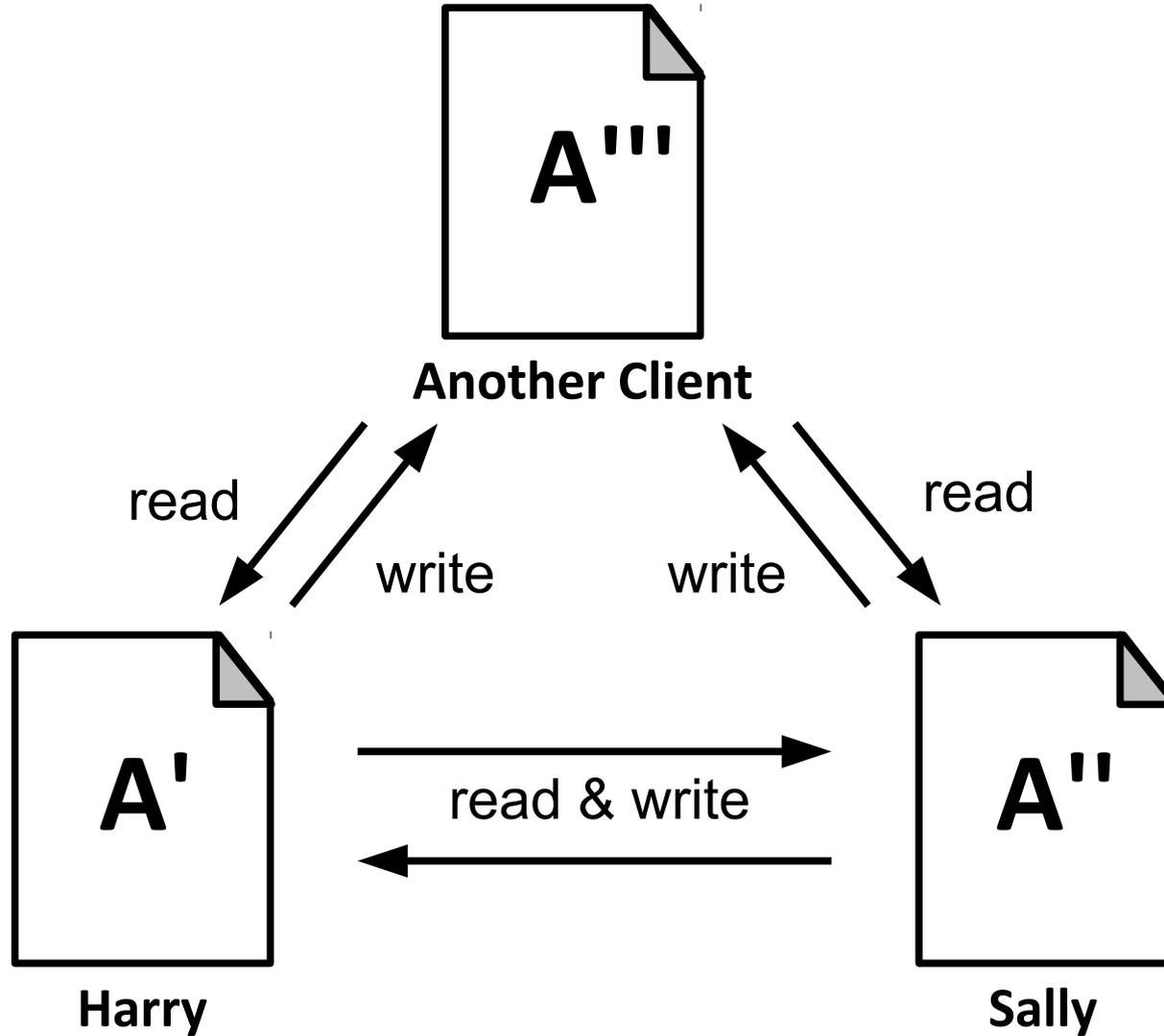
# SVN - eine zentrale Versionsverwaltung



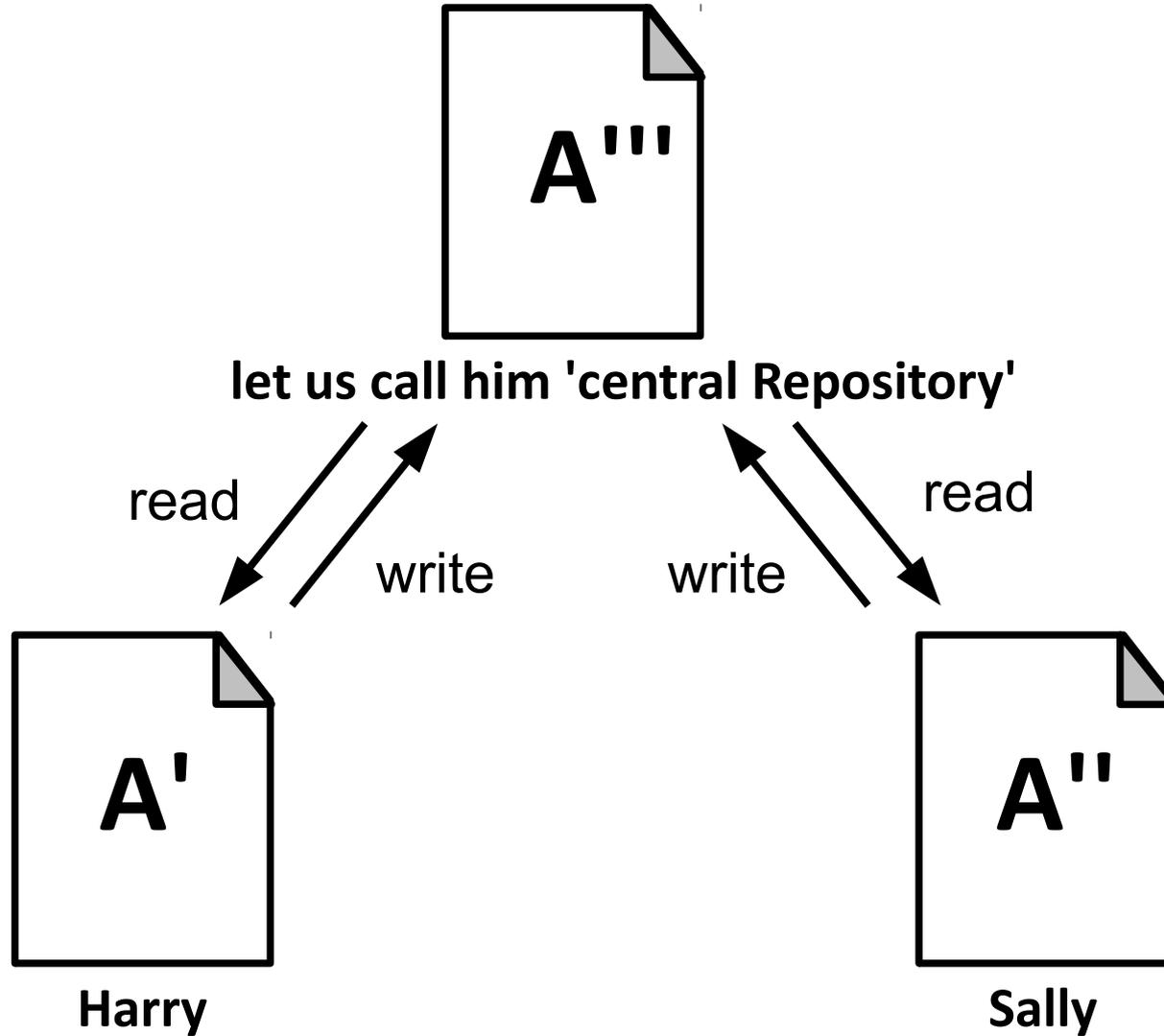
# git - eine dezentrale Versionsverwaltung



# git - eine dezentrale Versionsverwaltung



# git - eine dezentrale Versionsverwaltung



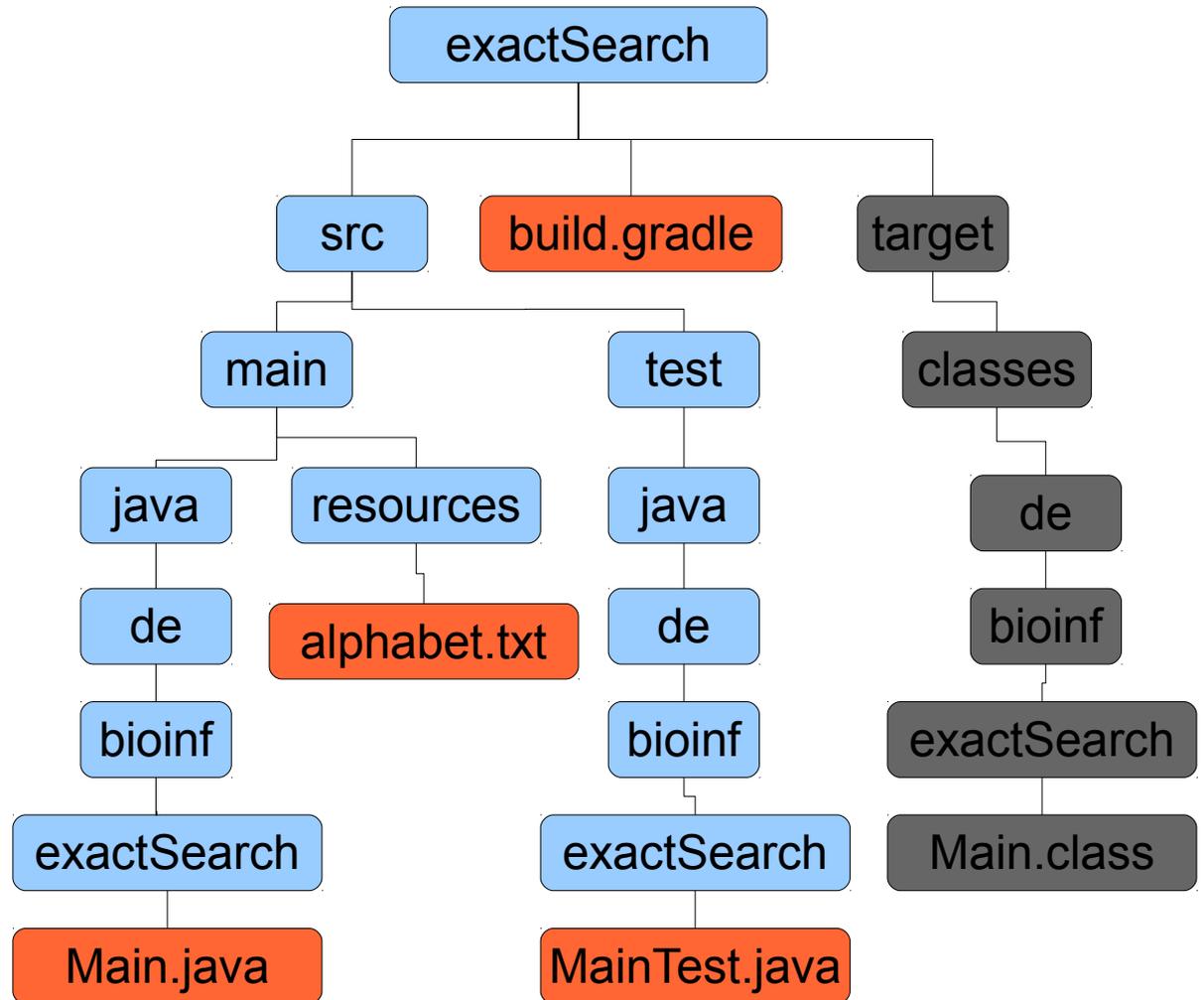
# git - eine dezentrale Versionsverwaltung

- 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

## Repository:

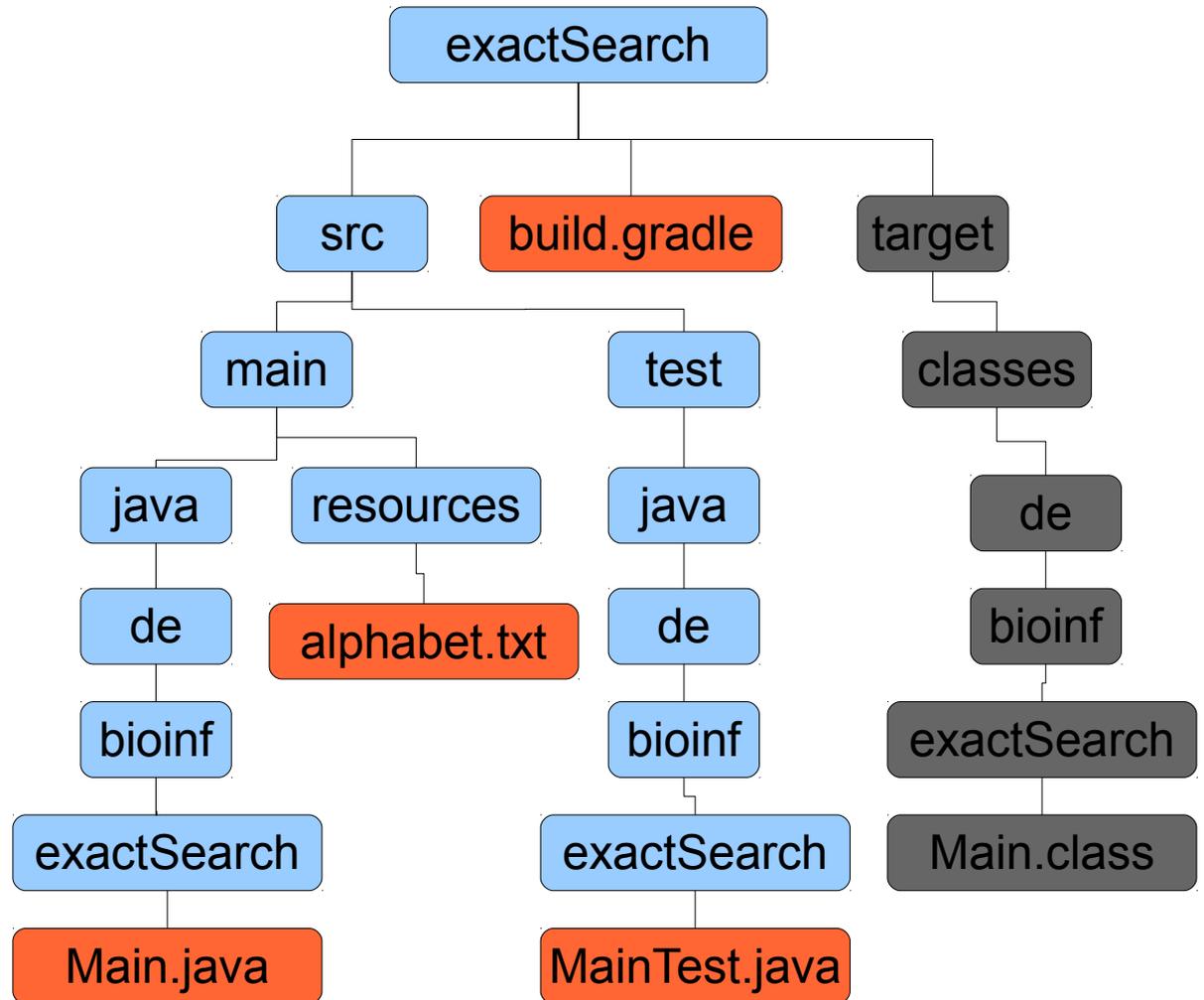
- ein versioniertes Projekt
- die Gesamtheit aller versionierten Dokumente
- in der Regel ein Verzeichnisbaum
- nicht alle Dateien im Verzeichnisbaum müssen versioniert sein! (**untracked**)
- in git: nur Dateien sind versioniert. Verzeichnisse nicht! Leere Verzeichnisse „existieren nicht“ im Repository



