Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

SOFTWARETECHNIK

Teil 1

Letztes Update: 19. Oktober 2019

Dr. Eva-Maria Iwer

Fachbereich Design Informatik Medien (DCSM)
Hochschule **RheinMain**

GLIEDERUNG

1. Versionierung

2. Abkürzungen

VERSION

Versionierung

Version	Datum	Kommentare
1.0	April 2019	Initial
1.1	Mai 2019	Fehlerbehebung
1.2	Oktober 2019	Erweiterung Branches

VERSIONIERUNG

VERSIONIERUNG

Was ist Versionskontrolle und warum es so wichtig ist!

→ Versionsverwaltungssystem (VCS) ist ein System, welches die Änderungen an einer oder einer Reihe von Dateien über die Zeit hinweg protokolliert.

→ Mit dem System kann man später auf eine bestimmte Version zurückgreifen kann

VERSIONIERUNG

Lokales VCS

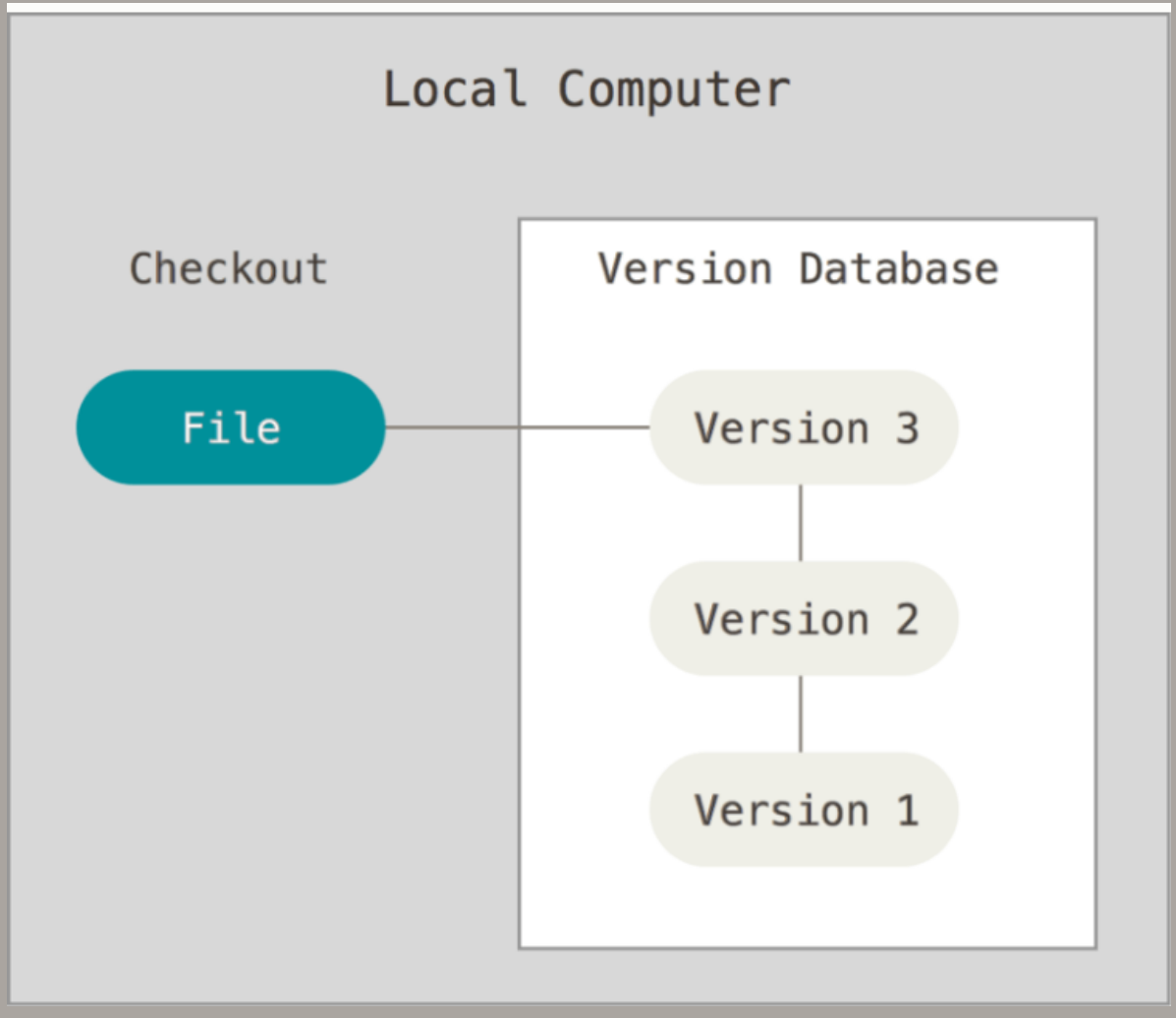
→ Eines der populäreren Versionsverwaltungssysteme war RCS

→ RCS arbeitet nach dem Prinzip, dass für jede Änderung ein Patch (ein Patch umfasst alle Änderungen an einer oder mehreren Dateien) in einem speziellen Format auf der Festplatte gespeichert wird

→ Um eine bestimmte Version einer Datei wiederherzustellen, wendet es alle Patches bis zur gewünschten Version an und rekonstruiert damit die Datei in der gewünschten Version.

VERSIONIERUNG

Lokales VCS



VERSIONIERUNG

Zentralisierte Versionskontrolle (CVCS)

→ Beispielsysteme sind CVS, Subversion und Perforce

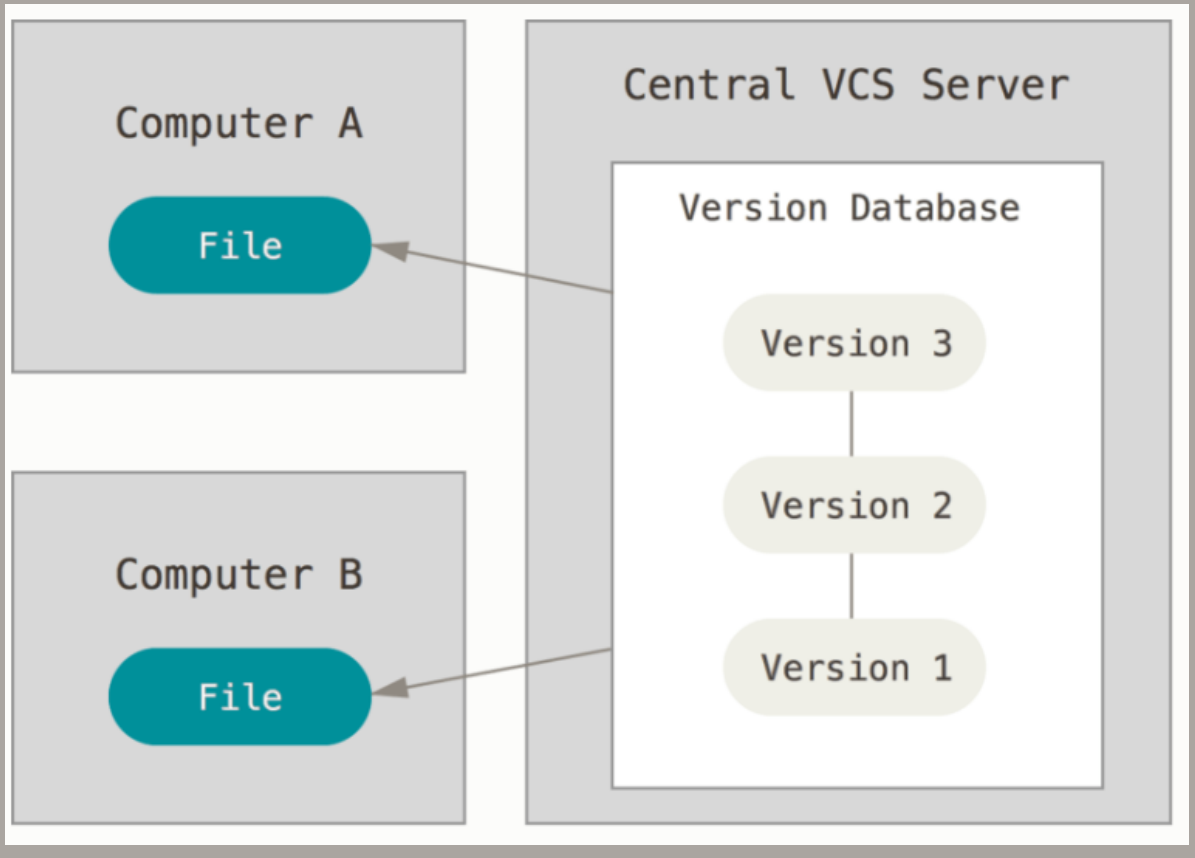
→ basieren auf einem zentralen Server, der alle versionierten Dateien verwaltet

→ Die Clients können die Dateien von diesem zentralen Ort abholen und auf ihren PC übertragen.

→ Den Vorgang des Abholens nennt man Auschecken (engl. to check out).

VERSIONIERUNG

Zentralisiertes VCS



VERSIONIERUNG

Verteilte Versionsverwaltungsysteme (DVCS) - 1

→ Beispielsysteme sind Git, Mercurial, Bazaar oder Darcs

→ Anwender nicht einfach nur den jeweils letzten Zustand des Projektes von einem Server: Sie erhalten stattdessen eine vollständige Kopie des Repositorys.

→ Jede Kopie, ein sogenannter Klon (engl. clone), ist ein vollständiges Backup der gesamten Projektdaten.

