

Vorlesung Unix-Praktikum

7. Git - globales Arbeiten, Branches

Dirk Frettlöh

Technische Fakultät
Universität Bielefeld

27. November 2019

Willkommen zur siebten Vorlesung

Was gab es beim letzten Mal?

- ▶ Versionskontrolle
- ▶ git - Prinzipien, lokales Arbeiten

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Willkommen zur siebten Vorlesung

Was machen wir heute?

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Git

Verteiltes Arbeiten

Rückgängig

Branches

Wie Leute git auch benutzen

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches



Git - Verteiltes Arbeiten

Remote Repositories

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Siehe man `giteveryday`:

A developer working as a participant in a group project needs to learn how to communicate with others, and uses these commands in addition to [the ones above]

- ▶ `git-clone` from the upstream to prime your local repository.
- ▶ `git-pull` and `git-fetch` from origin to keep up-to-date with the upstream.
- ▶ `git-push` to shared repository
- ▶ `git-format-patch` to prepare e-mail submission
- ▶ `git-send-email` to send your e-mail submission
- ▶ `git-request-pull` to create a summary of changes for your upstream to pull

Git - Verteiltes Arbeiten

Globales Repository klonen

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Um mit anderen zusammenzuarbeiten gibt es wieder extrem viele Möglichkeiten. Hier aber nur eine häufige.

Eine weitere häufige Möglichkeit ist, dass es einen git-Manager gibt. Dann reicht es für den Normalnutzer ebenso aus, das Folgende zu wissen.

Ein zentrales (“globales”) Repository.

Um die Daten von dort auf den eigenen Rechner zu bekommen (nur einmal am Anfang der Projekts):

```
$ git clone https://github.com/pfad/projekt  
(per https) oder
```

```
$ git clone juser@files.techfak.de:/pfad/projekt  
(per ssh) oder
```

```
$ git clone /vol/lehre/pfad/projekt      (lokal)
```

Danach hat man ein neues Verzeichnis (normalerweise projekt bzw der Name am Ende der Pfade oben)

Git - Verteiltes Arbeiten

Globale Repositorien

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Das Repo, das wir klonen, heißt *origin*.

```
$ git remote
```

```
origin
```

```
$ git status
```

```
On branch master
```

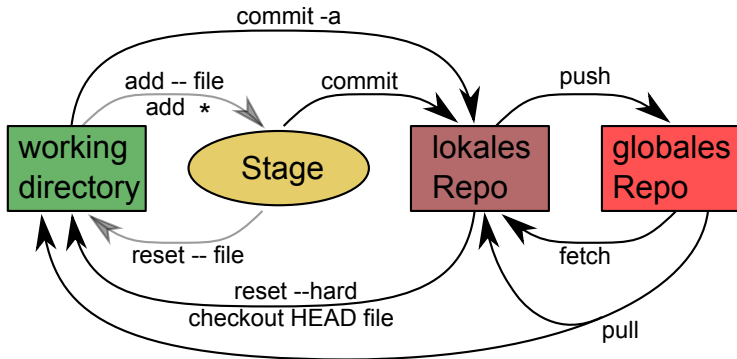
```
Your branch is up-to-date with 'origin/master'.
```

```
nothing to commit, working directory clean
```

(Es kann auch mit mehreren globalen Repos gearbeitet werden, das zeigen wir hier nicht.)

Git - Verteiltes Arbeiten

Schema



Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Git - Verteiltes Arbeiten

fetch und pull

Zwei Möglichkeiten, die aktuelle Version von origin zu holen:

```
git fetch
```

```
git pull
```

Unterschied:

fetch holt die Dateien, aber fügt sie nicht unbedingt mit meinen lokalen Dateien zusammen: parallele Versionen (branches) möglich.

pull holt die Dateien und fügt (*merget*) sie mit meinen lokalen Dateien zusammen.

(Das ist auch das Ergebnis von clone: lokale Version = globale Version, bzw master = origin)

Git - Verteiltes Arbeiten

push und pull

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Umgekehrt fügt push die lokalen Änderungen dem globalen Repo hinzu (falls ich Schreibberechtigung habe)

```
$ git push
```

(kurz für `git push origin master`, in komplexeren Situationen `git push remote branch`)

Ein einfacher Arbeitsablauf wäre also (ein branch, master = origin)

1. `git clone /vol/lehre/Linux/projekt`
2. Dateien bearbeiten
3. `git add Datei ; git commit`
4. `git pull`
5. `git push`
6. GOTO 2.

(Zeigen)

Git - Verteiltes Arbeiten

push und pull

...wenn alles gutgeht. Was ist, wenn zwei Leute A und B gleichzeitig Dateien ändern?