# Grundlegende Begriffe

## Repository:

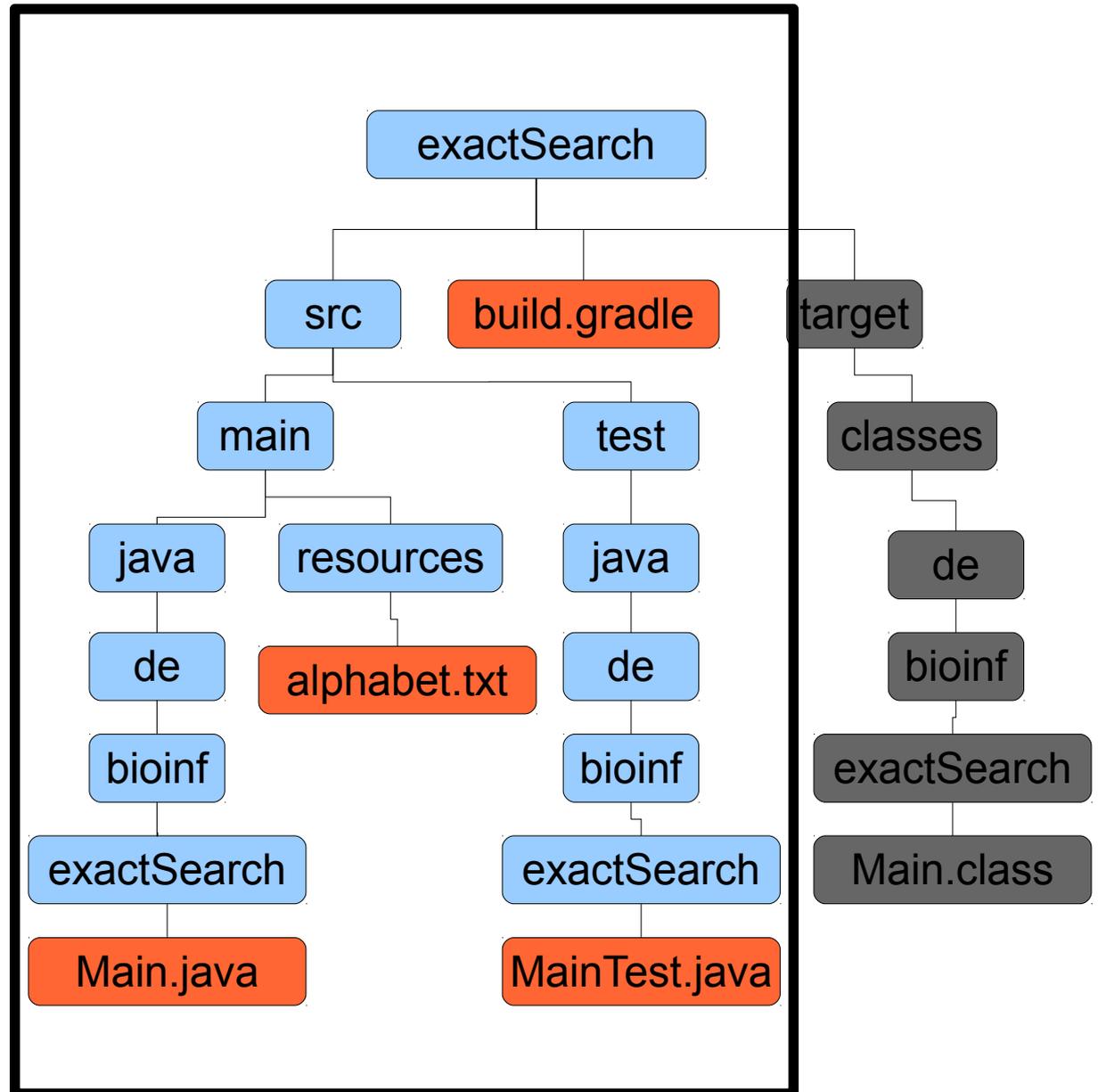
- alles was **Text** ist und **manuell** geändert wird sollte versioniert werden
- alles was **binary** data ist (Bilder, class files, jar files ...) oder **automatisch** generiert wird (.iml files eurer IDE, javadoc, test reports) gehört **NICHT** ins repository



# Grundlegende Begriffe

## Snapshot:

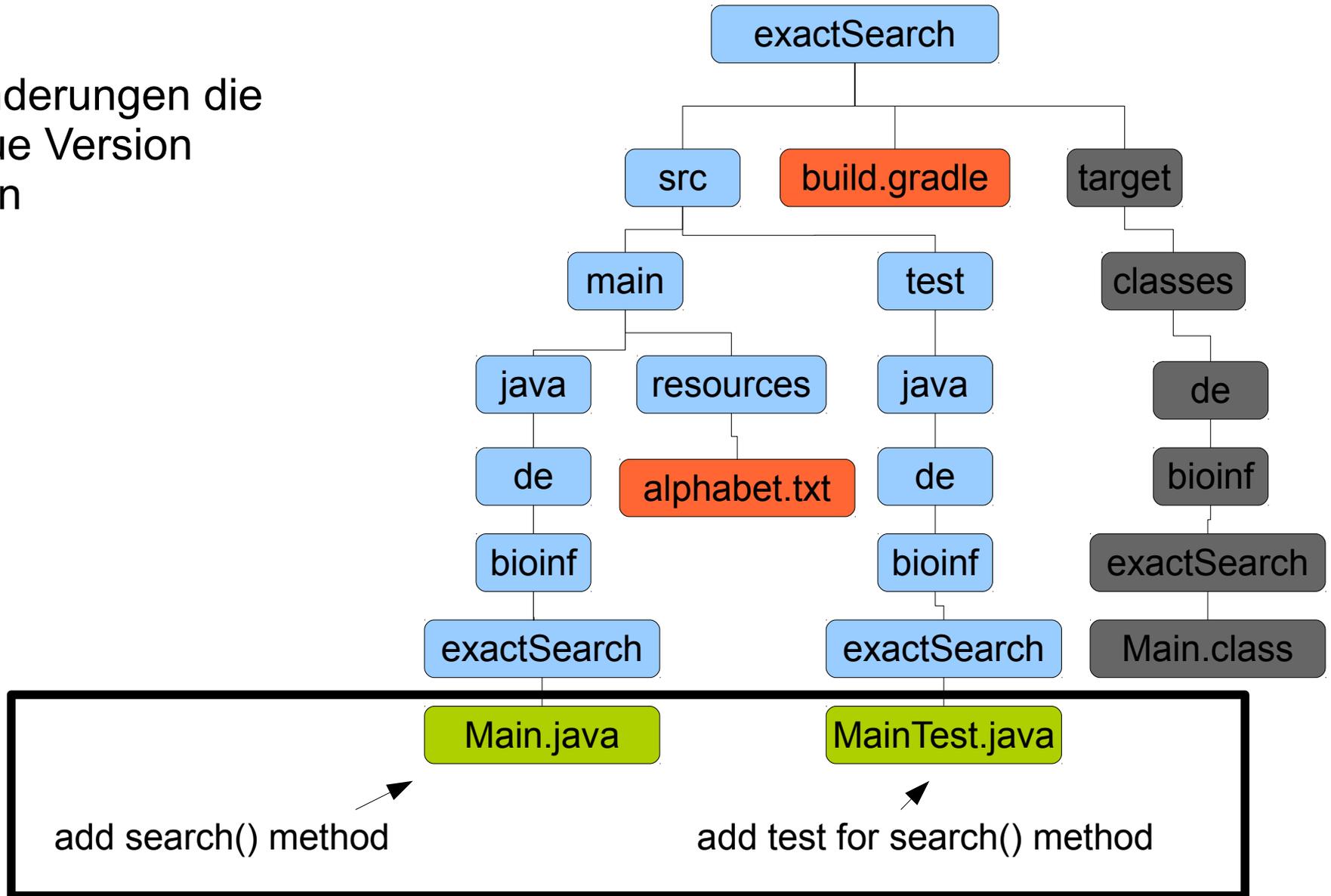
- die Gesamtheit aller versionierten Dateien innerhalb einer Version



# Grundlegende Begriffe

## Commit:

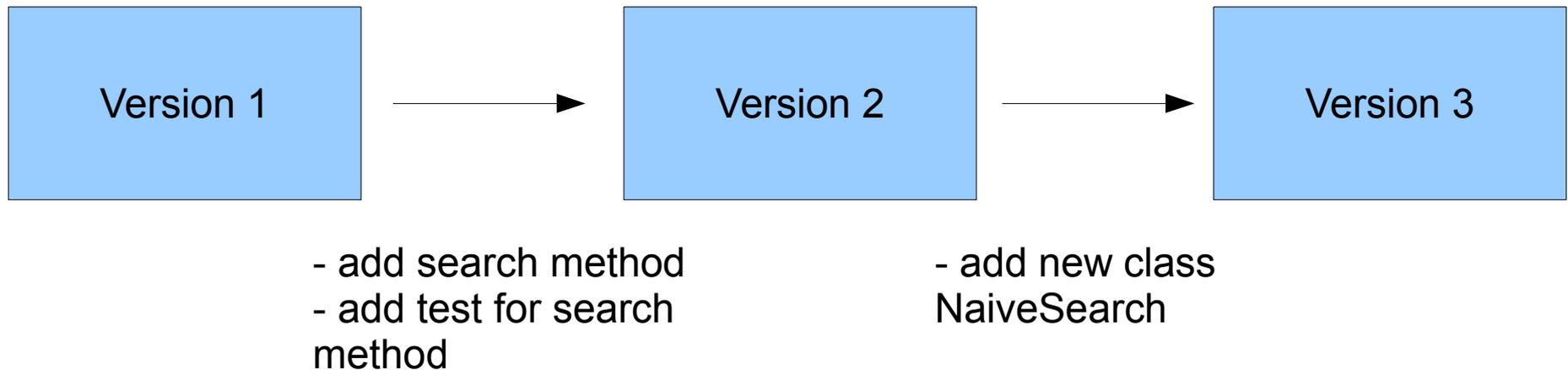
- Alle Änderungen die eine neue Version erzeugen



# Grundlegende Begriffe

## Commit:

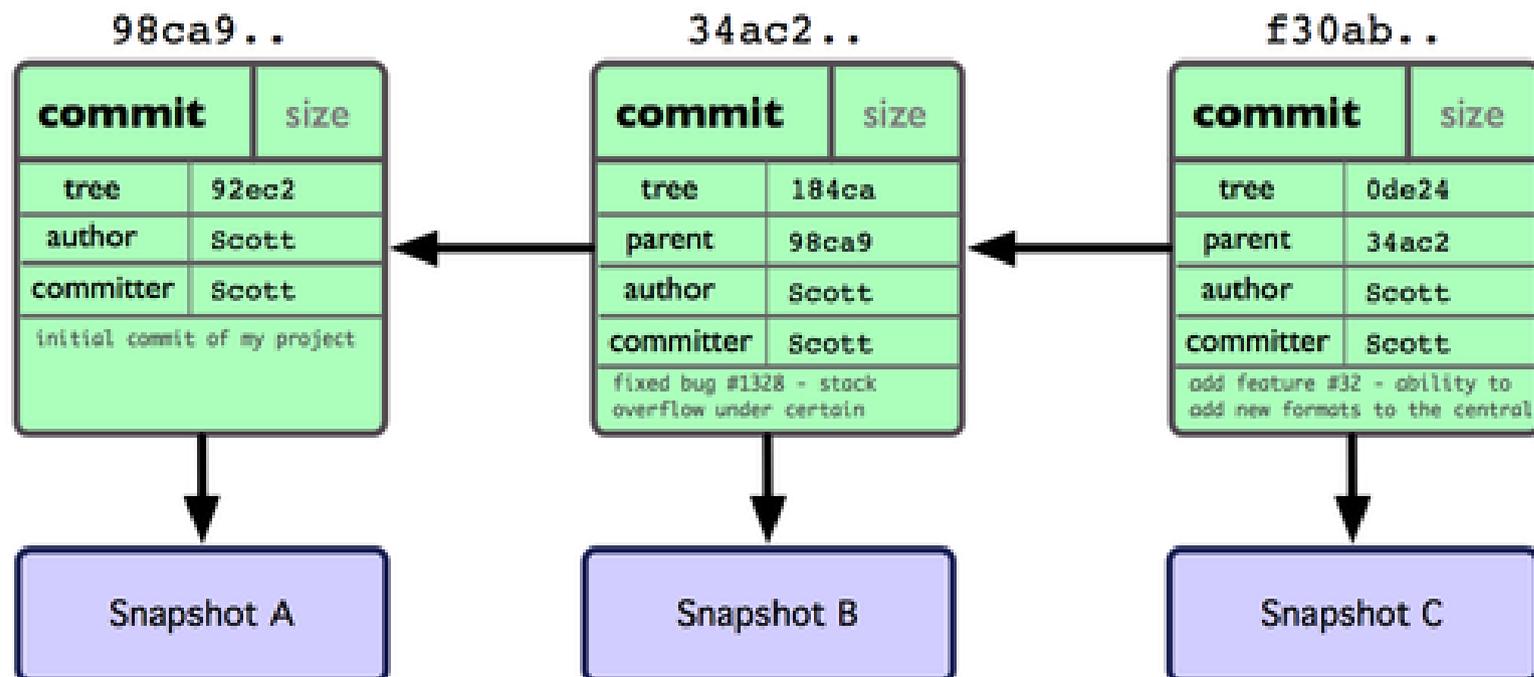
- Alle Änderungen die eine neue Version erzeugen