VERSIONIERUNG

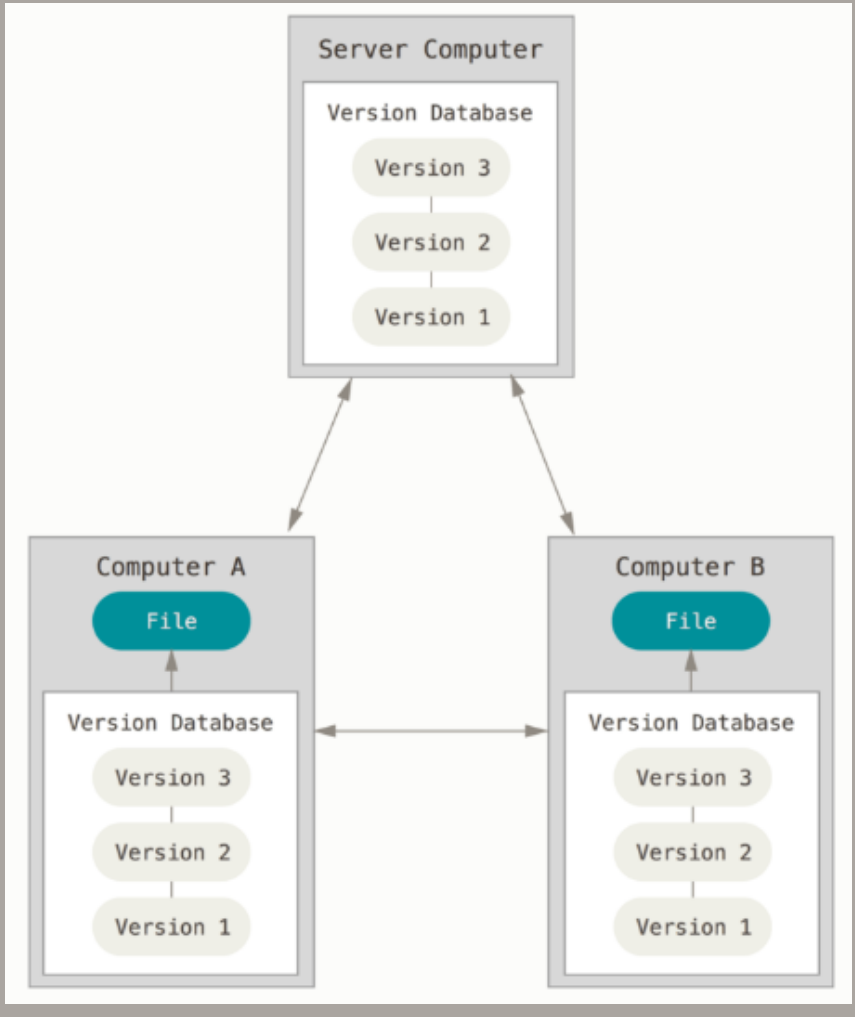
Verteilte Versionsverwaltungsysteme (DVCS) - 2

→ Darüber hinaus können derartige Systeme hervorragend mit verschiedenen externen Repositorys, sogenannten Remote-Repositorys, umgehen, sodass man mit verschiedenen Gruppen von Leuten simultan auf verschiedene Art und Weise, an einem Projekt zusammenarbeiten kann.

→ Damit ist es möglich, verschiedene Arten von Arbeitsabläufen zu erstellen und anzuwenden, welche mit zentralisierten Systemen nicht möglich wären. Dazu gehören zum Beispiel hierarchische Arbeitsabläufe.

VERSIONIERUNG

Verteilte VCS



VERSIONIERUNG

GIT

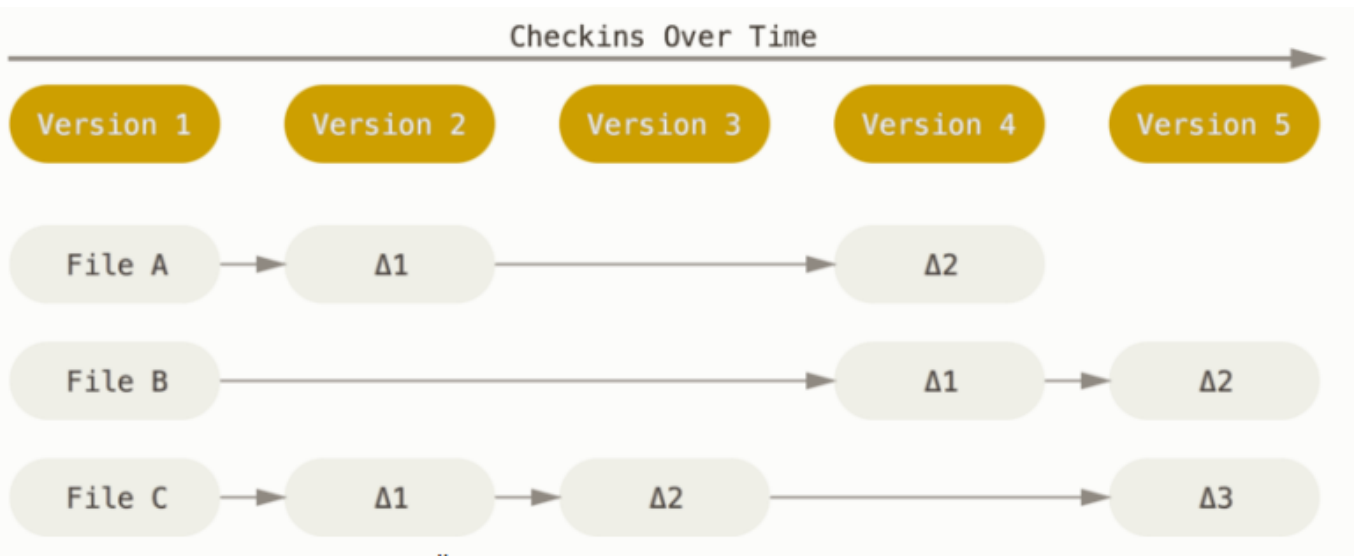
- → Geburt 2005
- → Entstanden aus der Linux Entwickler Community
- → Ziele:
  - → Geschwindigkeit
  - → Einfaches Design
  - → Gute Unterstützung von nicht-linearer Entwicklung (tausende parallele Entwicklungszweige)
  - → Vollständig dezentrale Struktur
  - → Fähigkeit große Projekte, wie den Linux Kernel, effektiv zu verwalten (Geschwindigkeit und Datenumfang)

GIT GRUNDLAGEN

Snapshots und nicht die Unterschiede

→ Die meisten anderen Systeme speichern Information, als eine fortlaufende Liste von Änderungen an Dateien.

→ Diese Systeme betrachten die Informationen, die sie verwalten, als eine Menge von Dateien und die Änderungen, die über die Zeit hinweg an einzelnen Dateien vorgenommen werden.



GIT GRUNDLAGEN

Snapshots und nicht die Unterschiede

→ Stattdessen betrachtet Git seine Daten eher als eine Reihe von Snapshots eines Mini-Dateisystems.

→ Der gegenwärtigen Status Ihres Projekts wird als eine Version in Git gespeichert

→ Sichert Git den Zustand sämtlicher Dateien in diesem Moment und macht sozusagen ein Schnappschuss (engl. Snapshot) von all Ihren Daten.

→ Zusätzlich speichert Git eine Referenz auf diesen Snapshot.

→ Um dies möglichst effizient und schnell tun zu können, kopiert Git unveränderte Dateien nicht, sondern legt lediglich eine Verknüpfung zu der vorherigen Version der Datei an.

GIT GRUNDLAGEN

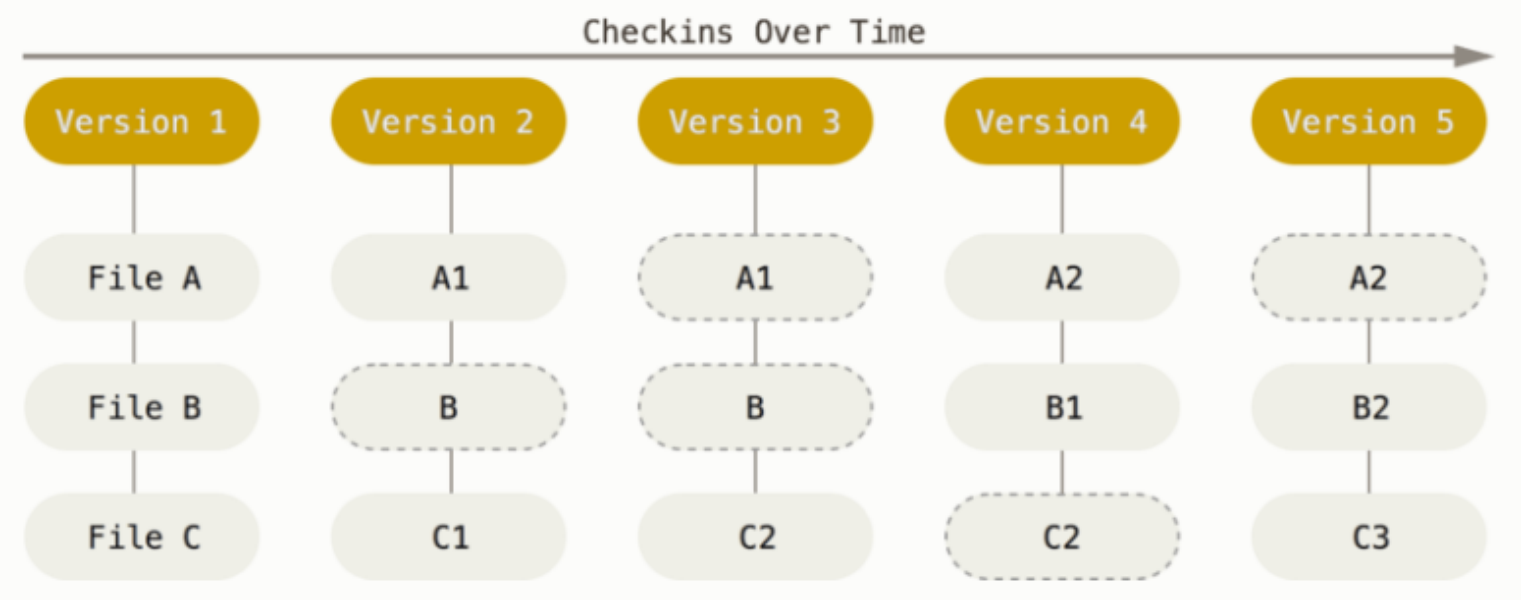
Zustände

→ Committet (engl. committed) - dass die Daten sicher in der lokalen Datenbank gespeichert sind

→ Geändert (engl. modified) - dass eine Datei geändert, aber noch nicht in die lokale Datenbank eingecheckt wurde

→ für Commit vorgemerkt (engl. staged) - dass eine geänderte Datei in ihrem gegenwärtigen Zustand für den nächsten Commit vorgemerkt ist.