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Git - Verteiltes Arbeiten

push und pull

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Fall 1: A ändert Datei eins.dat, B ändert Datei zwei.dat

A pusht zuerst, das wird durchgeführt.

Jetzt ist im globalen Repo eins.dat neu

B pusht danach, das wird nicht durchgeführt.

Also handelt B wie oben: erst `git pull`:

Dann repariert git das: pull ist holen (*fetch*) und zusammenfügen (*merge*)

Kein Problem: eins.dat neu wird geholt, zwei.dat wird durch Bs neue Version ersetzt. Jetzt sind im wd und im lokalen Repo eins.dat neu und zwei.dat neu

Nun kann B `git push` ausführen. (Zeigen)

Jetzt ist auch im globalen Repo zwei.dat neu

git - Verteiltes Arbeiten

push und pull

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Fall 2: A ändert Datei eins.dat in Zeile 1, B ändert Datei eins.dat in Zeile 3

A: `git pull` ; `git push`, das wird durchgeführt.

B: `git pull`:

Git repariert auch das: “Merge branch master of [remote]”
(Zeigen)

Git - Verteiltes Arbeiten

Versionen

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

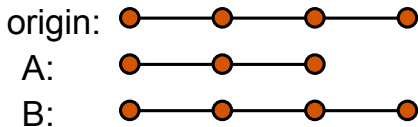
Rückgängig

Branches

Von Hand reparieren, dann

```
git add eins.dat; git commit; git push
```

Bislang: immer nur ein branch. Ein möglicher Verlauf:



Git - Verteiltes Arbeiten

push und pull

Unix-
Praktikum

Dirk Frettlöh

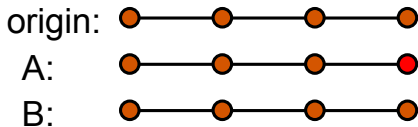
Git

Verteiltes Arbeiten

Rückgängig

Branches

A pullt:



Git - Verteiltes Arbeiten

push und pull

Unix-
Praktikum

Dirk Frettlöh

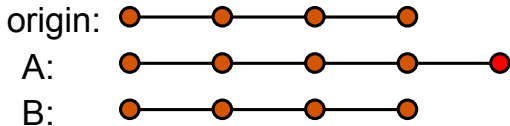
Git

Verteiltes Arbeiten

Rückgängig

Branches

A commitet was neues:



Git - Verteiltes Arbeiten

push und pull

Unix-
Praktikum

Dirk Frettlöh

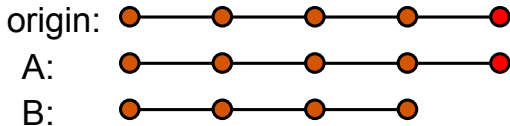
Git

Verteiltes Arbeiten

Rückgängig

Branches

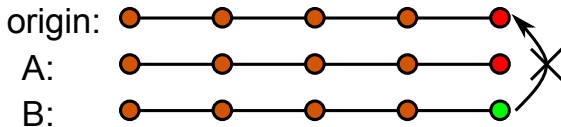
A pusht:



Git - Verteiltes Arbeiten

push und pull

B macht was neues und will pushen:



Missrät, da B nicht auf dem letzten Stand ist.

Git - Verteiltes Arbeiten

push und pull

Unix-
Praktikum

Dirk Frettlöh

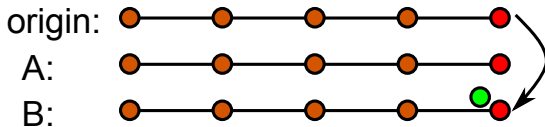
Git

Verteiltes Arbeiten

Rückgängig

Branches

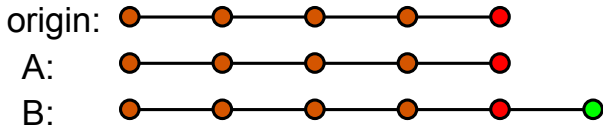
B pullt:



Git - Verteiltes Arbeiten

push und pull

B merget (oder macht was neues):



Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Git - Verteiltes Arbeiten

push und pull

Unix-
Praktikum

Dirk Frettlöh

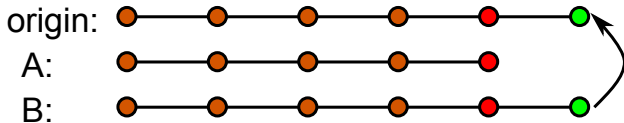
Git

Verteiltes Arbeiten

Rückgängig

Branches

B pusht:



usw.