# Grundlegende Begriffe

## Commit:

- Alle Änderungen zwischen zwei Versionen



# Grundlegende Begriffe

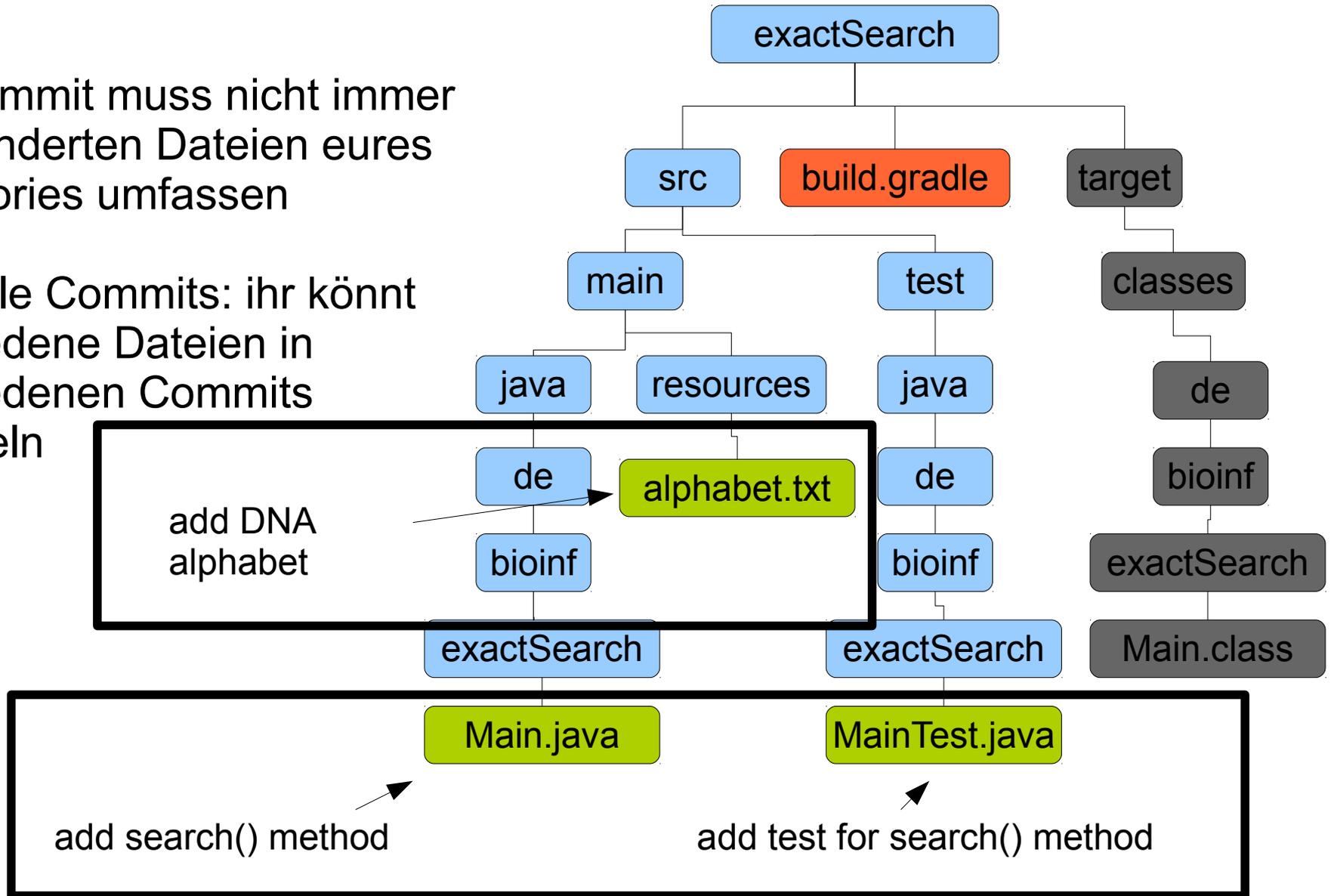
## **Commit:**

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

# Grundlegende Begriffe

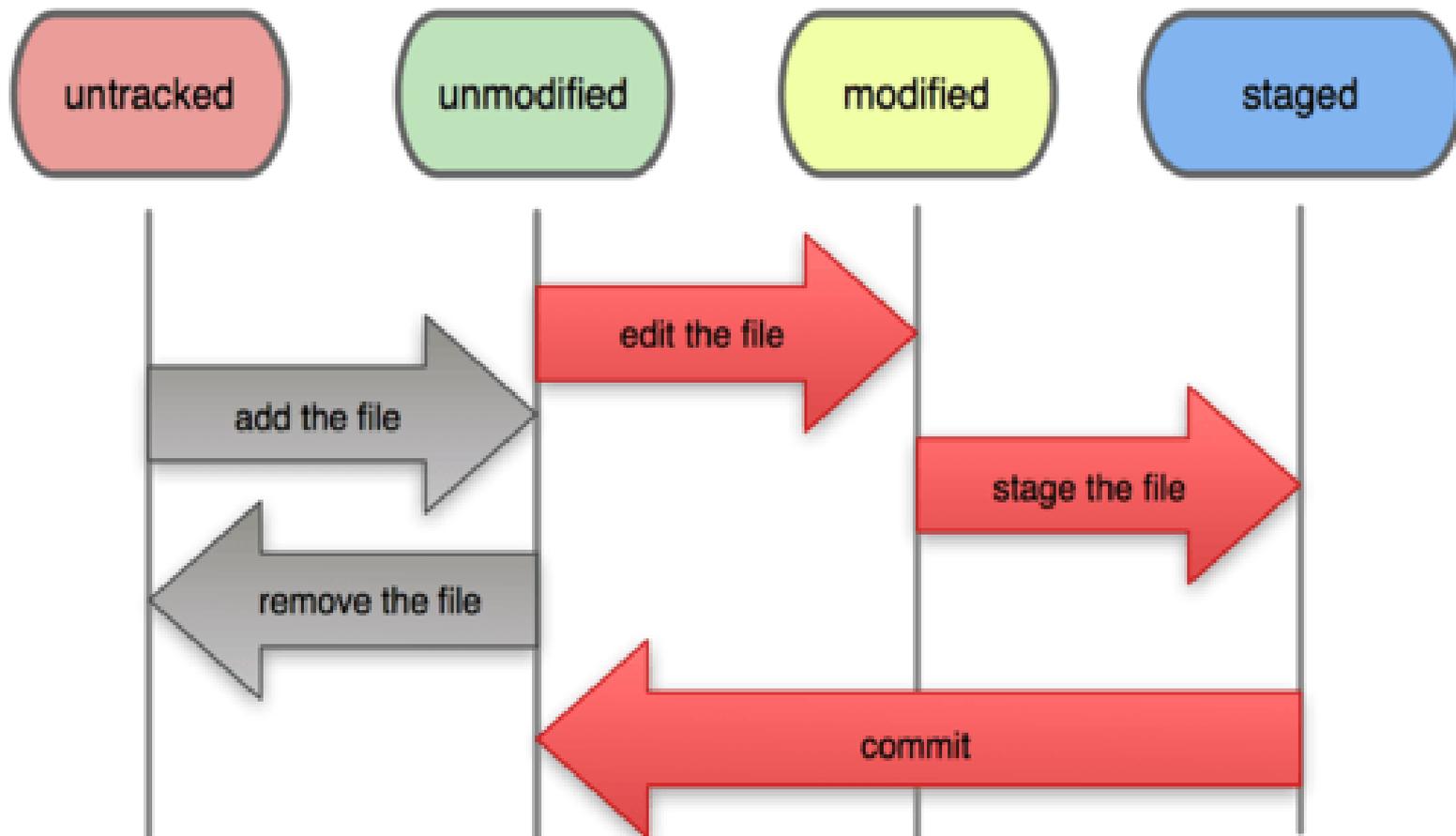
## Commit:

- 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

## File Status Lifecycle



# Grundlegende Begriffe

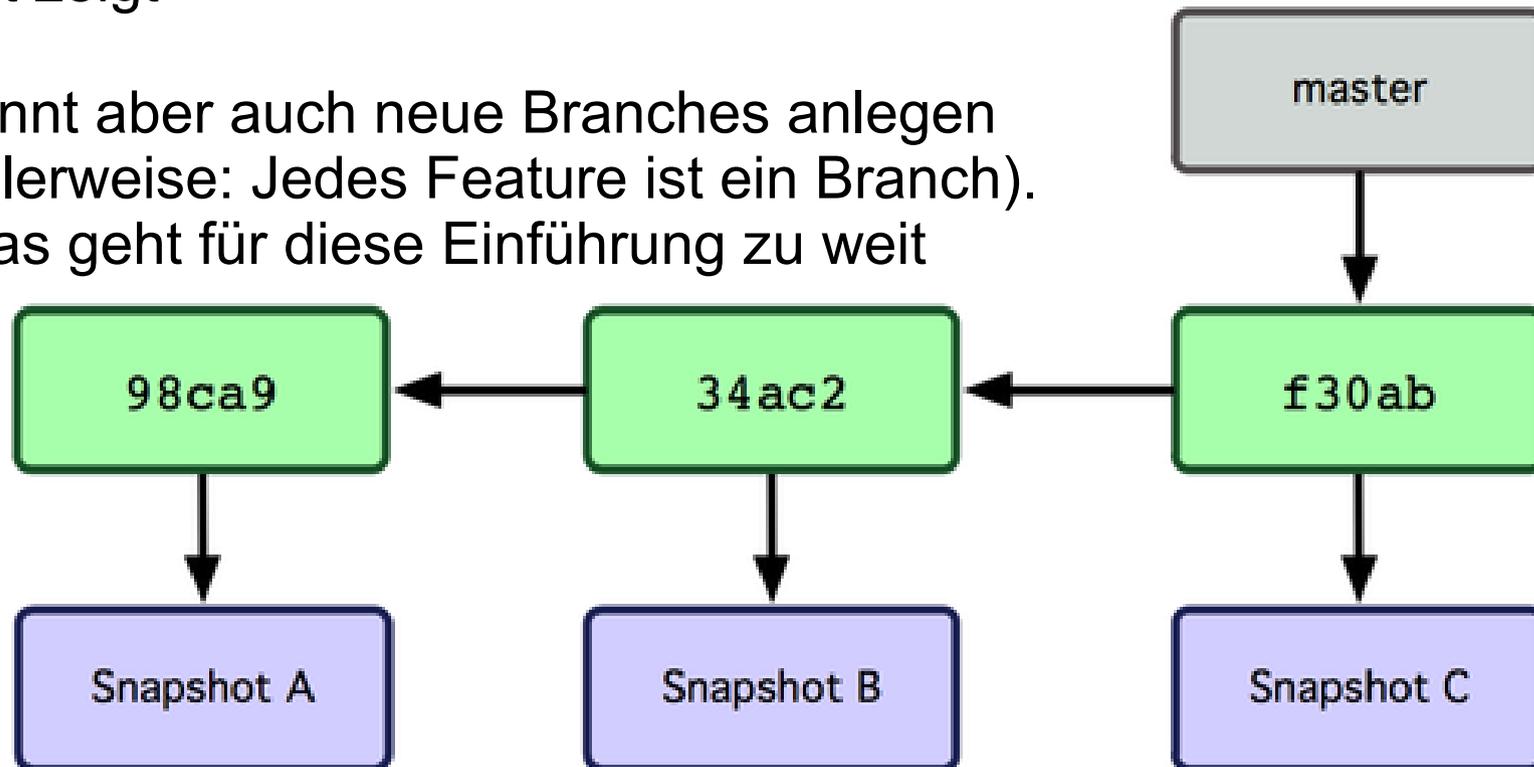
## Commit:

- 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
- Spezialfall: **bare** repository ist ein Repository in das jeder reinschreiben darf
- In Arbeitsrepository darf nur der Eigentümer reinschreiben

# Grundlegende Begriffe

## Remote Branches:

- jeder Branch existiert einmal lokal und einmal für jedes remote Repository
- wenn jemand Änderungen ins zentrale Repository einpflegt, landen diese im remote Branch
- wir müssen manuell sagen, dass wir diese Änderungen in unserem lokalen Branch haben wollen

# Grundlegende Begriffe

## Merge:

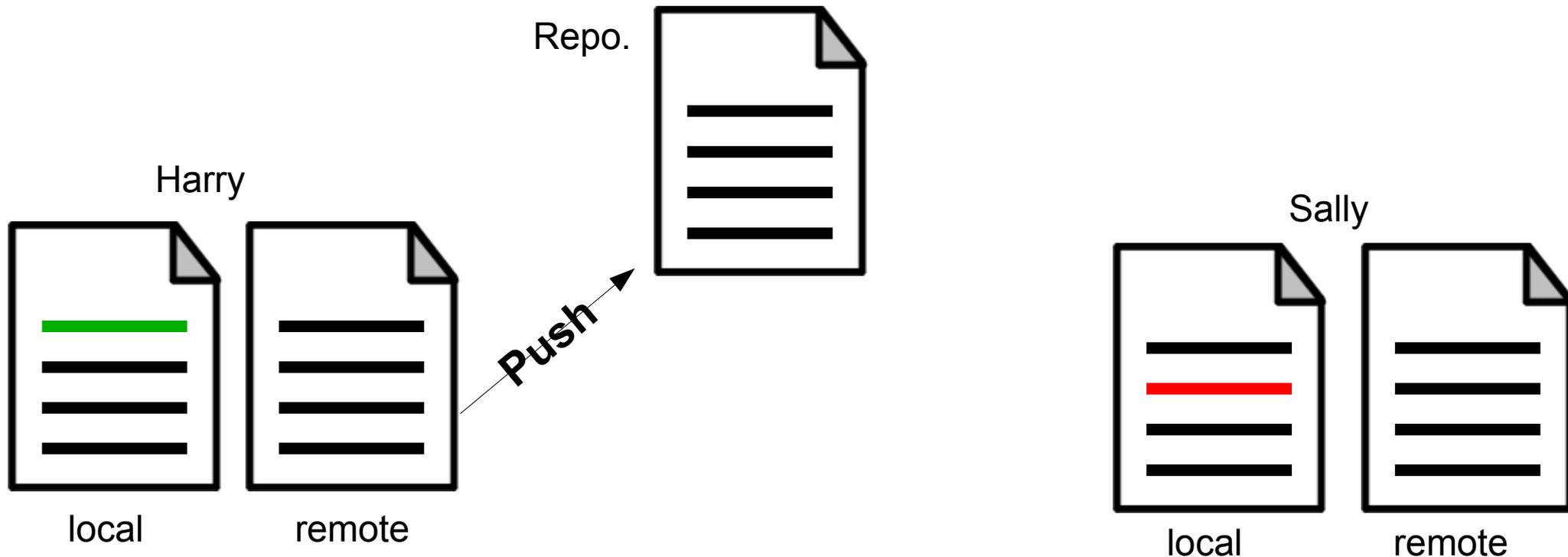
- Branches können **gemerged** werden
- Daraus entsteht ein neuer Commit, der alle Änderungen der Commits beider Branches beinhaltet
- Für uns nur wichtig: Wir können **remote Branches** und **lokale Branches** mergen um Änderungen vom Server in unser lokales Repository zu übernehmen



# Grundlegende Begriffe

## Push und Pull:

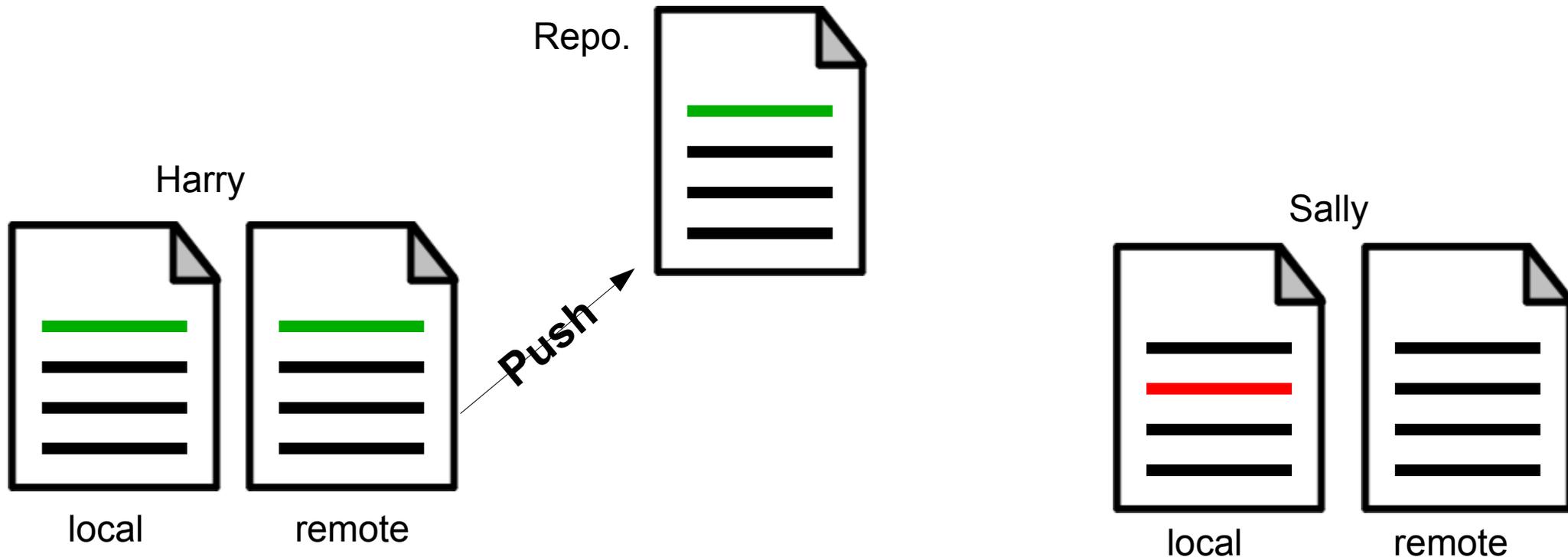
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