GIT GRUNDLAGEN



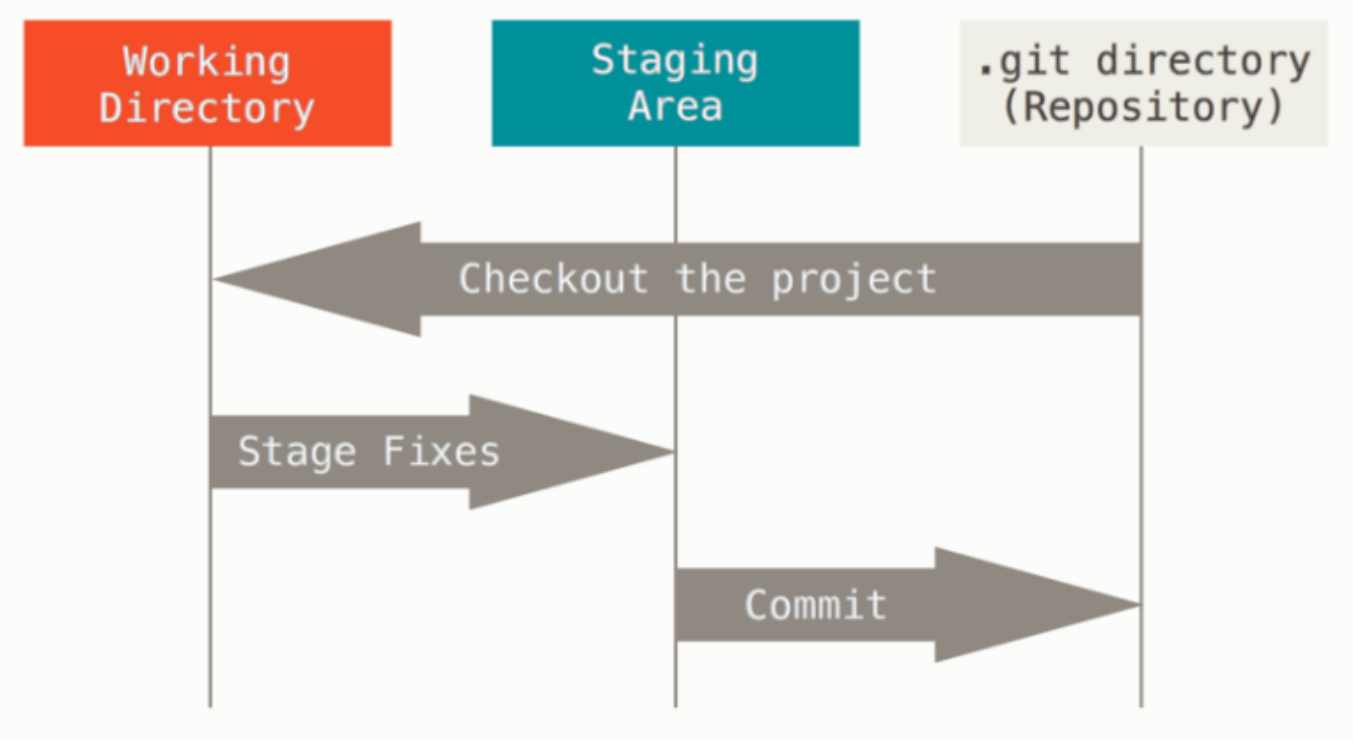
GIT GRUNDLAGEN

Hauptbereiche

→ Working Directory - Das Arbeitsverzeichnis

→ Staging Area

→ Repository - Das Git Verzeichnis



GIT GRUNDLAGEN

Hauptbereiche

→ Working Directory - Das Arbeitsverzeichnis ist ein einzelnes Abbild einer spezifischen Version des Projektes. Die dort enthaltenen Dateien werden aus der komprimierten Datenbank geholt und auf der Festplatte in einer Form gespeichert, sodass man sie nutzen oder bearbeiten kann.

GIT GRUNDLAGEN

Hauptbereiche

→ Staging Area - eine Datei, normalerweise befindet sich diese im Git Verzeichnis, in der vorgemerkt wird, welche Änderungen der nächste Commit umfassen soll. Manchmal wird dieser Ort auch als “Index” bezeichnet, aber der Begriff Staging-Area ist der gängigere.

GIT GRUNDLAGEN

Hauptbereiche

→ Repository - Das Git Verzeichnis ist der Ort, an dem Git Metadaten und die lokale Datenbank für ein Projekt sichert. Dies ist der wichtigste Teil von Git, und dieser Teil wird kopiert, wenn man ein Repository von einem anderen Rechner klont.

GIT GRUNDLAGEN

Arbeitsschritte beim Einchecken

1. Man ändert die zu bearbeitenden Dateien im Arbeitsverzeichnis
2. Man merkt die Dateien für einen Commit vor, fügt also einen Schnappschuss der Dateien der Staging-Area hinzu
3. Man führt einen Commit aus, wodurch der in der Staging-Area vorgemerkte Schnappschuss dauerhaft im Git Verzeichnis gespeichert wird

GIT GRUNDLAGEN

Installation unter Linux

```
sudo yum install git-all
```

bzw.

```
sudo apt-get install git-all
```

GIT GRUNDLAGEN

Installation unter Windows

Eine offizielle Windows Version findet man direkt auf der Git Homepage.

GIT BASIS-KONFIGURATION

git config

Git umfasst das Werkzeug git config, welches die Möglichkeit bietet, Konfigurationswerte zu verändern. Die Konfiguration ist an drei verschiedenen Orten gespeichert:

→ /etc/gitconfig enthält Parameter, die für jeden Anwender des Systems und alle Projekte gelten. Wenn man git config mit der Option –system verwendet, wird von dieser Datei gelesen bzw. in diese Datei geschrieben.

→ /.gitconfig gelten ausschließlich für den jeweiligen Benutzer. Wenn man git config mit der Option –global verwendet, wird von dieser Datei gelesen bzw. in diese Datei geschrieben.

→ config im Git Verzeichnis (also ‘.git/config) enthält Parameter, die für das jeweilige Projekt gelten.

GIT BASIS-KONFIGURATION

git config

Diese drei Dateien haben unterschiedliche Prioritäten. Die oberste Priorität haben die Werte aus .git/config, dann folgt /.gitconfig und zuletzt /etc/gitconfig. Ist zum Beispiel ein Parameter in .git/config und /etc/gitconfig mit unterschiedlichen Werten gesetzt, so gilt in diesem Fall der höherpriore Wert aus der Datei .git/config.

GIT BASIS-KONFIGURATION

Persönliche Konfiguration

→ Name angeben:
`git config --global user.name "`Eva-Maria Iwer"'

→ eMail angeben: `git config --global user.email`„eva-maria.iwer@hs-rm.de“

→ Konfig anzeigen: `git config --list`

ARBEITEN MIT GIT

Ein existierendes Verzeichnis als Git Repository initialisieren

Wenn Sie ein bestehendes Projekt in Zukunft versionieren möchten, können Sie dazu in das Hauptverzeichnis des Projekts wechseln und den folgenden Befehl ausführen:

```
git init
```

Achtung

Zu diesem Zeitpunkt werden noch keine Dateien in Git versioniert.

ARBEITEN MIT GIT

initalen Commit

Mit dem Befehl `git add` legen Sie fest, welche Dateien versioniert werden sollen und mit dem Befehl `git commit` erzeugen Sie einen neuen Commit:

```
→  git add *.c
→  git add LICENSE
→  git commit -m 'initial project version'
```

ARBEITEN MIT GIT

Ein existierendes Repository klonen

→ Wenn Sie eine Kopie eines existierenden Git Repositorys anlegen wollen, können Sie den Befehl
`git clone` verwenden.

→ jede einzelne Version jeder einzelnen Datei, also die gesamte Historie eines Projekts auf den Rechner heruntergeladen.

ARBEITEN MIT GIT

Änderungen nachverfolgen und im Repository speichern

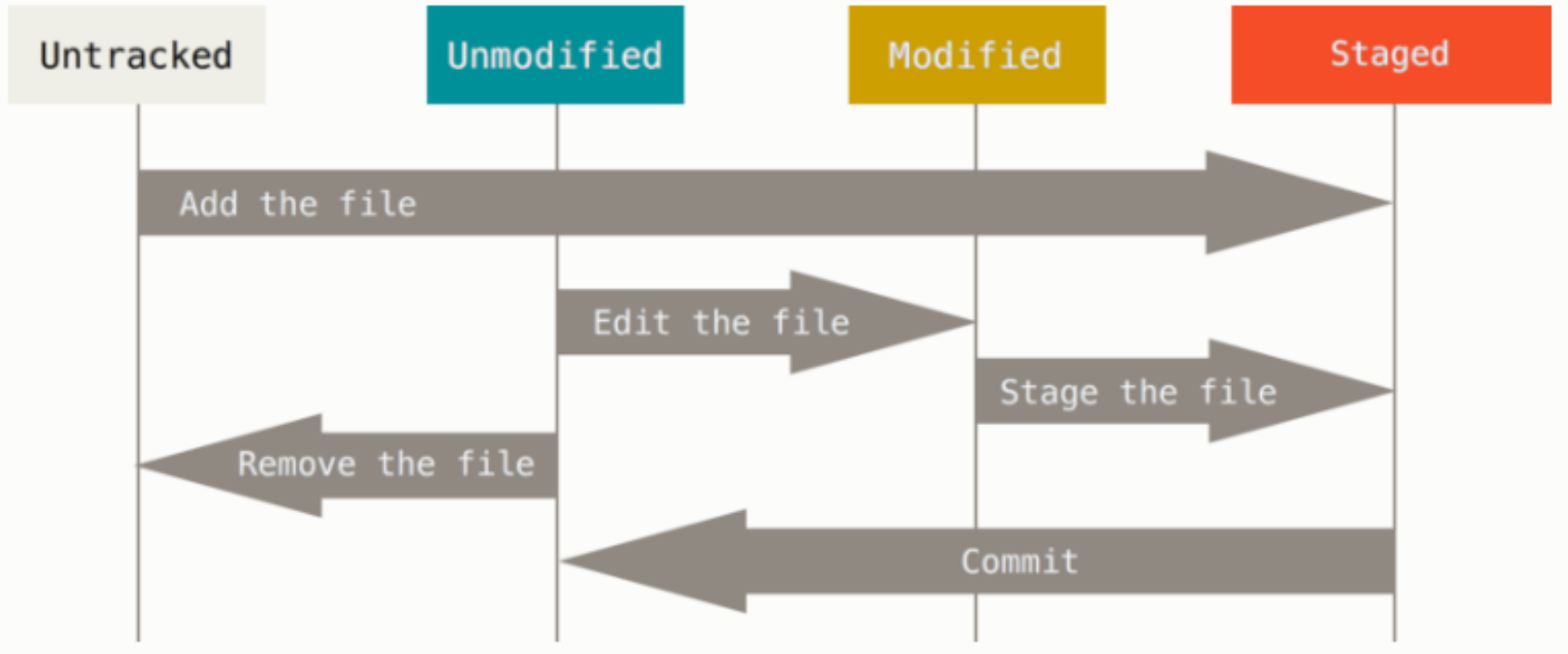
Ab jetzt können Dateien im Projekt bearbeitet und eincheckt werden.

→ Wenn Sie eine Kopie eines existierenden Git Repositorys anlegen wollen, können Sie den Befehl `git clone` verwenden.

→ jede einzelne Version jeder einzelnen Datei, also die gesamte Historie eines Projekts auf den Rechner heruntergeladen.

ARBEITEN MIT GIT

Änderungen nachverfolgen und im Repository speichern



ARBEITEN MIT GIT

Änderungen nachverfolgen und im Repository speichern

→ Sobald Sie anfangen, versionierte Dateien zu bearbeiten, erkennt Git diese als modifiziert, weil sie sich im Vergleich zum letzten Commit verändert haben.

→ Die geänderten Dateien können Sie dann für den nächsten Commit vormerken und schließlich alle Änderungen, die sich in der Staging-Area befinden, einchecken.

→ Danach geht der Vorgang wieder von vorne los.

ARBEITEN MIT GIT

Änderungen nachverfolgen und im Repository speichern

→ Das wichtigste Hilfsmittel, um den Zustand zu überprüfen, in dem sich Ihre Dateien gerade befinden, ist der Befehl `git status`.

→ Wenn Sie diesen Befehl unmittelbar nach dem Klonen eines Repositorys ausführen, sollte er in etwa folgende Ausgabe liefern:

```
git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
```

ARBEITEN MIT GIT

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

        Vorlesung/SW_Vorlesung/test

no changes added to commit (use "git add" and/or "git commit -a")
```

Neue Dateien zur Versionsverwaltung hinzufügen

```
→ git add test
→ git status
```

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   Vorlesung/SW_Vorlesung/test
```

ARBEITEN MIT GIT

Neue Dateien zur Versionsverwaltung hinzufügen

→ Dass die Datei für den nächsten Commit vorgemerkt ist, sehen Sie daran, dass sie im Abschnitt „Changes to be committed“ aufgelistet ist.

→ Wenn Sie jetzt einen Commit anlegen, wird der Schnappschuss den Zustand der Datei beinhalten, den sie zum Zeitpunkt des Befehls `git add` hatte.

→ Der `git add` Befehl akzeptiert einen Pfadnamen einer Datei oder eines Verzeichnisses. Wenn Sie ein Verzeichnis angeben, fügt `git add` alle Dateien in diesem Verzeichnis und allen Unterverzeichnissen rekursiv hinzu.

ARBEITEN MIT GIT

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   Vorlesung/SW_Vorlesung/SW_Vorlesung.pdf
        modified:   Vorlesung/SW_Vorlesung/SW_Vorlesung.tex
        modified:   Vorlesung/SW_Vorlesung/git.tex
```

Geänderte Dateien zur Staging-Area hinzufügen

→ „Changed but not staged for commit“ – das bedeutet, dass eine versionierte Datei im Arbeitsverzeichnis verändert worden ist, aber noch nicht für den Commit vorgemerkt wurde.

→ Um sie vorzumerken, führen Sie den Befehl `git add` aus.

→ Befehl git add - bestimmten Inhalt für den nächsten Commit vorbereiten.

ARBEITEN MIT GIT

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   Vorlesung/SW_Vorlesung/SW_Vorlesung.pdf
        modified:   Vorlesung/SW_Vorlesung/SW_Vorlesung.tex
        modified:   Vorlesung/SW_Vorlesung/git.tex
        new file:   Vorlesung/SW_Vorlesung/images/gitadd.PNG
        new file:   Vorlesung/SW_Vorlesung/images/gitadd1.PNG
        new file:   Vorlesung/SW_Vorlesung/images/gitadd2.PNG
        new file:   Vorlesung/SW_Vorlesung/test
```

Geänderte Dateien zur Staging-Area hinzufügen

→ alle Dateien im Abschnitt „Changes to be committed“ sind für den nächsten Commit vorgemerkt

→ Wenn Sie jetzt einen Commit anlegen, wird der Schnappschuss den Zustand der Datei beinhalten, den sie zum Zeitpunkt des Befehls git add hatte.

ARBEITEN MIT GIT

Dateien commit

Dateien in das Verzeichnis hochladen

`git commit`

```
[master 233ba0a] neue Version für gitadd mit beispiel test hinzufügen
7 files changed, 100 insertions(+), 15 deletions(-)
create mode 100644 Vorlesung/SW_Vorlesung/images/gitadd.PNG
create mode 100644 Vorlesung/SW_Vorlesung/images/gitadd1.PNG
create mode 100644 Vorlesung/SW_Vorlesung/images/gitadd2.PNG
create mode 100644 Vorlesung/SW_Vorlesung/test
```

ARBEITEN MIT GIT

Dateien löschen

Als erstes müssen Sie die Datei von ihren tracked Dateien entfernen und dann commiten. Das `git rm` Kommando untersützt. Wenn nur die Datei entfernt wird, wird „Changed but not updated“ in ihrem Status.

```
git rm test
git status
```

ARBEITEN MIT GIT

Dateien bewegen

```
git mv file_from file_to
```

ARBEITEN MIT GIT

Dateien ignorieren

Erstellen einer .gitignore Datei

```
# no tex created files
*.log
*.tcp
*.toc
*.aux
*.nav
*.out
*.snm
*.bbl
*.blg
# ignore all files with grading relevant data
Klausur/
Praktikum/solution
```

ARBEITEN MIT GIT

Commit History

`git log`

```
commit 8f09fd3218d88fd9fa6e2109bc4d1b7fa80ff5e4 (HEAD -> master)
Author: Eva-Maria Iwer <eva-maria.iwer@hs-rm.de>
Date:   Thu Jul 18 09:35:46 2019 +0200

    Beispieldatei wieder l<C3><B6>schen

commit 233ba0aae4ccbc56d7f1dd524b87c78544d9d36e
Author: Eva-Maria Iwer <eva-maria.iwer@hs-rm.de>
Date:   Thu Jul 18 09:32:20 2019 +0200

    neue Version f<C3><BC>r gitadd mit beispiel test hinzuf<C3><BC>gen

commit 9f60c5de8a2aa8051d770df5cf69e5d7b2edcfae
Author: Eva-Maria Iwer <eva-maria.iwer@hs-rm.de>
Date:   Wed Jul 17 15:12:31 2019 +0200

    zeusch
```

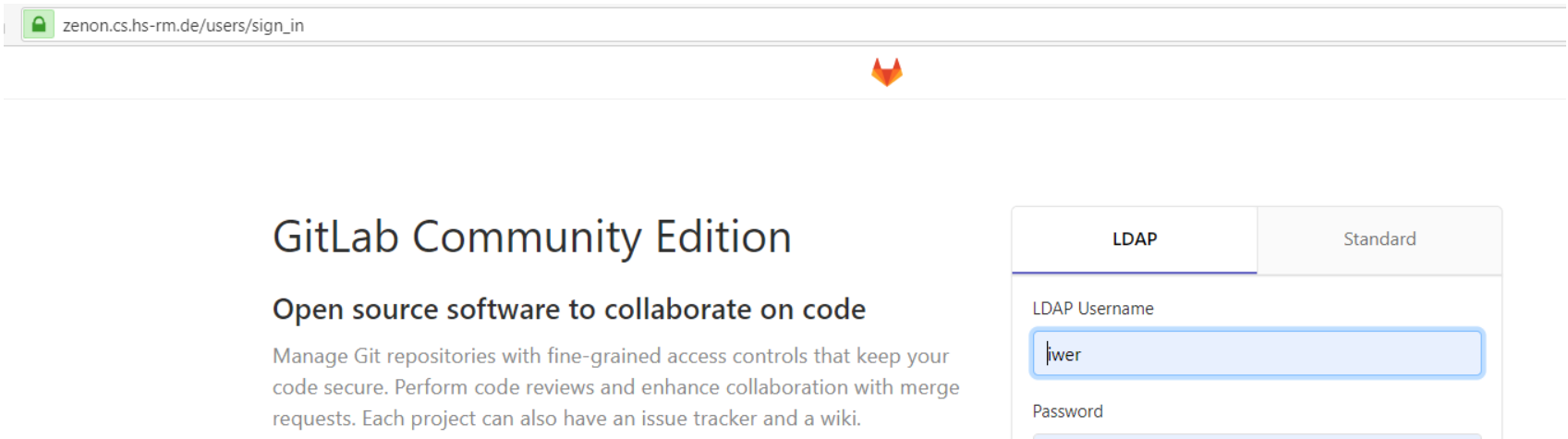
VERTEILTES ARBEITEN MIT GIT

Git-Server der HS-RM

→ Im LDAP-Server muss eine Mailadresse hinterlegt sein, was normalerweise nicht der Fall ist.

→ Um die Adresse zu setzen, bitte unter https://ldap.local.cs.hs-rm.de/mail/index.php einloggen und dort die Hochschul email-Adresse angeben.

→ Unter https://zenon.cs.hs-rm.de kann jetzt das Gitlab Account angelegt werden.