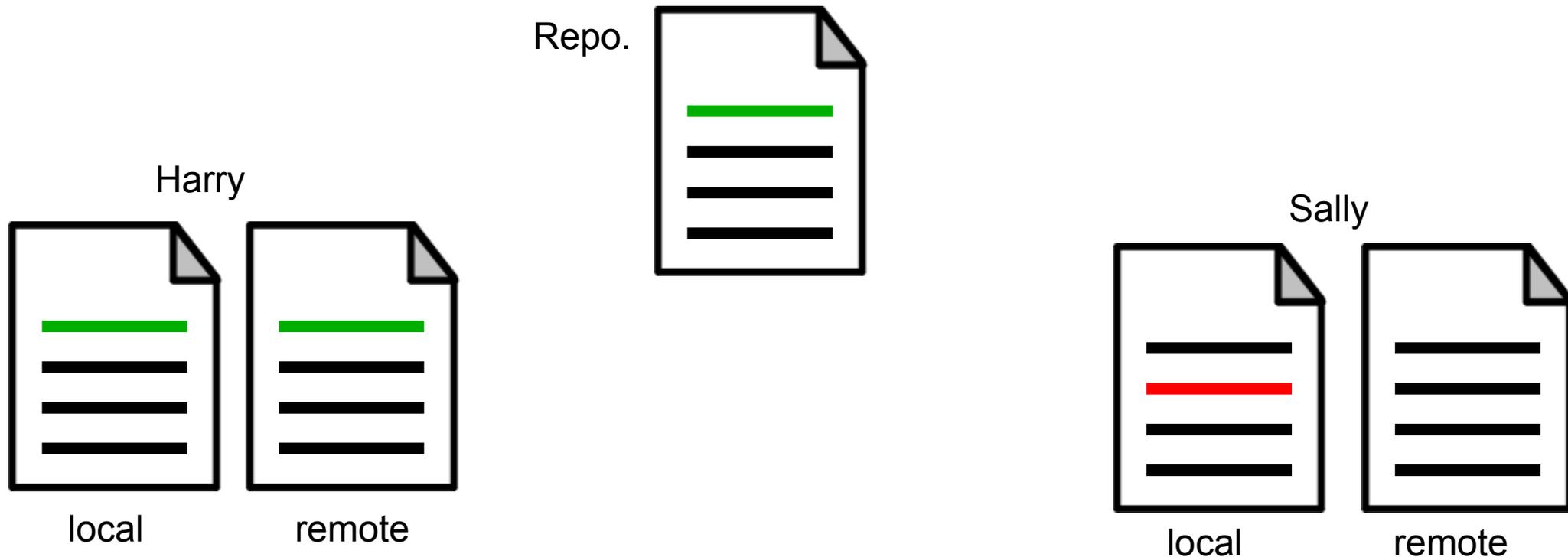
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

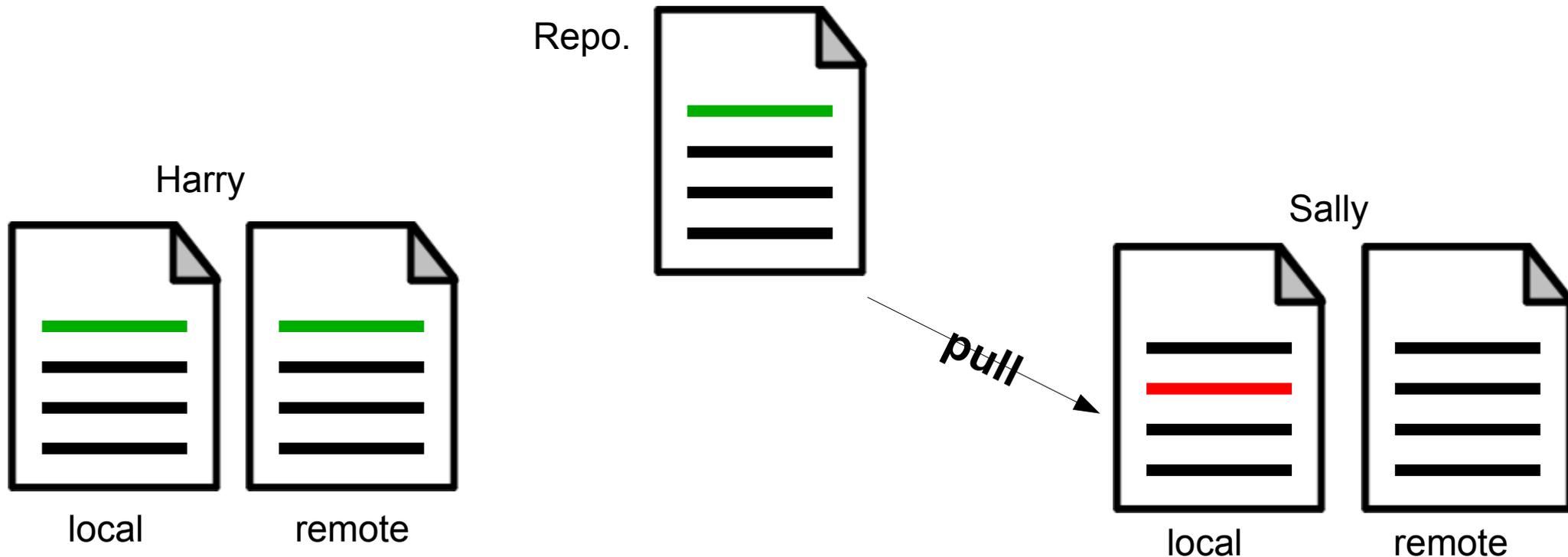
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

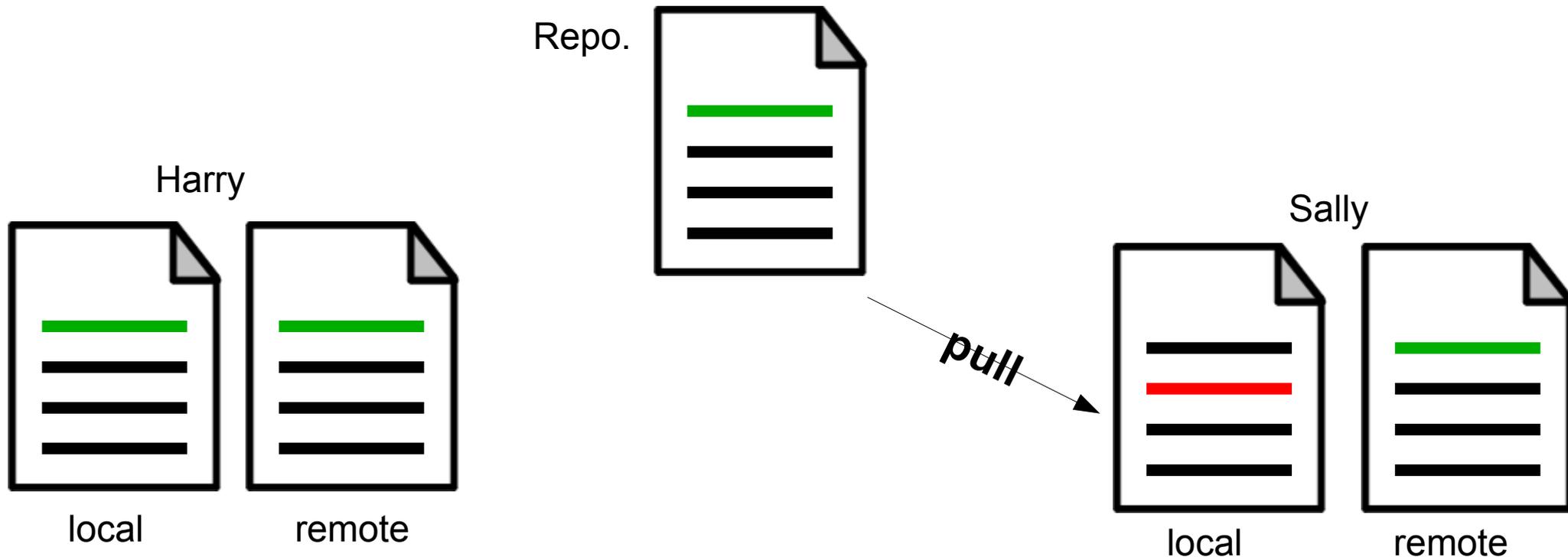
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

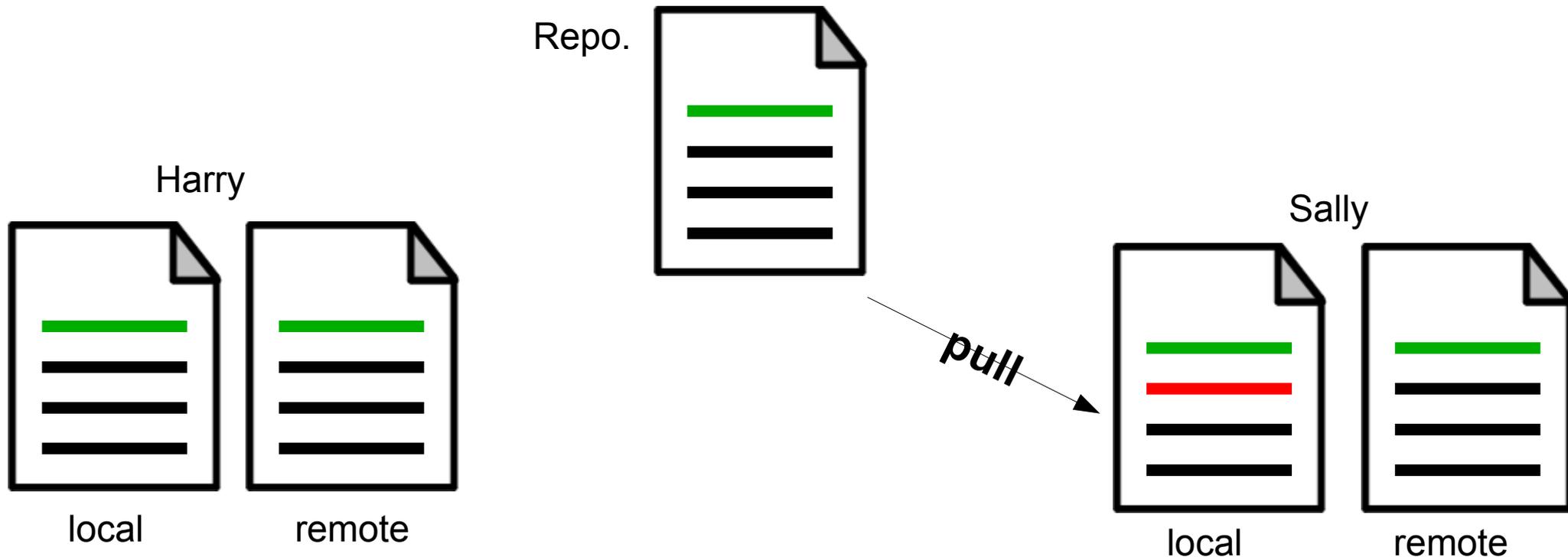
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

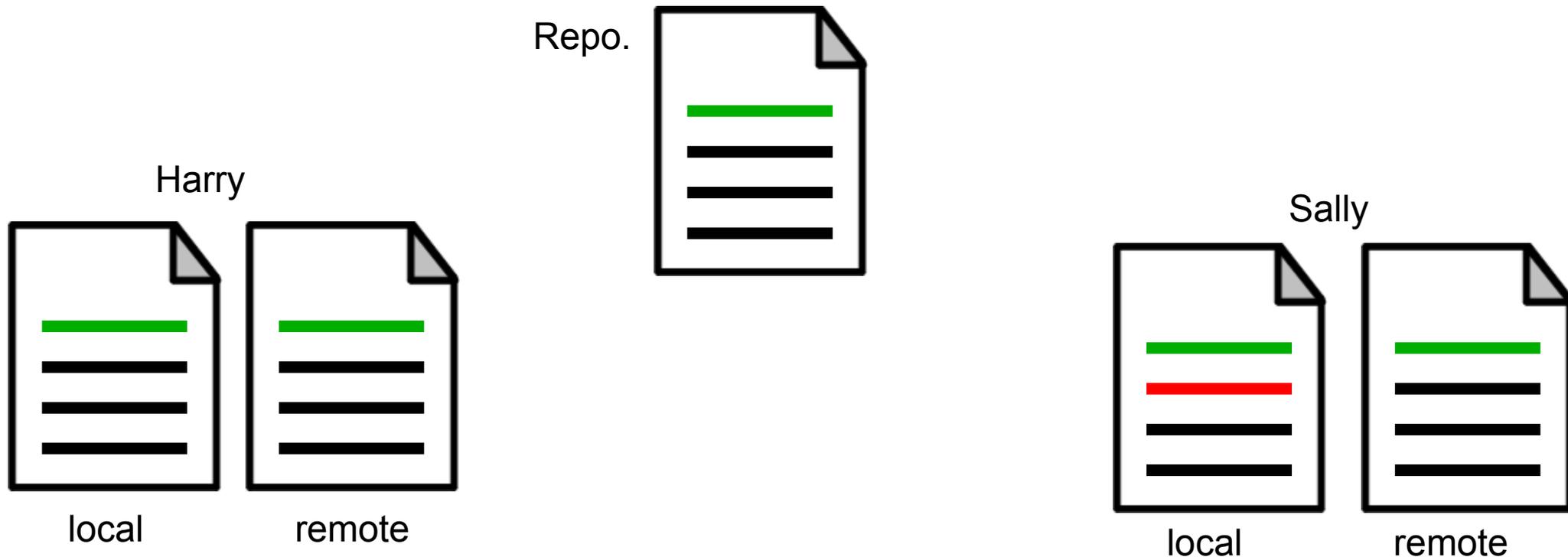
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