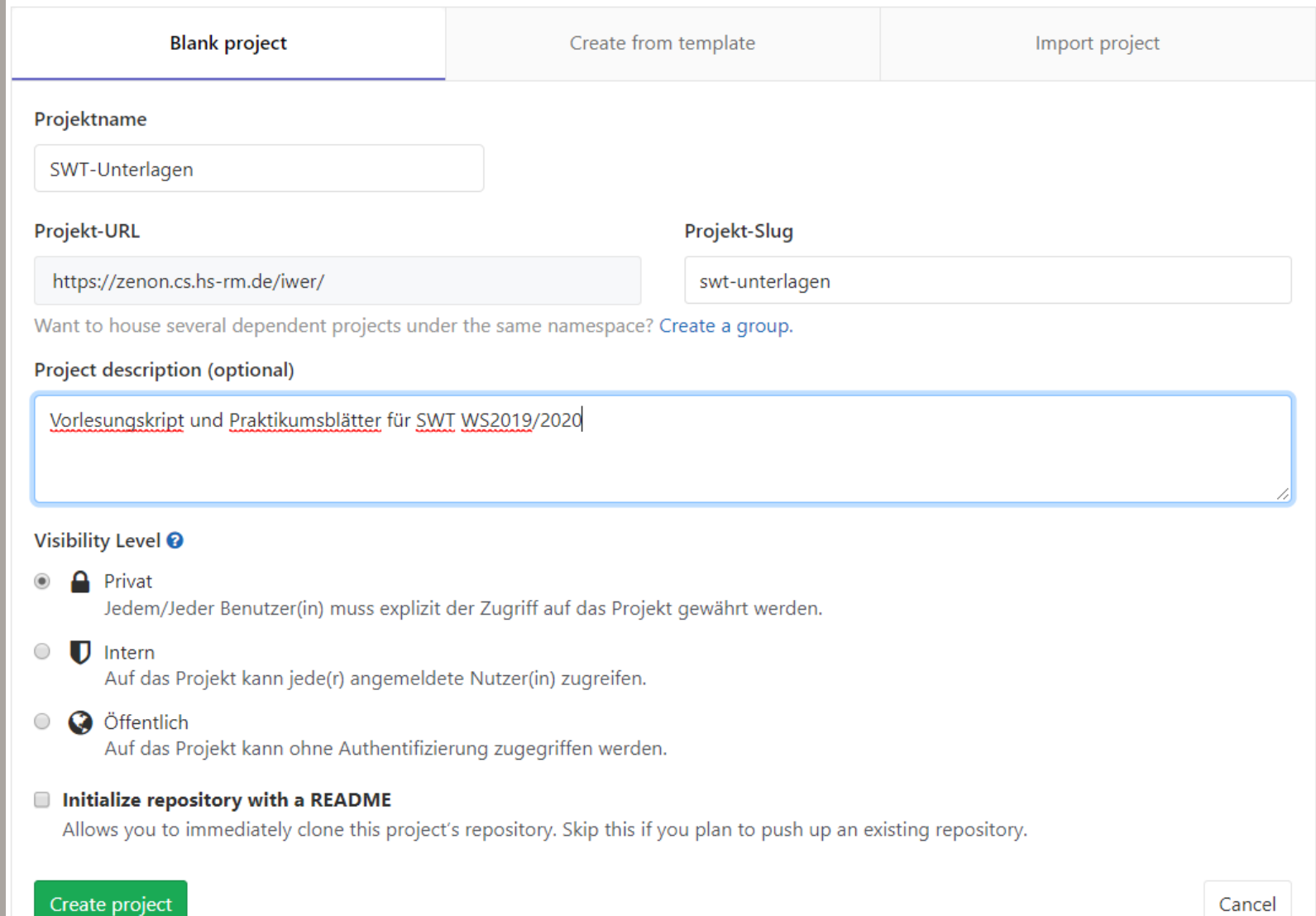


VERTEILTES ARBEITEN MIT GIT

Git-Server der HS-RM - neues Projekt hinzufügen

VERTEILTES ARBEITEN MIT GIT

Git-Server der HS-RM - neues Projekt hinzufügen



VERTEILTES ARBEITEN MIT GIT

Übersicht Befehle

Git global setup

```
git config --global user.name "Eva Iwer"
git config --global user.email "eva-maria.iwer@hs-rm.de"
```

Create a new repository

```
git clone git@gitlab.cs.hs-rm.de:iwer/swt-unterlagen.git
cd swt-unterlagen
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Push an existing folder

```
cd existing_folder
git init
git remote add origin git@gitlab.cs.hs-rm.de:iwer/swt-unterlagen.git
git add .
git commit -m "Initial commit"
git push -u origin master
```

Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin git@gitlab.cs.hs-rm.de:iwer/swt-unterlagen.git
git push -u origin --all
git push -u origin --tags
```

BRANCHES

Einleitung

→ Nahezu jedes VCS unterstützt eine Form von Branching

→ Branching bedeutet, dass Sie von der Hauptlinie der Entwicklung abzweigen und Ihre Arbeit fortsetzen

→ GIT ist unglaublich leichtgewichtig, wodurch Branch-Operationen nahezu verzögerungsfrei ausgeführt werden

→ leichtes Hin- und Herschalten zwischen einzelnen Entwicklungszweigen

BRANCHES

Branches auf einen Blick

→ Wenn Sie einen Commit durchführen, speichert Git ein Commit-Objekt, das einen Zeiger auf den Snapshot des von Ihnen bereitgestellten Inhalts enthält

→ zeigt auf den Commit oder die Commits, die direkt vor diesem Commit stattfanden (zu seinem Vorgänger bzw. seinen Vorgängern)

→ Besonderheiten: keine Vorgänger für den ersten Commit, einen Vorgänger für einen normalen Commit und mehrere Vorgänger für einen Commit, welcher aus dem Zusammenführen (engl. mergen) von zwei oder mehr Branches resultiert.

BRANCHES

Branches auf einen Blick

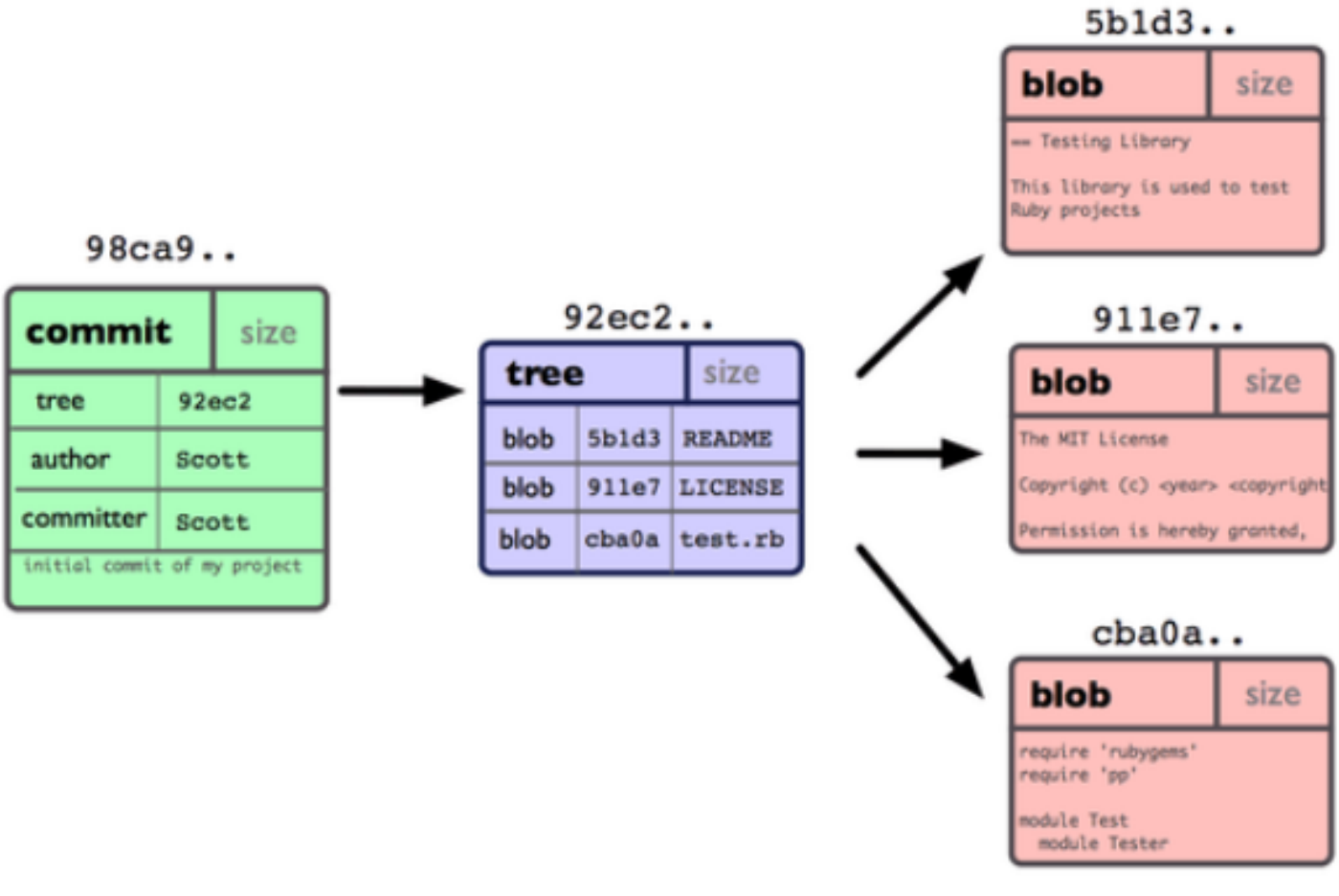
→ Bei Commit wird Prüfsumme erstellt

→ Speichert diese Version der Datei im Git-Repository (Git verweist auf diese als blobs) und fügt die Prüfsumme der Staging-Area hinzu

→ Ihr Git-Repository enthält jetzt verschiedene Objekte: ihre blobs (die jeweils den Inhalt einer der drei Dateien repräsentieren), ein tree-Objekt, welches den Inhalt des Verzeichnisses auflistet und angibt, welcher Dateiname zu welchem Blob gehört, und ein commit-Objekt mit dem Zeiger, der auf die Root des Projektbaumes und die Metadaten des Commits verweist.

BRANCHES

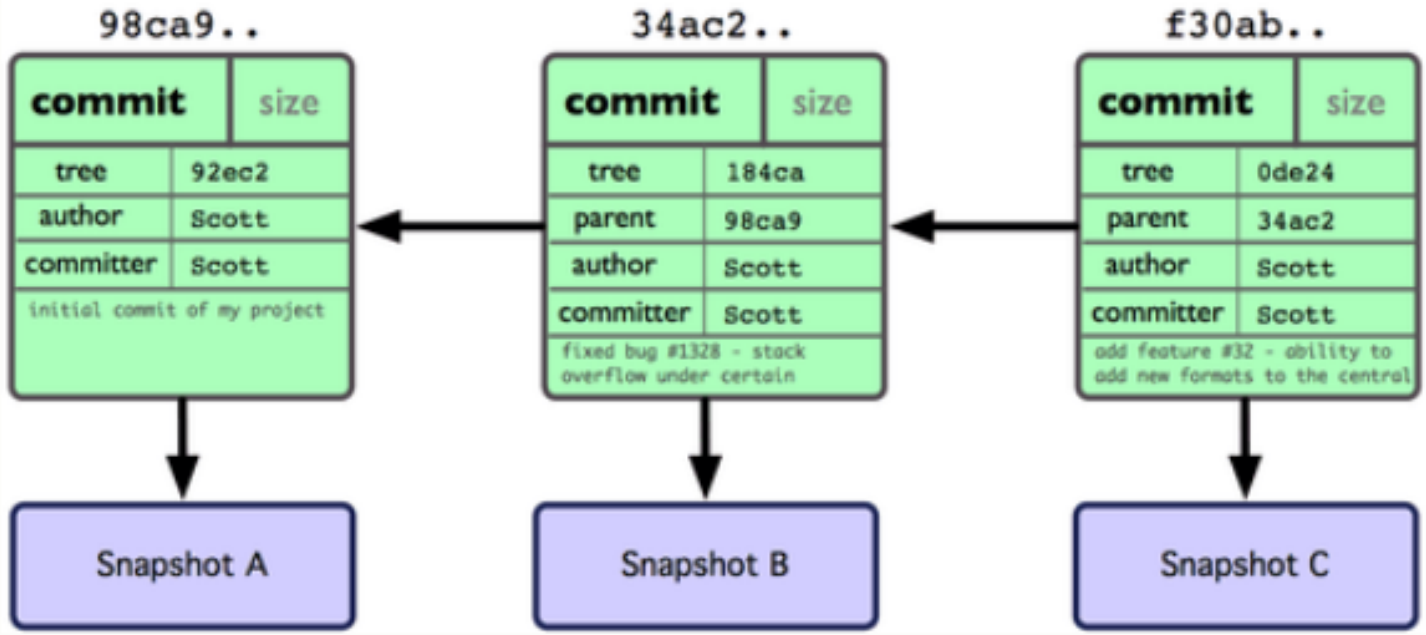
Commit und sein Tree



BRANCHES

Commits und ihre Vorgänger

Wenn Sie einige Änderungen vornehmen und wieder einen Commit durchführen, speichert dieser einen Zeiger zu dem Commit, der unmittelbar davor gemacht wurde.



BRANCHES

Master

→ Ein Branch in Git ist einfach ein leichter, beweglicher Zeiger auf einen dieser Commits.

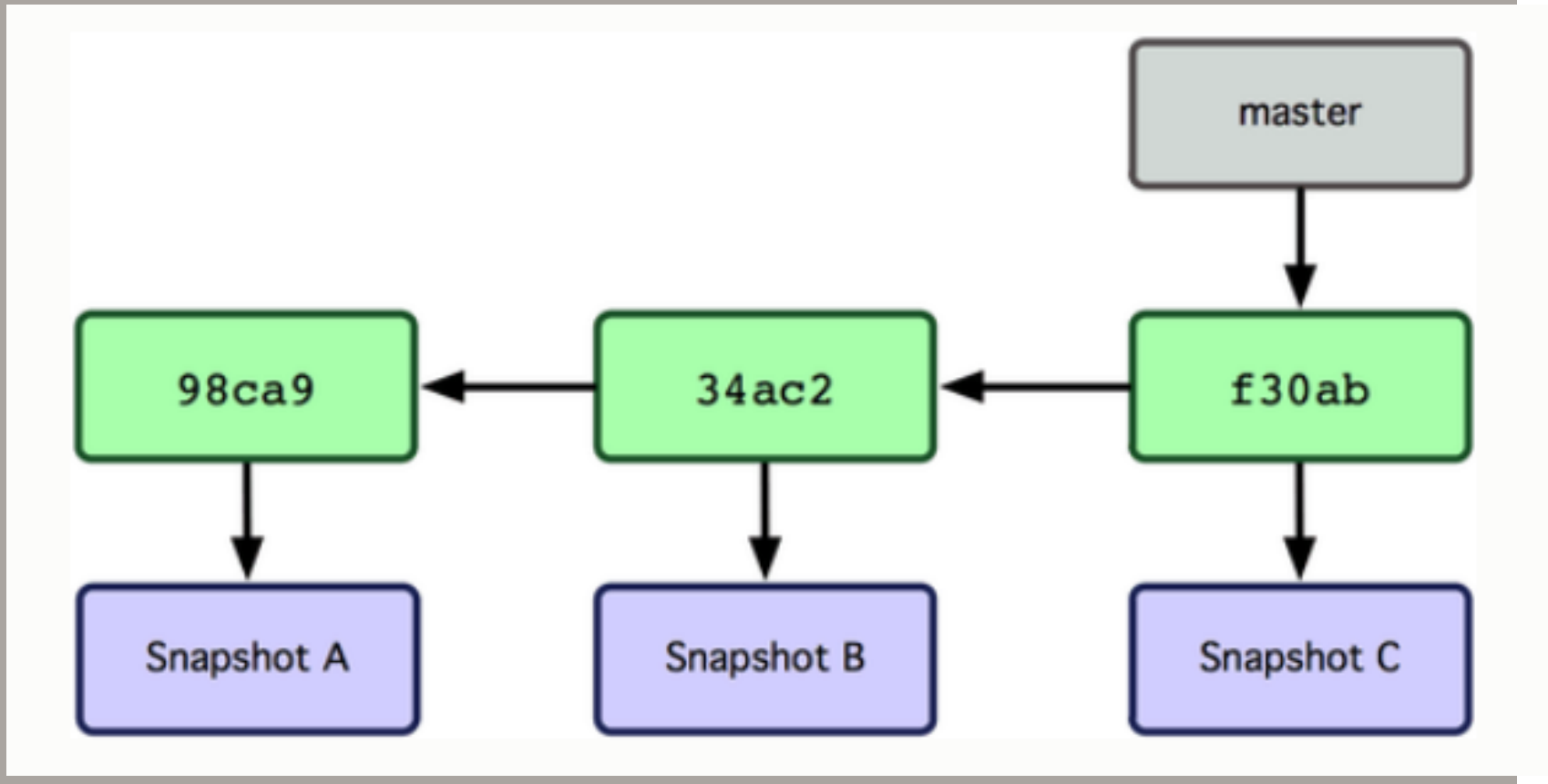
→ Die Standardbezeichnung für einen Branch bei Git lautet master.

→ Wenn Sie damit beginnen, Commits durchzuführen, erhalten Sie einen master-Branch, der auf den letzten Commit zeigt, den Sie gemacht haben.

→ Jedes Mal, wenn Sie einen Commit durchführen, bewegt er sich automatisch vorwärts.

BRANCHES

Ein Branch und sein Commit-Verlauf



BRANCHES

Erzeugen eines neuen Branches

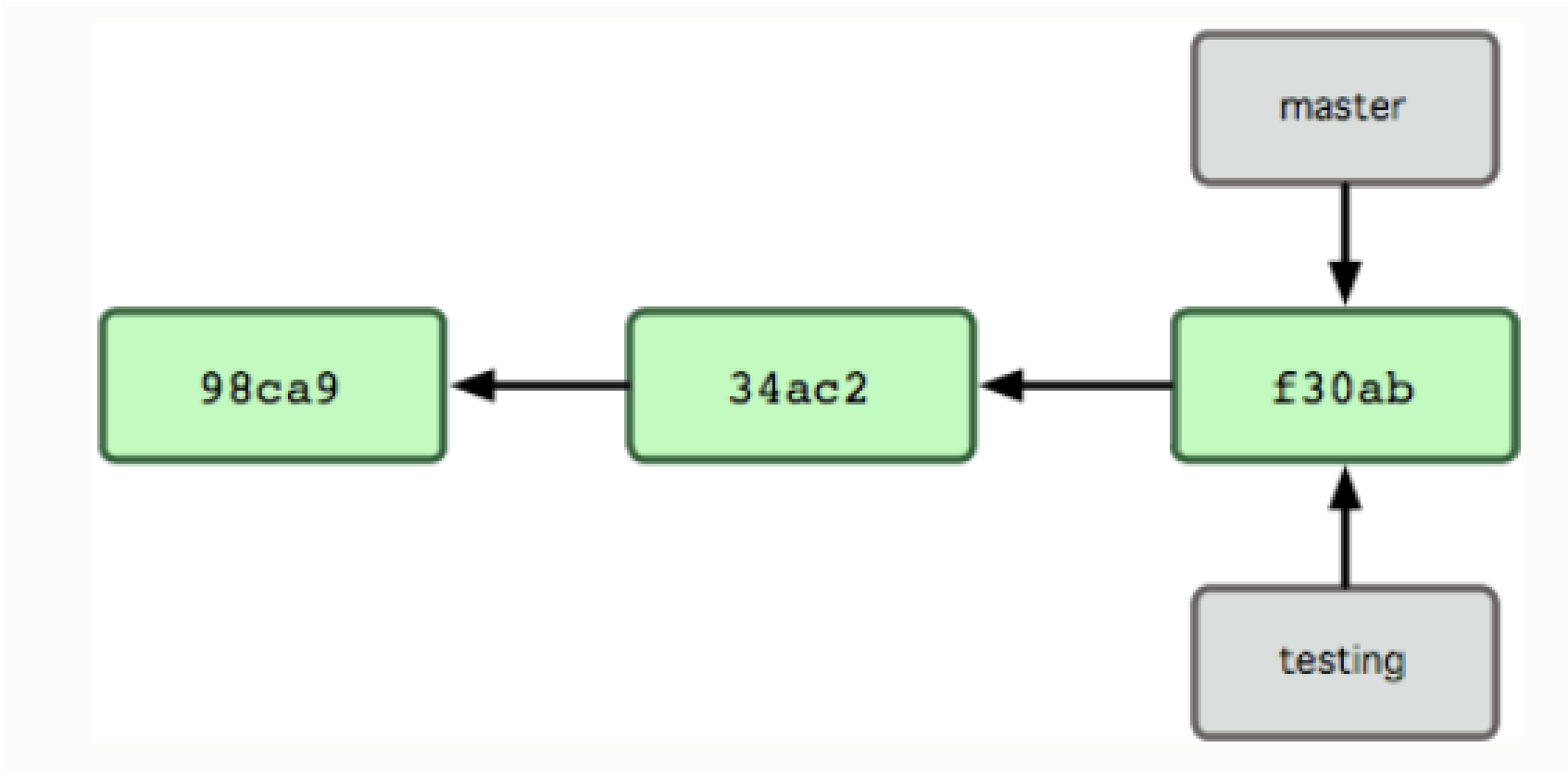
Was passiert, wenn Sie einen neuen Branch erzeugen?
Ein neuer Zeiger erstellt, mit dem Sie sich in der Entwicklung fortbewegen können
Bsp Brunch testing:

```
git branch testing
```

Dieser Befehl erzeugt einen neuen Zeiger, der auf den selben Commit zeigt, auf dem Sie sich gegenwärtig befinden.