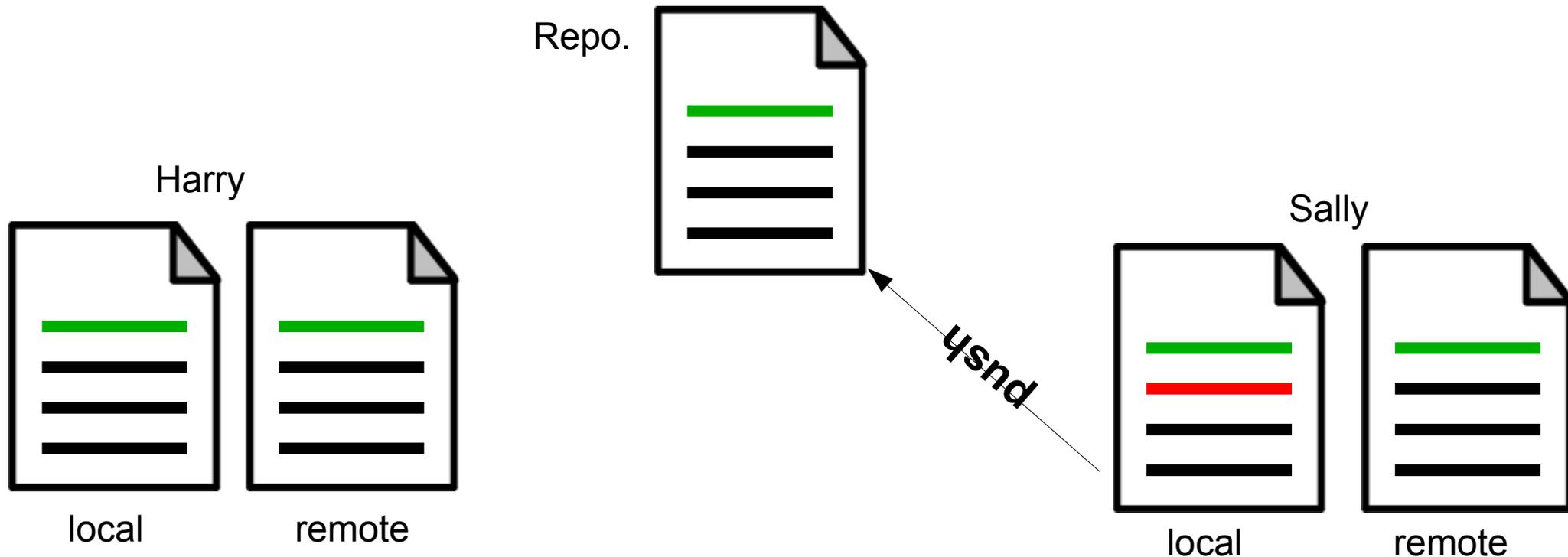
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

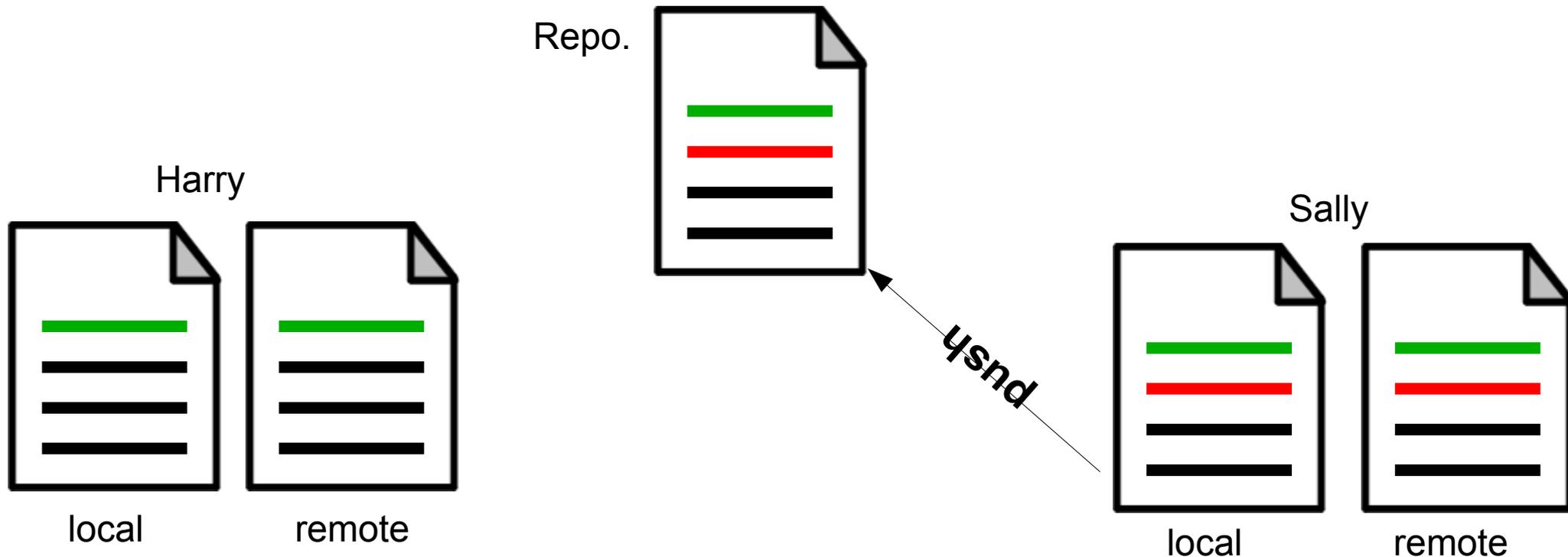
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

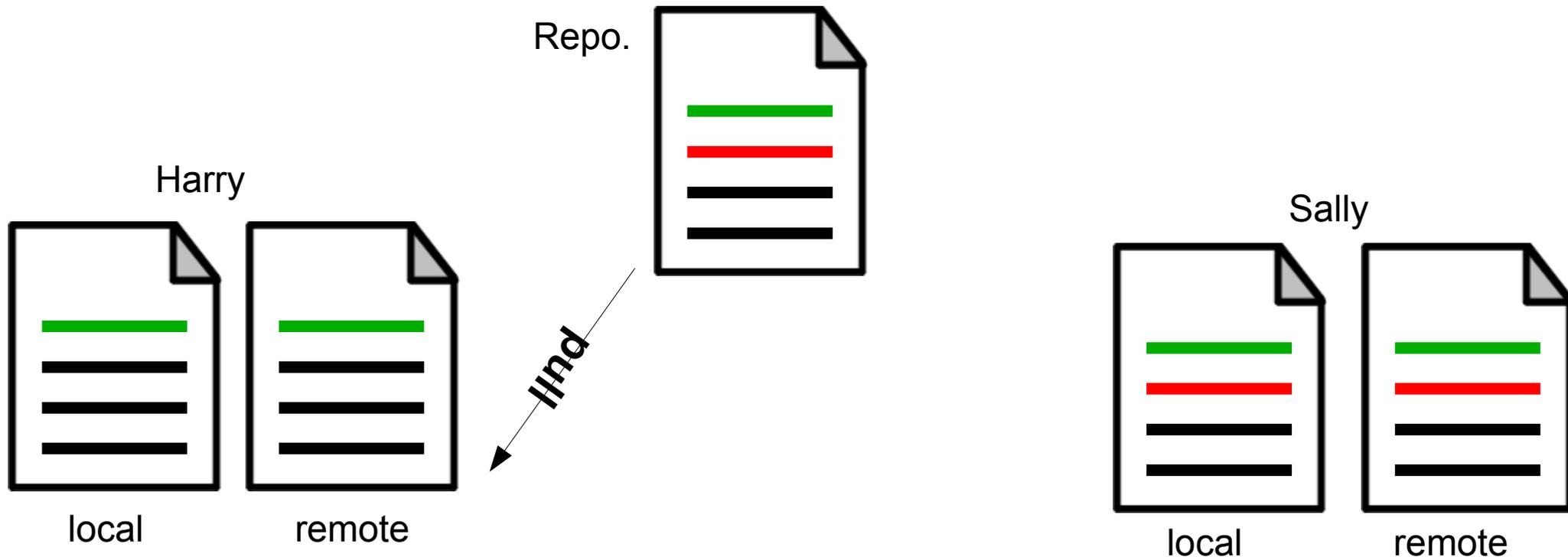
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

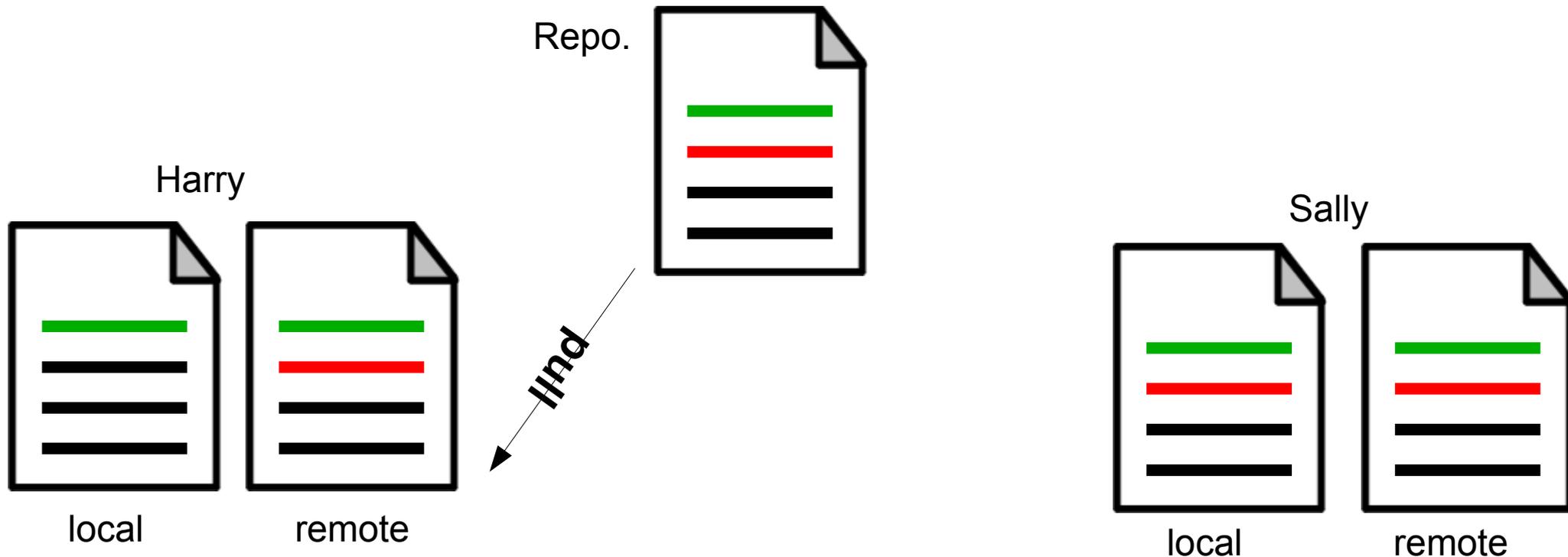
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

## Push und Pull:

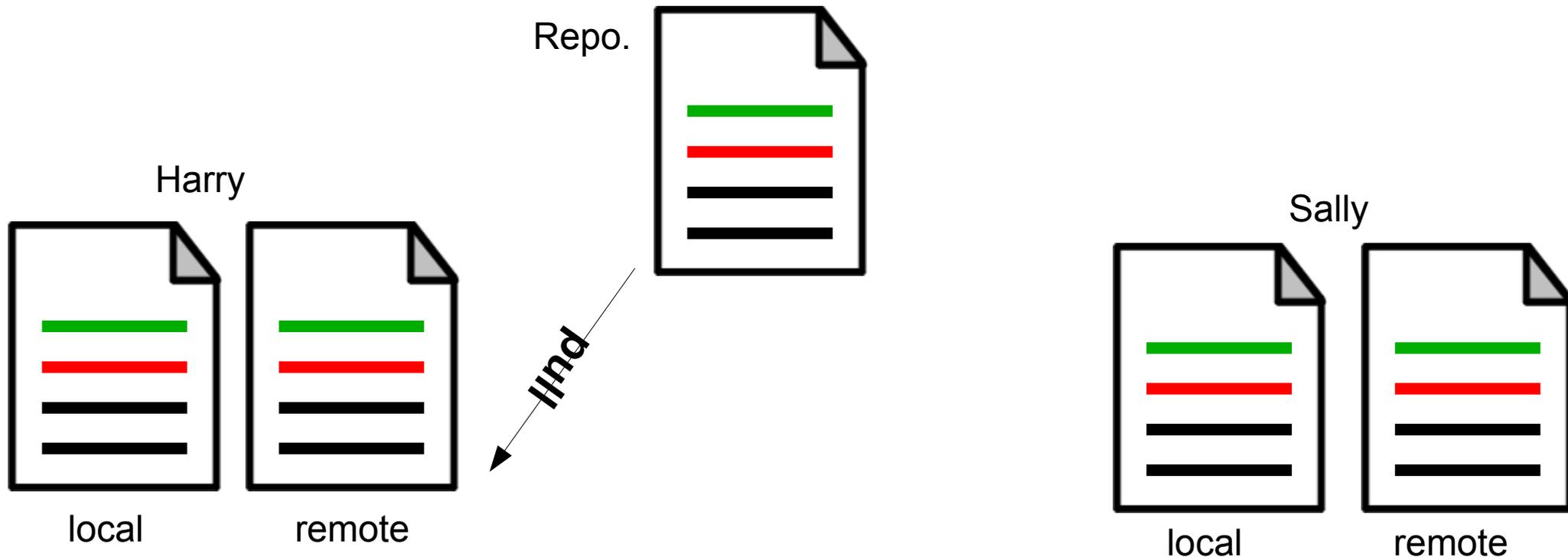
- Der **Push** Befehl merged Änderungen unseres Branches in den Branch des remote Repositories
- Der **Pull** Befehl merged Änderungen vom Remote Branch des remote Repositories in unseren lokalen Branch