BRANCHES

Zwei Branches, die auf die selbe Serie von Commits zeigen



BRANCHES

Woher weiß Git, auf welchem Branch Sie gegenwärtig sind?

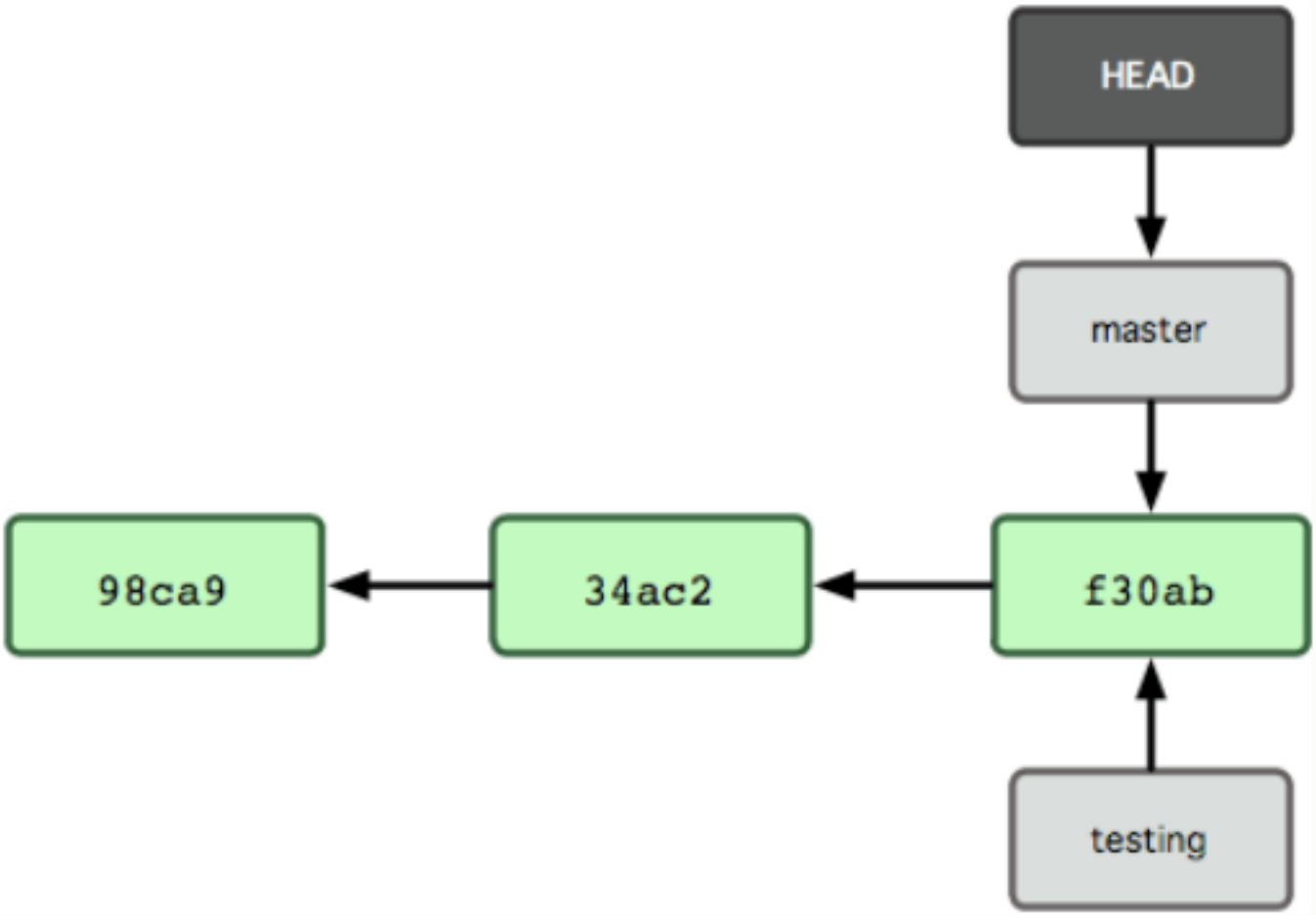
→ speziellen Zeiger namens HEAD

→ Zeiger auf den lokalen Branch, auf dem Sie sich gegenwärtig befinden

→ Anweisung git branch hat den neuen Branch nur erzeugt, aber nicht zu diesem gewechselt.

BRANCHES

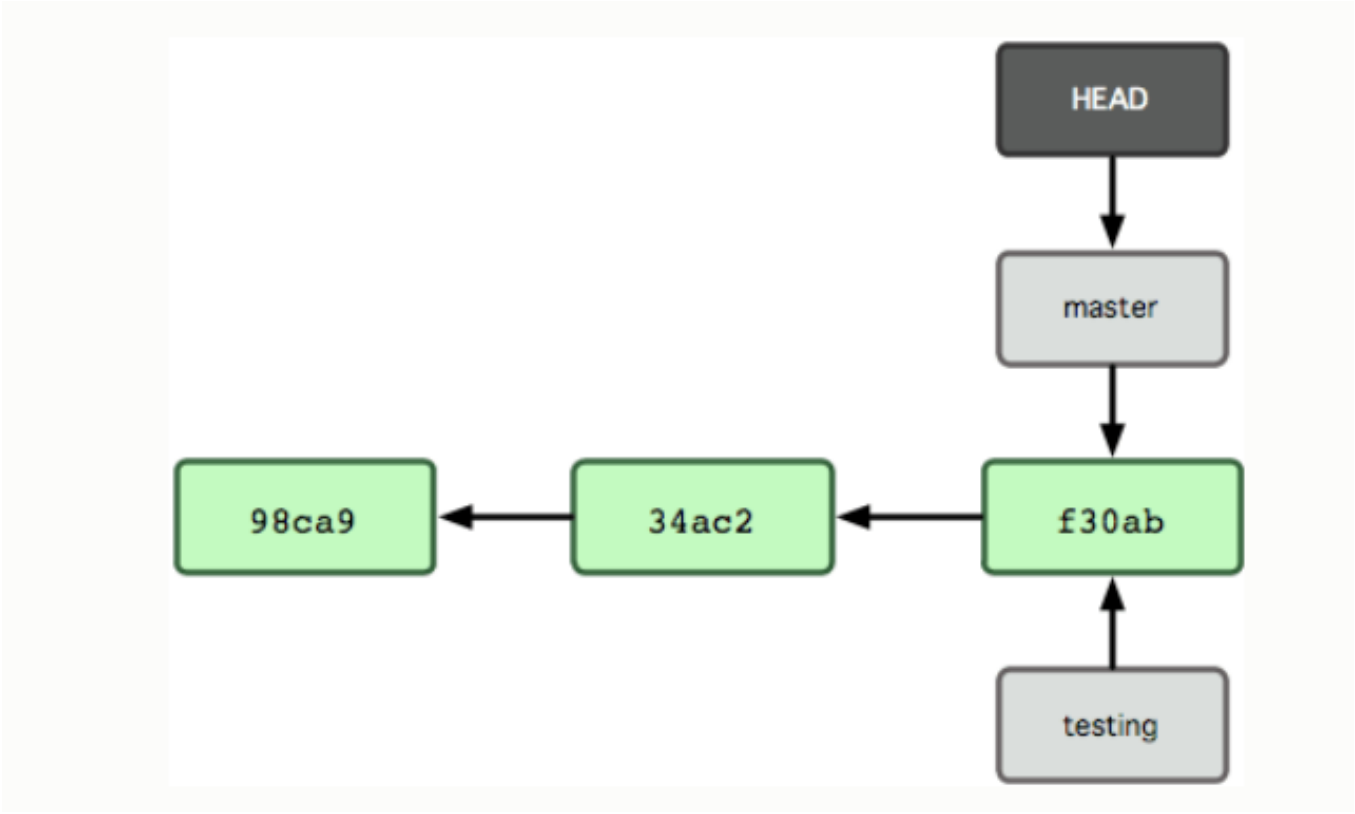
Auf einen Branch zeigender HEAD



BRANCHES

Wechseln der Branches

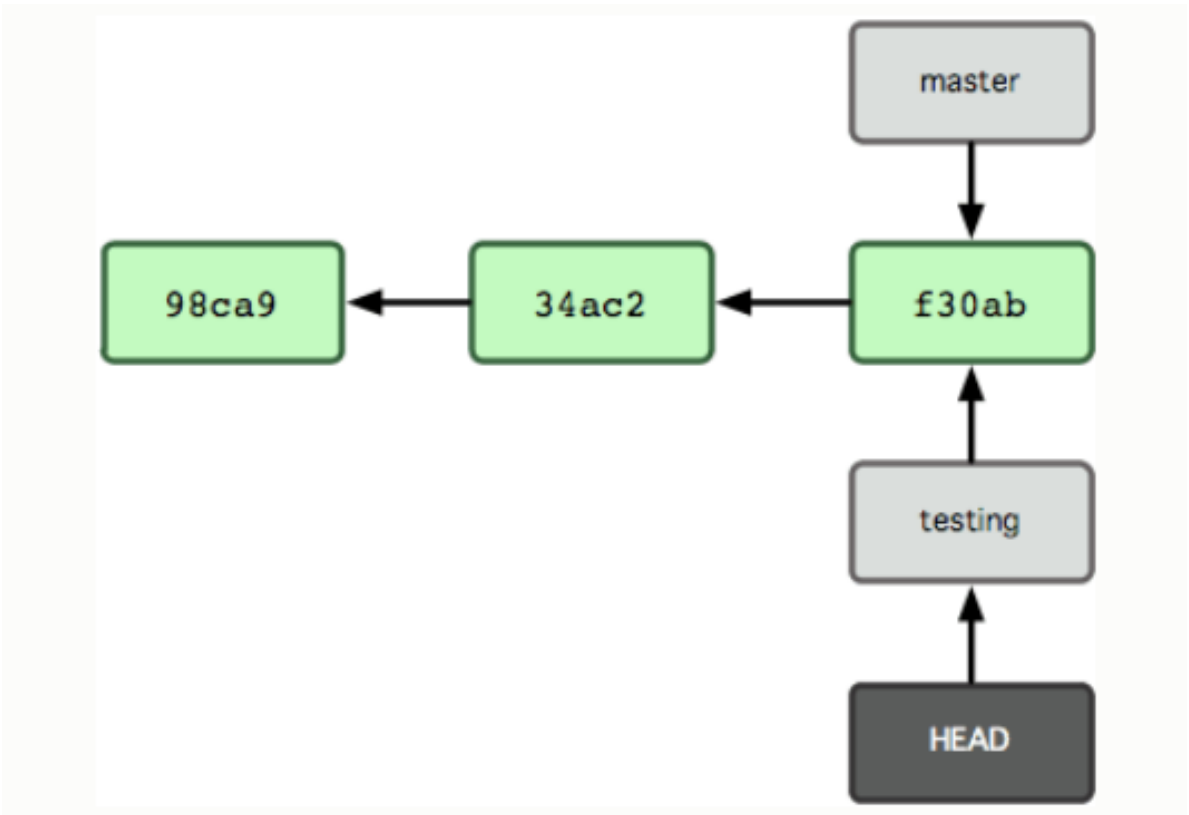
```
git checkout testing
```



BRANCHES

HEAD zeigt auf den aktuellen Branch

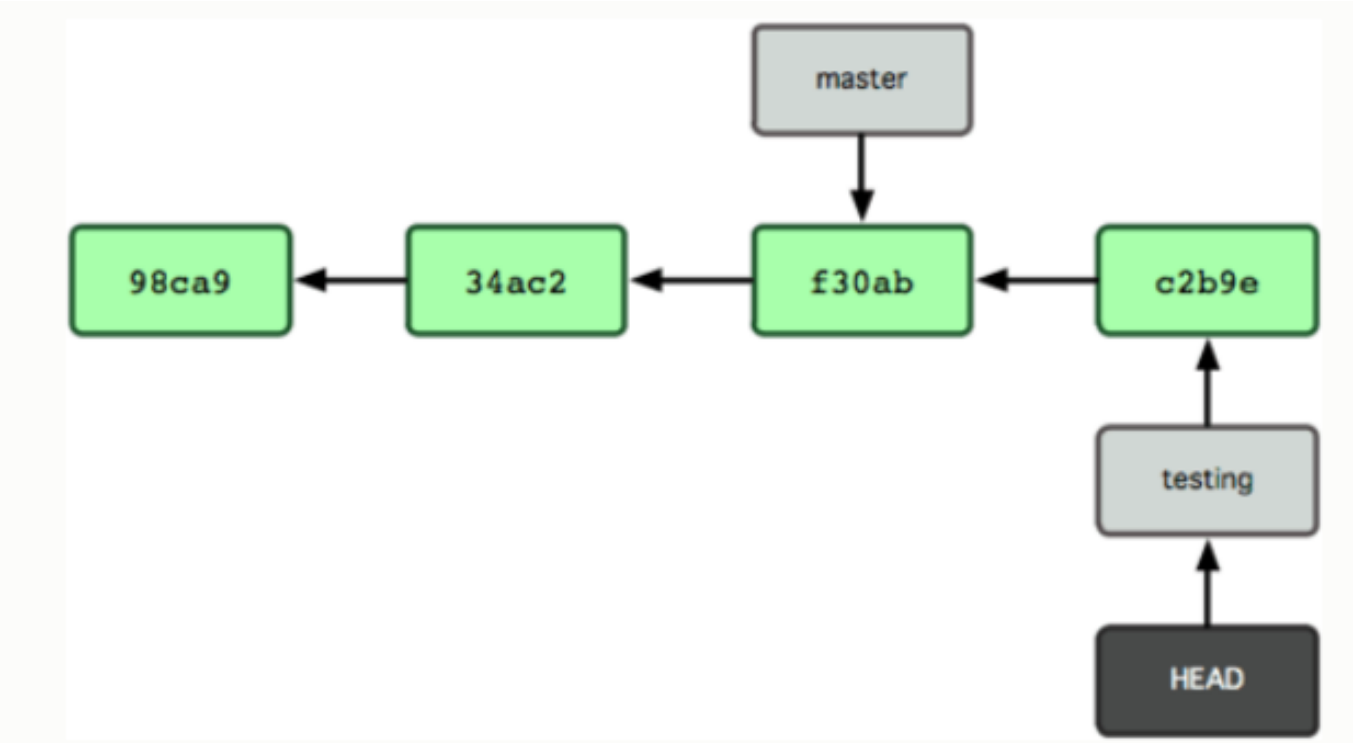
```
git checkout testing
```



BRANCHES

Der Branch, auf den HEAD zeigt, bewegt sich vorwärts, wenn ein Commit gemacht wird

```
 vim test.rb
git commit -a -m 'made a change'
```

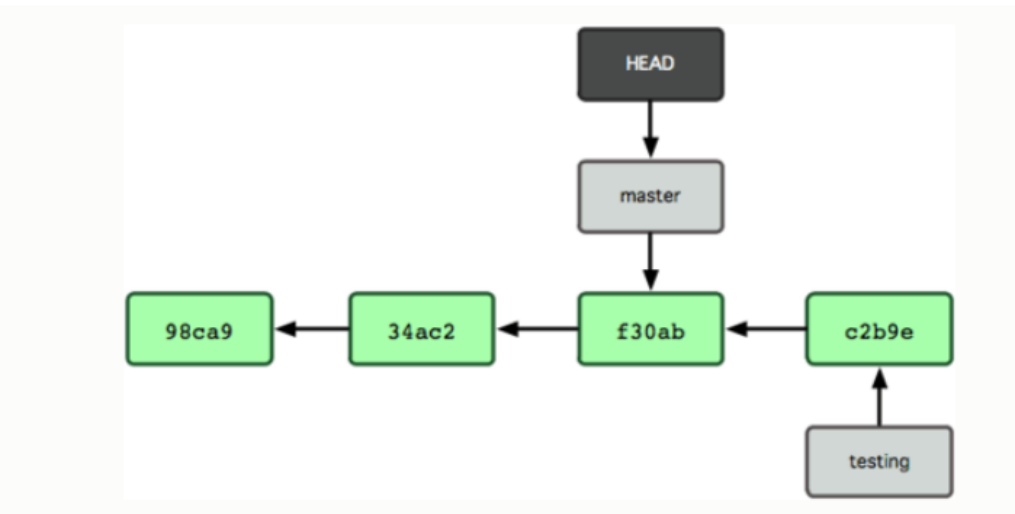


BRANCHES

HEAD bewegt sich, wenn Sie auschecken

Der Branch, auf den HEAD zeigt, bewegt sich vorwärts, wenn ein Commit gemacht wird
Wechseln zum master-Branch.

```
git checkout master
```

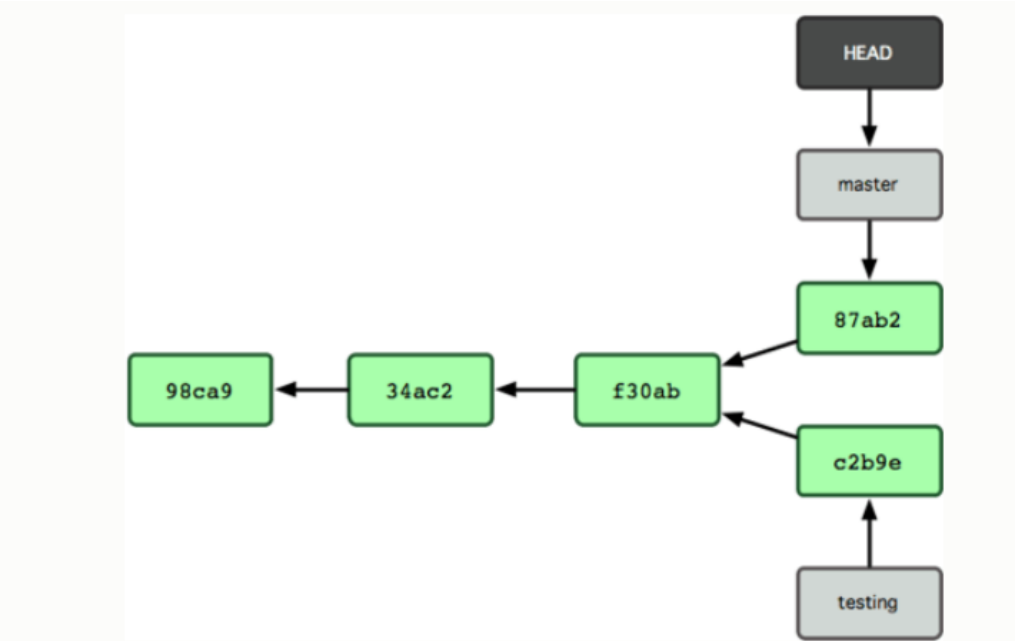


BRANCHES

Verzweigter Verlauf

Wen Sie jetzt noch weitere Änderungen hinzufügen, entsteht ein verzweigter Verlauf

```
vim test.rb
git commit -a -m 'made other changes'
```



MERGING?

Dazu mehr im Praktikum

LITERATUR

nützliche Bücher und Links

→ https://git-scm.com/book/en/v2

→ http://gitbu.ch/pr01.html

ABKÜRZUNGEN

VERSIONIERUNG

VCS Version Control System

DVCS Verteilte Versionsverwaltungsysteme

CVCS Zentralisierte Versionskontrolle