# Grundlegende Begriffe

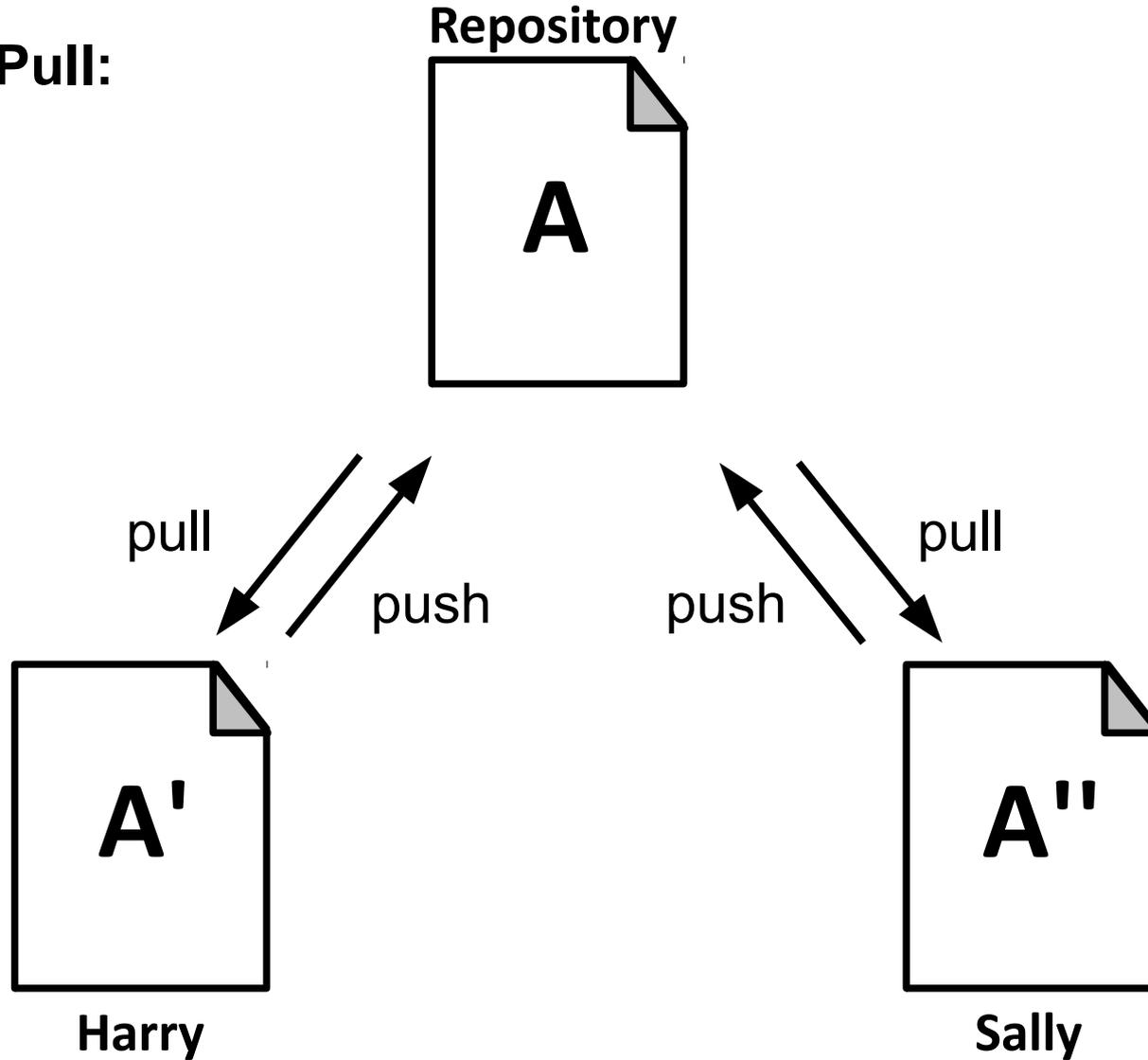
## Push und Pull:

- 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



# Grundlegende Begriffe

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

# git Repositories

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

## **Beispiel:**

```
ssh <nutzernamen>@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
- 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

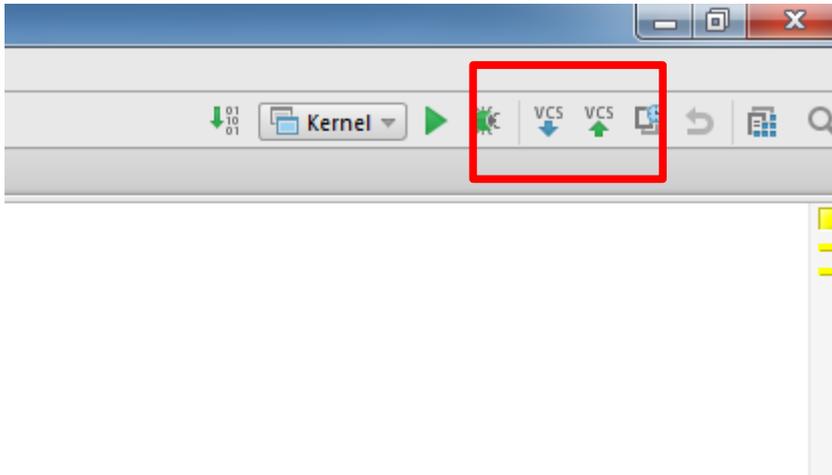
```
kaidu:linux$ git mergetool
```

- 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

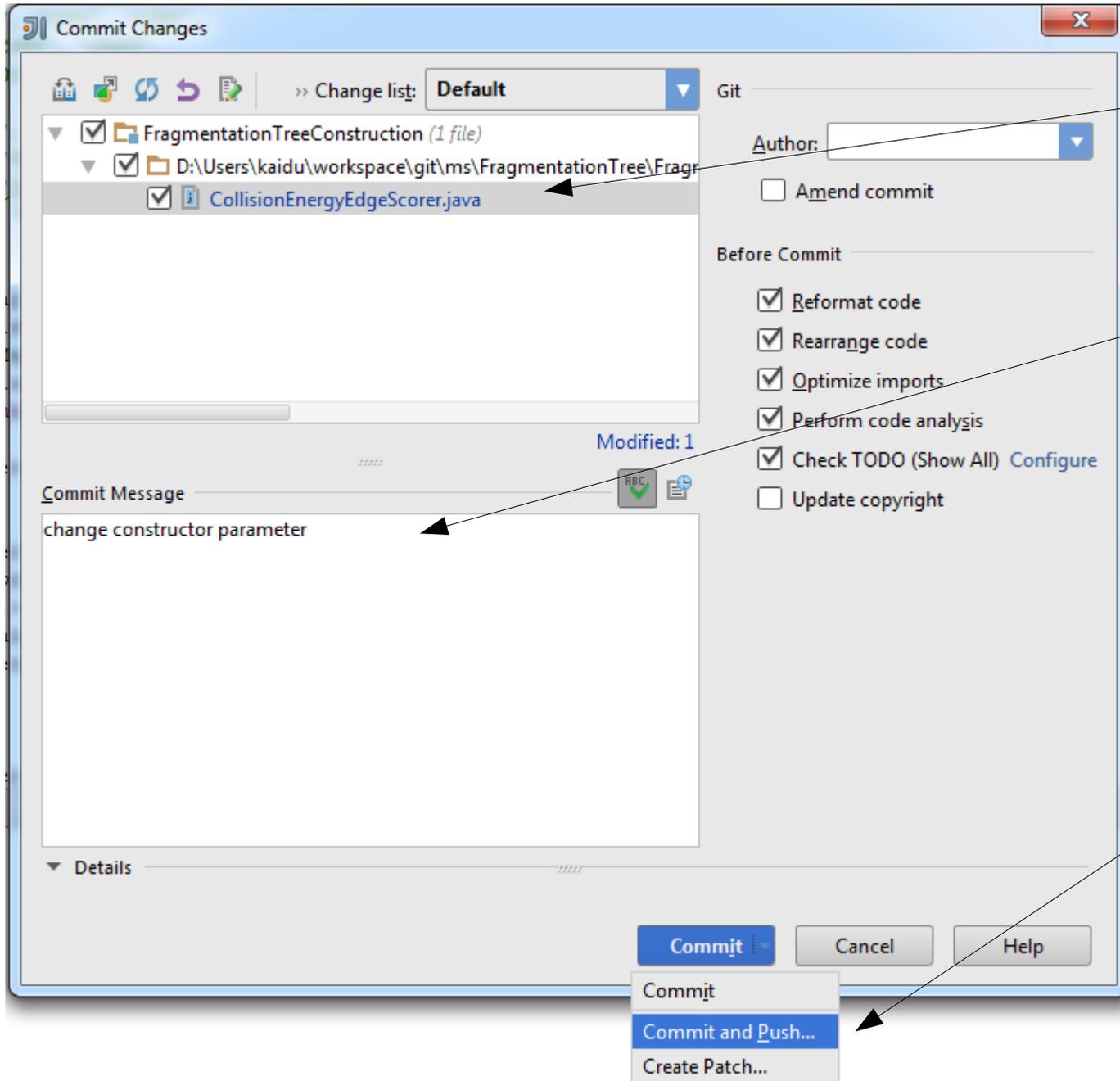


Lade Änderungen vom Server runter (pull)



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

# git und IntelliJ

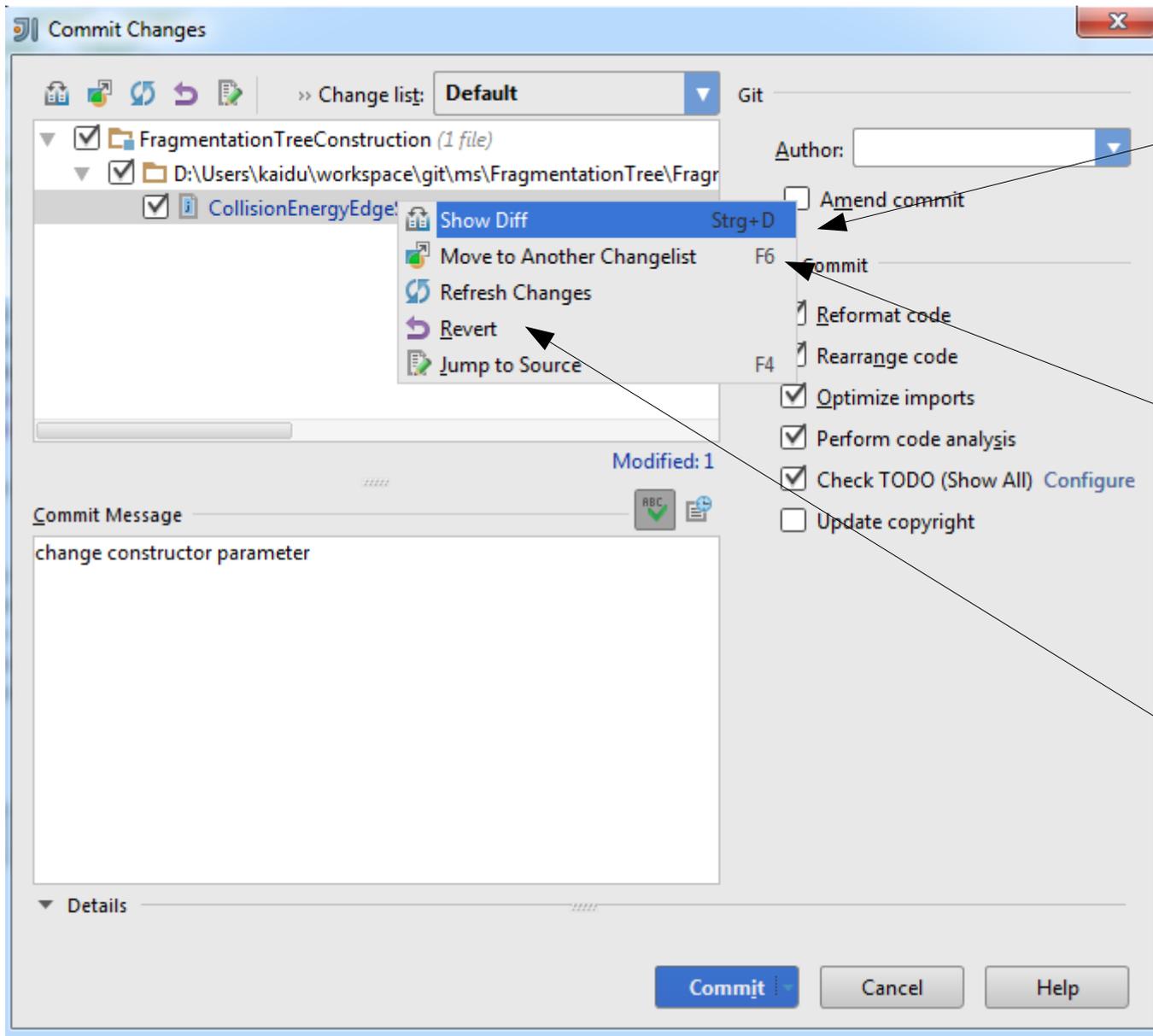


Checkbox aller geänderter Files die committed werden sollen

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

D:\Users\kaidu\workspace\git\ms\FragmentationTree\FragmentationTreeConstruction\src\main\java\de\unijena\bioinf\FragmentationTreeConstruction\computation\scoring\CollisionEnergyEdgeScorer.java

Ignore whitespace: Do not ignore Highlight: By word Include into commit

819538b6c225b03123691790977bb689004f7d1e (Read-only)

```
public CollisionEnergyEdgeScorer(double alpha, double beta) {
    this.alpha = alpha;
    this.beta = beta;
    this.logAlpha = Math.log(alpha);
    this.logBeta = Math.log(beta);
}

public double getAlpha() {
    return alpha;
}

public double getBeta() {
    return beta;
}

public void setAlpha(double alpha) {
    this.alpha = alpha;
    this.logAlpha = Math.log(alpha);
}

public void setBeta(double beta) {
    this.beta = beta;
    this.logBeta = Math.log(beta);
}

/*
public Object prepare(ProcessedInput input, FragmentationGraph graph) {
    this.logBeta = Math.log(beta);
}

public String foo() {
    return "foo";
}

public double getAlpha() {
    return alpha;
}

public double getBeta() {
    return beta;
}

public void setAlpha(double alpha) {
    this.alpha = alpha;
    this.logAlpha = Math.log(alpha);
}

/*
public Object prepare(ProcessedInput input, FragmentationGraph graph) {
    final int n = input.getMergedPeaks().size();
    final int[] minEnergy = new int[n], maxEnergy = new int[n];
    for (int i=0; i < input.getMergedPeaks().size(); ++i) asser
    Arrays.fill(minEnergy, -1); Arrays.fill(maxEnergy, -1);
    final Ms2Spectrum[] spectra = input.getOriginalInput().getM
```

Your version

39  
40  
41  
42 X  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64

Commit Message

change constructor parameter

3 differences Deleted Changed Inserted

# git und IntelliJ

The screenshot displays the IntelliJ IDEA interface with a diff view of the file `CollisionEnergyEdgeScorer.java`. The interface is split into two panes: the left pane shows the previous snapshot, and the right pane shows the current working copy. The diff is color-coded: deleted lines are grey, changed lines are blue, and inserted lines are green. A commit message field is visible at the bottom, and a legend at the bottom indicates the color coding for differences.

```
819538b6c225b03123691790977bb689004f7d1e (Read-only)
public CollisionEnergyEdgeScorer(double alpha, double beta) {
    this.alpha = alpha;
    this.beta = beta;
    this.logAlpha = Math.log(alpha);
    this.logBeta = Math.log(beta);
}

public double getAlpha() {
    return alpha;
}

public void setAlpha(double alpha) {
    this.alpha = alpha;
    this.logAlpha = Math.log(alpha);
}

public void setBeta(double beta) {
    this.beta = beta;
    this.logBeta = Math.log(beta);
}

/*
public Object prepare(ProcessedInput input, FragmentationGraph graph) {
    this.logBeta = Math.log(beta);
}

public String foo() {
    return "foo";
}

public double getAlpha() {
    return alpha;
}

public void setAlpha(double alpha) {
    this.alpha = alpha;
    this.logAlpha = Math.log(alpha);
}

/*
public Object prepare(ProcessedInput input, FragmentationGraph graph) {
    final int n = input.getMergedPeaks().size();
    final int[] minEnergy = new int[n], maxEnergy = new int[n];
    for (int i=0; i < input.getMergedPeaks().size(); ++i) asser
    Arrays.fill(minEnergy, -1); Arrays.fill(maxEnergy, -1);
    final Ms2Spectrum[] spectra = input.getOriginalInput().getM
```

gelöschte Zeilen in grau

Hinzugefügte Zeilen in grün

links der letzte Snapshot

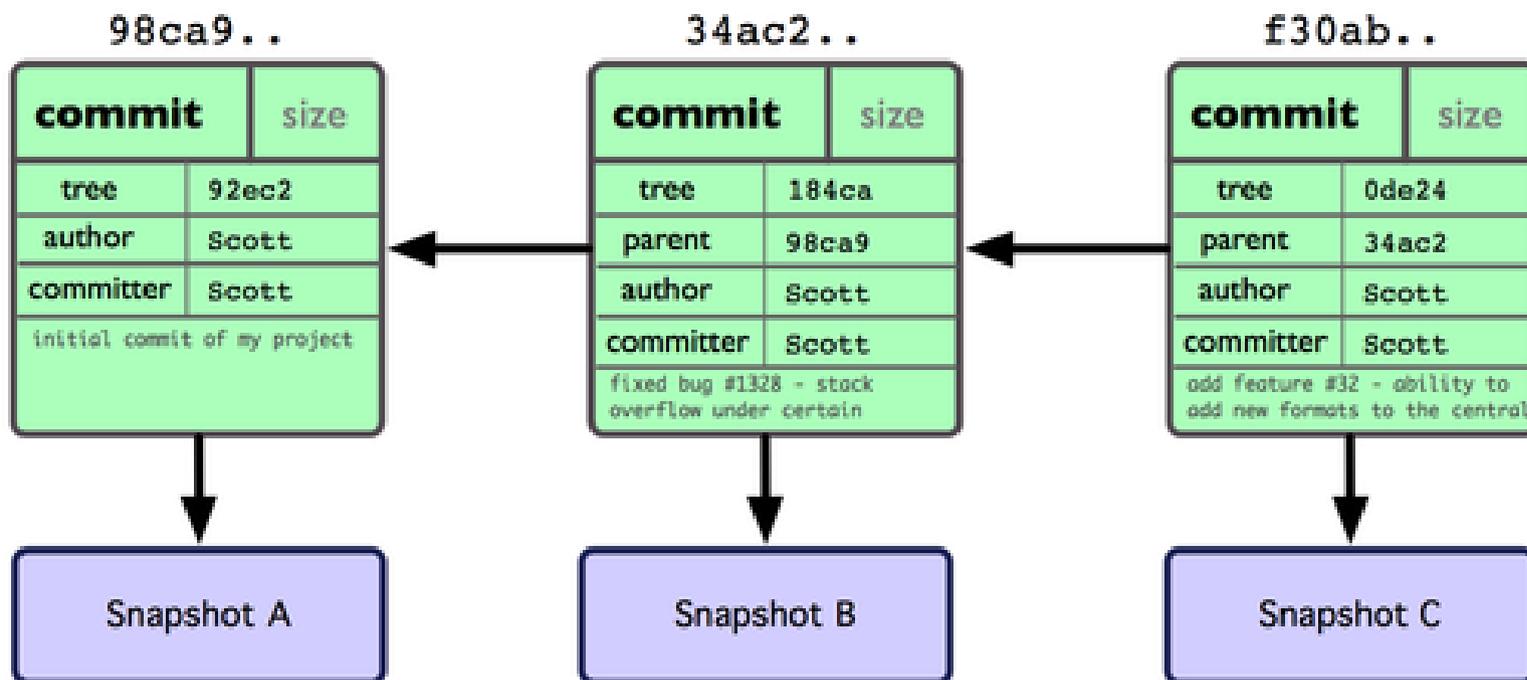
rechts die aktuelle Arbeitskopie

3 differences Deleted Changed Inserted

# Tipps für git

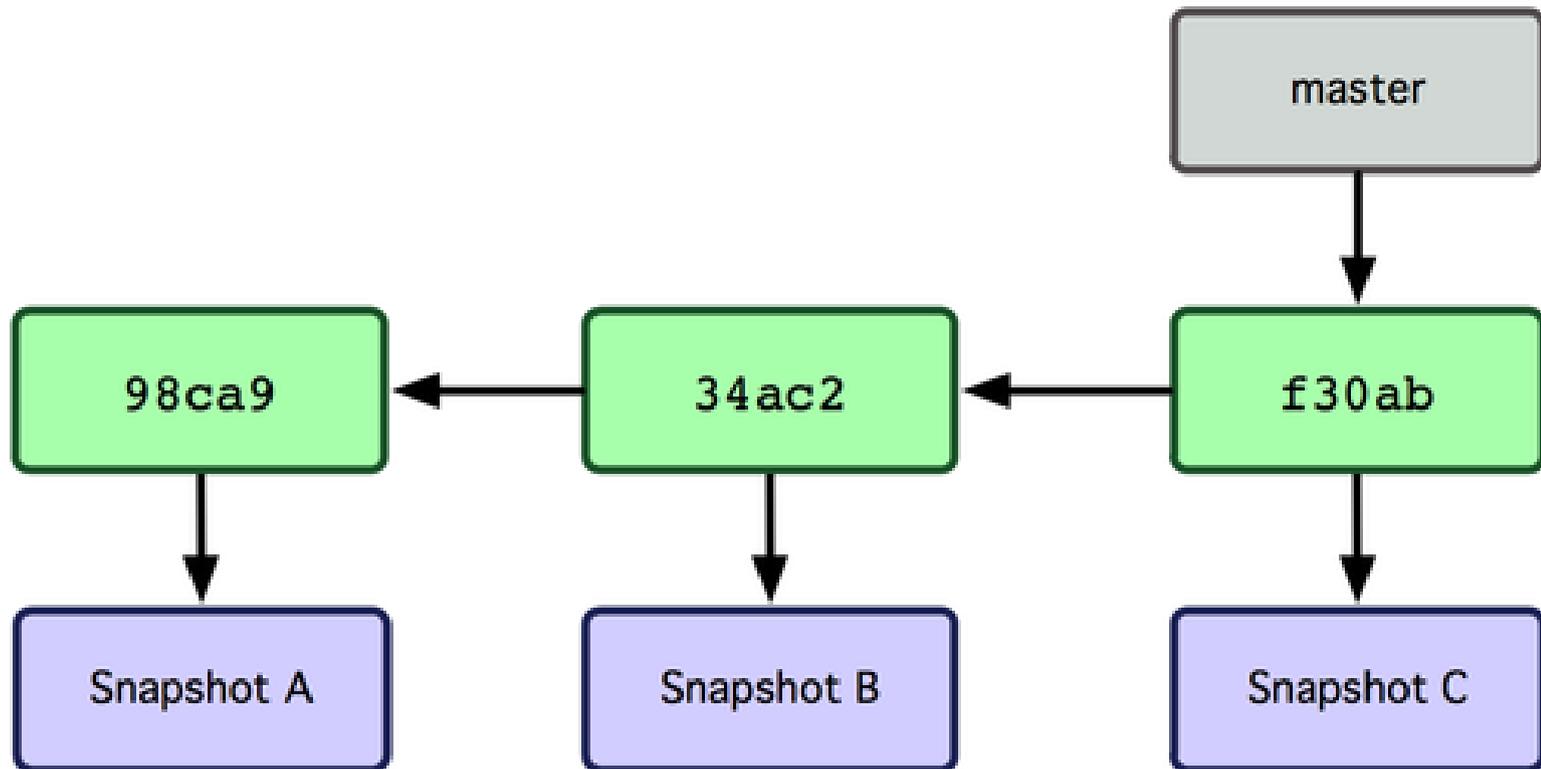
- <http://git-scm.com/> <--- lesen!
- Vermeide Binaries im Repository
  - Ausnahme: Icons für GUIs usw.
- Verwende einfache Pfadnamen
- Achtung: Windows kann nicht zwischen Groß/Kleinschreibung unterscheiden)
- Verwende getrennte Ordner für den Sourcecode und die Ausgabe
  - Die Ausgabe gehört nicht ins Repository

# Exkurs: Branches



# Exkurs: Branches

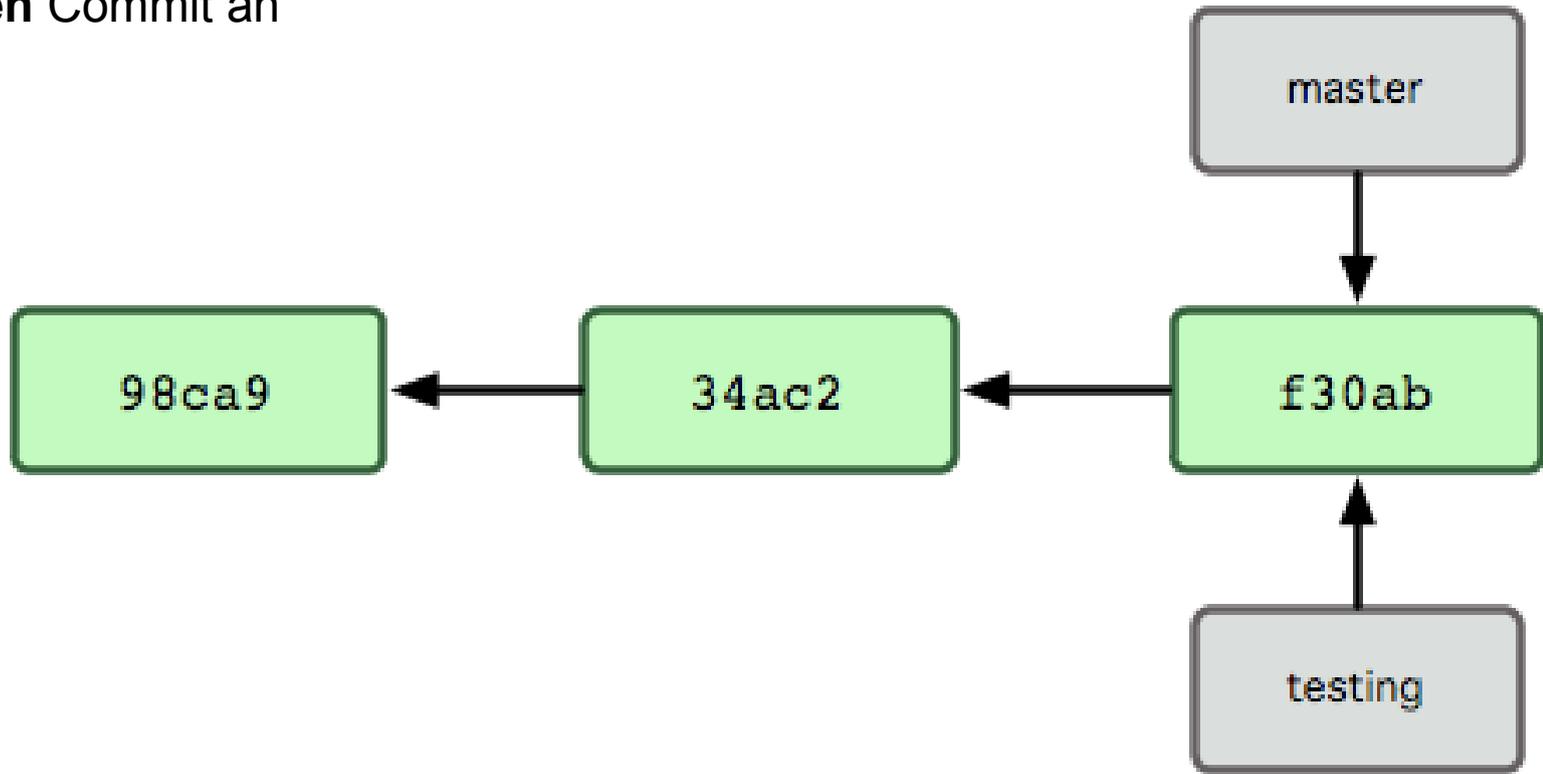
- ein Branch ist ein **Zeiger** auf einen Commit



# Exkurs: Branches

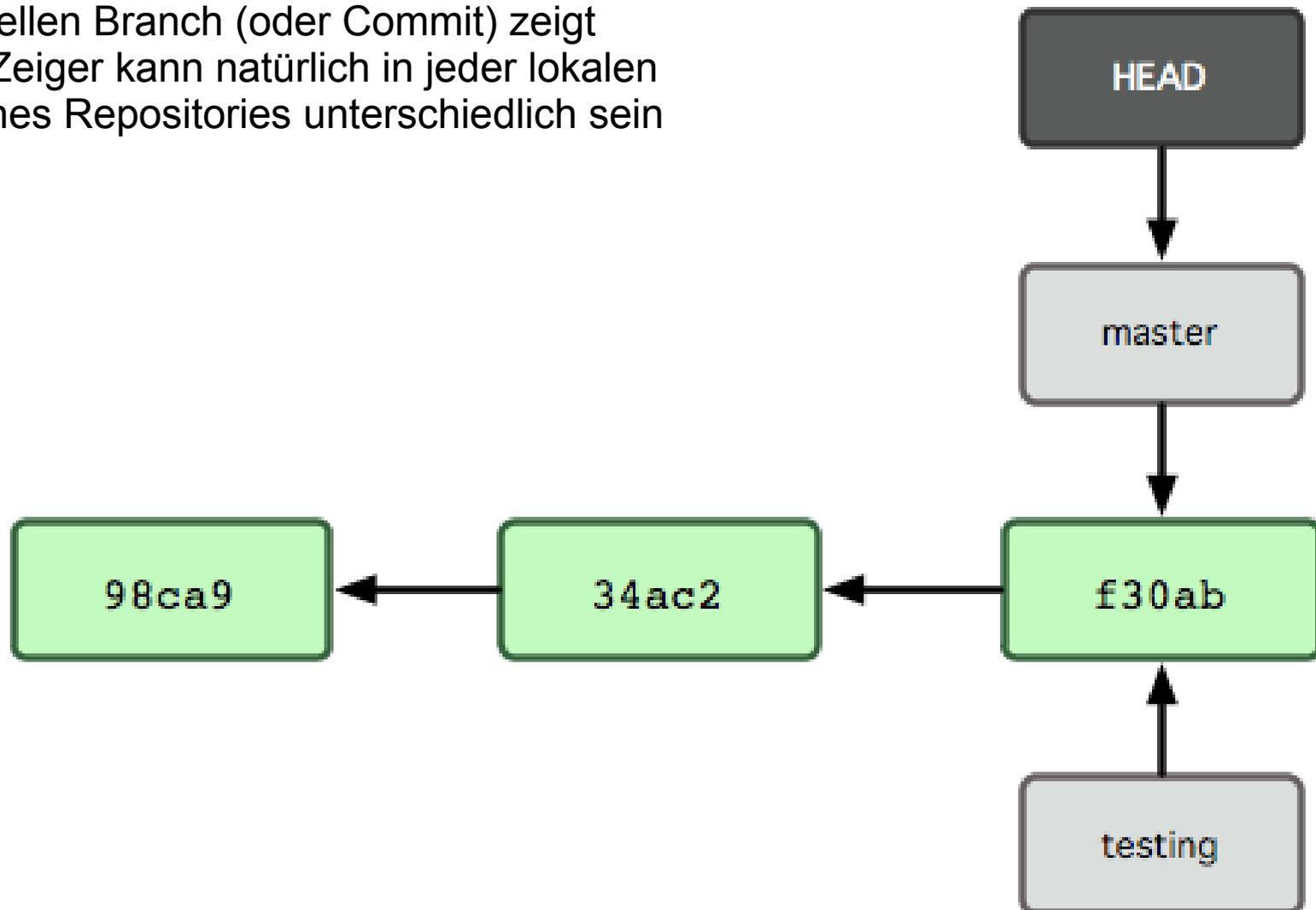
```
kaidu:linux$ git branch testing
```

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

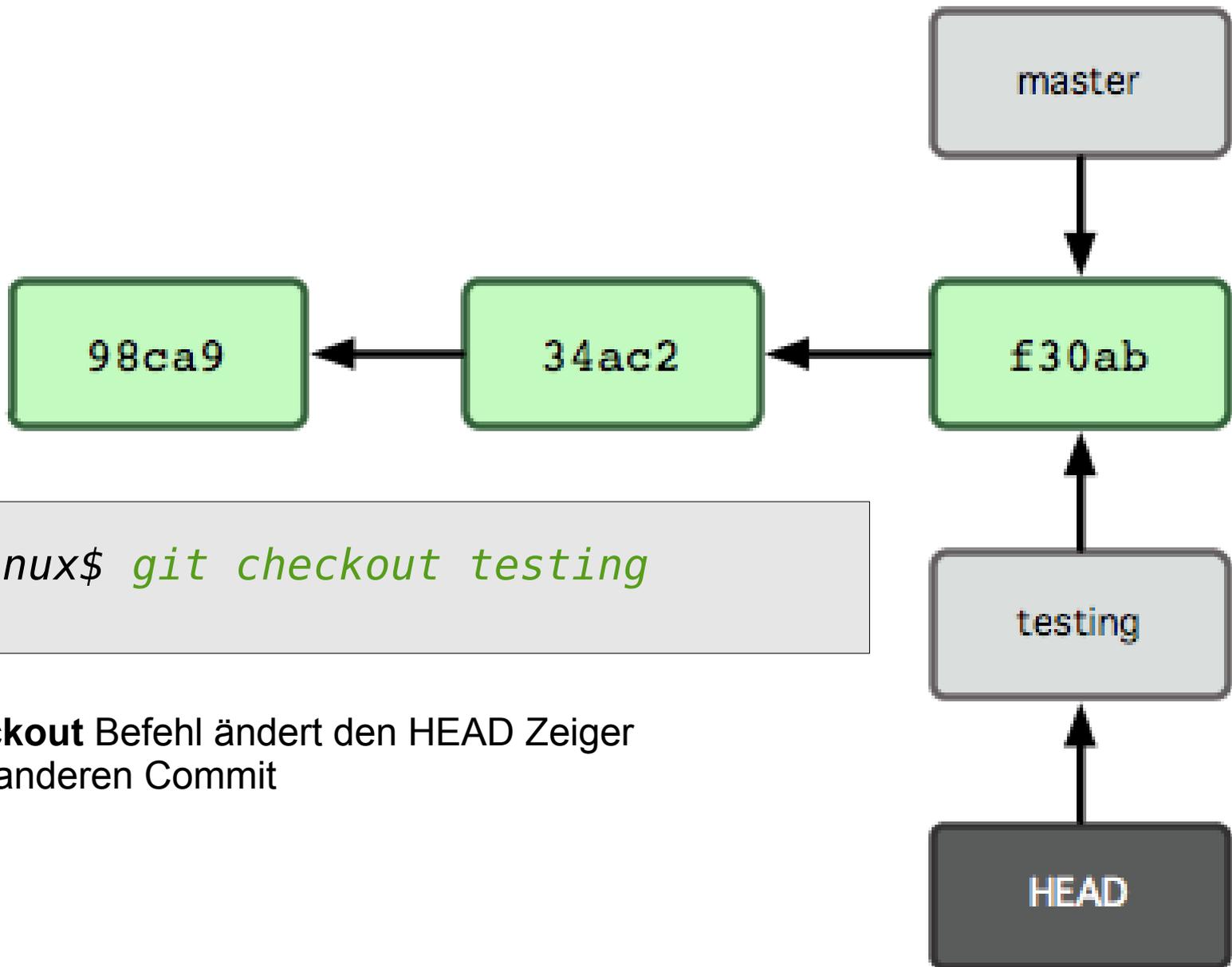


# Exkurs: Branches

- **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



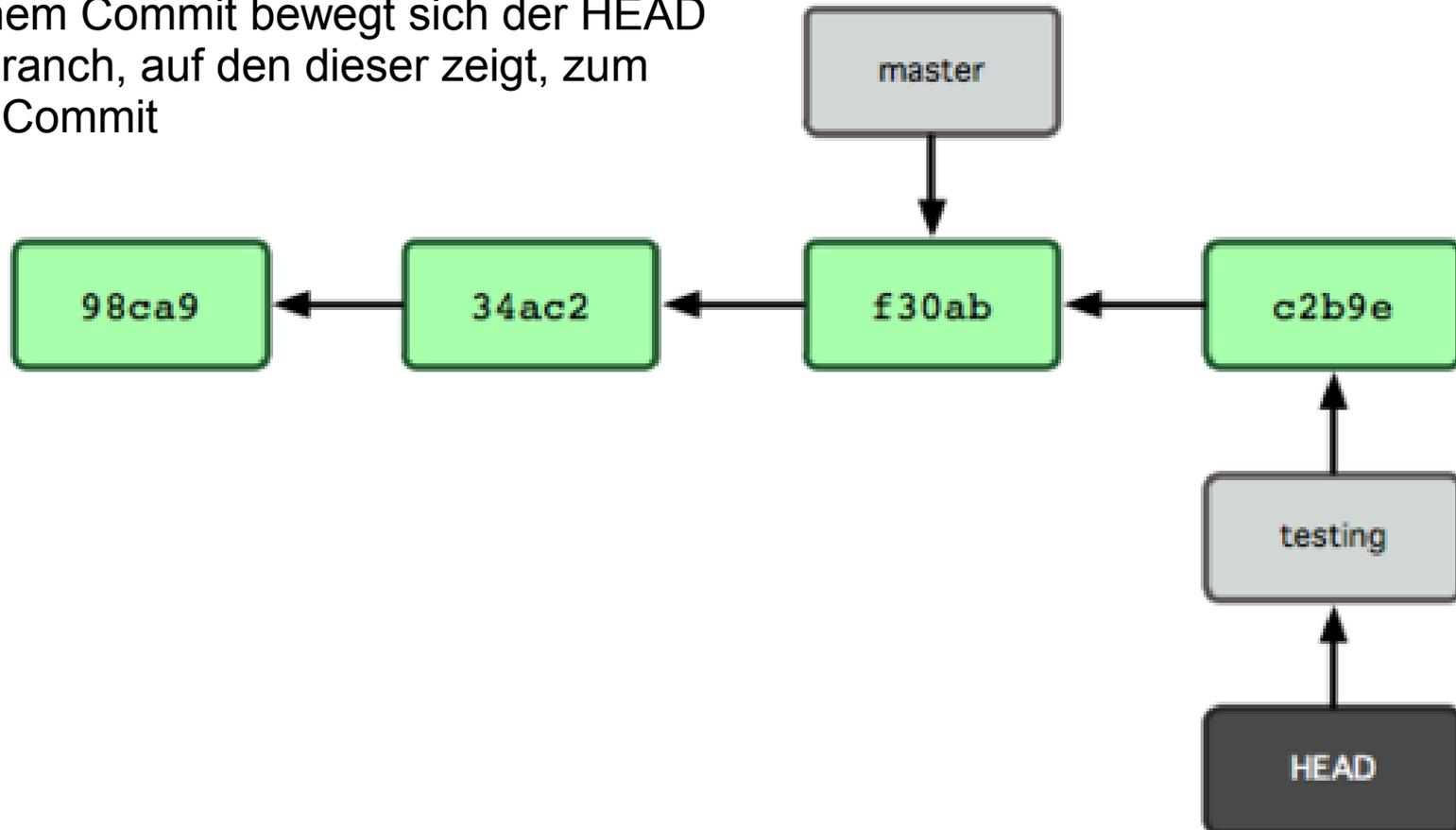
```
kaidu:linux$ git checkout testing
```

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

# Exkurs: Branches

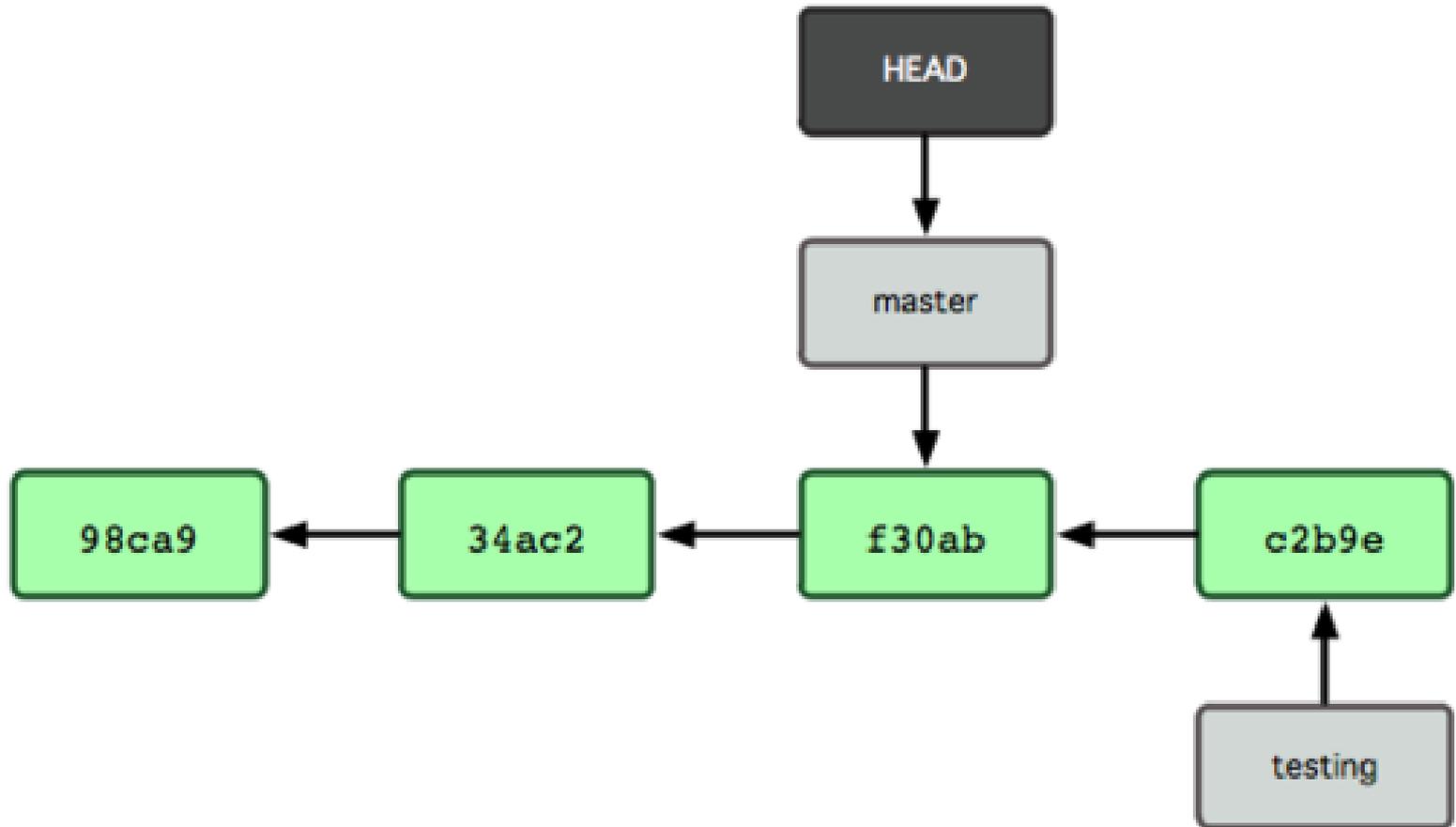
```
kaidu:linux$ git commit -am '...'
```

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



# Exkurs: Branches

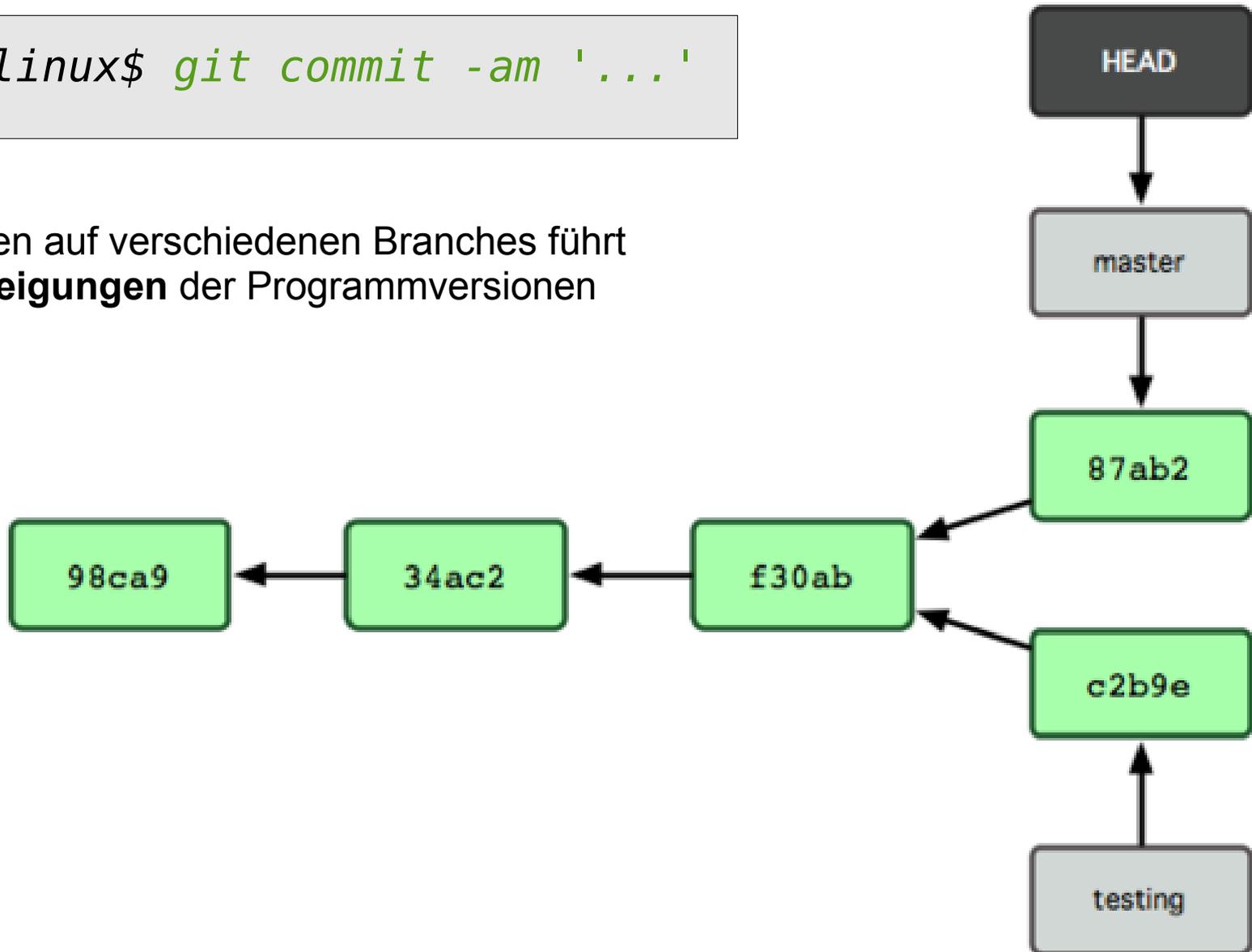
```
kaidu:linux$ git checkout master
```



# Exkurs: Branches

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

- Committen 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 abgibt, 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