Git - Verteiltes Arbeiten

push und pull

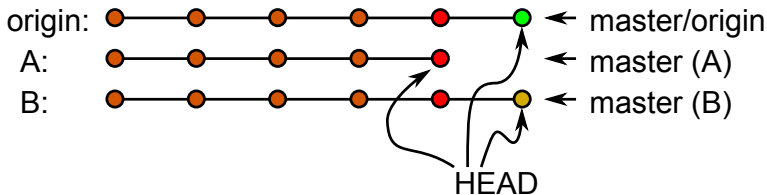
- ▶ git-clone from upstream to prime your local repository. ✓
- ▶ git-pull and git-fetch from origin to keep up-to-date with the upstream. ✓
- ▶ git-push to shared repository ✓
- ▶ git-format-patch to prepare e-mail submission
- ▶ git-send-email to send your e-mail submission
- ▶ git-request-pull to create a summary of changes for your upstream to pull

Recall: fetch ist pull ohne merge

Git - Dinge rückgängig machen

HEAD und master

Bezeichnungen:



Die aktuellste Version ist HEAD.

Git - Dinge rückgängig machen

HEAD und master

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Die früheren Versionen (*commits*) haben als Namen *hashes*:

```
$ git log
```

```
commit 8d2f775a1d18bbdd02951e25cfa575b0f8aebf43
```

```
Author: Dirk Frettlöh <dfrettloeh@techfak.uni-bielefeld.de>
```

```
Date: Mon Jan 16 16:21:55 2017 +0100
```

```
    eins.dat repariert
```

```
commit 4c1f5e121a6f1c8b7af74013c983f4325aa69a25
```

```
Merge: 1a61068 df96eac
```

```
Author: Dirk Frettlöh <dfrettloeh@techfak.uni-bielefeld.de>
```

```
Date: Mon Jan 16 16:20:01 2017 +0100
```

```
    drei.dat ist nun korrekt
```

```
commit 1a6106815ec314f07253f5ba08fc3e7bc554f15e
```

```
Author: Dirk Frettlöh <dfrettloeh@techfak.uni-bielefeld.de>
```

```
Date: Mon Jan 16 16:18:30 2017 +0100
```

Git - Dinge rückgängig machen

commits und hashes

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Oben ist also `8d2f775a1d18bbdd02951e25cfa575b0f8aebf43`
dasselbe wie HEAD (letzter commit).

Wenn ich im wd eine Datei lösche:

```
git reset --hard (setzt wd auf aktuelle Version im  
lokalen repo)
```

ist dasselbe wie

```
git reset --hard HEAD
```

ist dasselbe wie

```
git reset --hard 8d2f775a1d18bbdd02951e25cfa575...
```

Git - Dinge rückgängig machen

commits und hashes

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

```
git reset --hard 8d2f775a1d18bbdd02951e25cfa575...
```

ist natürlich etwas zu lang. Es geht auch

```
git reset --hard 8d2f77
```

Regel: die ersten n Zeichen ($n \geq 4$), so dass es eindeutig ist.

Das alles setzt auf letzten commit im lokalen repo zurück.

```
git reset --hard origin/master
```

Das setzt auf letzten commit im globalen repo zurück. (Zeigen)

Git - Dinge rückgängig machen

Weitere undos

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Es gibt noch viel mehr Möglichkeiten, Dinge rückgängig zu machen. Eine kleine Auswahl:

Situation: Ich habe gerade etwas gepusht und gemerkt, dass es Mist war. Der commit hash war a1b2c3...

```
$ git revert a1b2c3
```

```
$ git push
```

Git - Dinge rückgängig machen

Weitere undos

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Situation: Ich habe gerade etwas committed, noch nicht gepusht, und gemerkt, dass mein Kommentar ("eins.dat geixft") Mist war.

```
$ git commit --amend -m ''eins.dat gefixt''
```

(ersetzt meinen letzten commit durch das aktuell gestagete - hier: nix gestaged, also: ersetzt nur den Kommentar)

Situation: Ich habe gerade versehentlich eins.dat aus meinem wd gelöscht. Mist.

```
$ git checkout -- eins.dat
```

(Siehe oben: holt eins.dat aus dem lokalen git)

Git - Dinge rückgängig machen

Weitere undos

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Situation: Meine letzten drei commits waren alle Mist. Ich möchte zurück auf den Zustand vorher.

```
$ git reset hash
```

(setzt zurück auf commit *hash*. Dateien im wd sind immer noch Mist.)

Oder: `$ git reset --hard hash`

(setzt zurück auf commit *hash*. Dateien im wd sind auch wieder wie damals.)

Git - Dinge rückgängig machen

Weitere undos

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Situation: (siehe oben) Ich habe eine Datei `mist.txt` commitet, die ich gar nicht im repo haben möchte.

```
git rm mist.txt  
git commit -m "remove mist.txt"
```

(git rm nimmt sie aus dem repo und aus dem wd raus, aber sie ist noch gestaget.)

Situation: Ich habe eine Datei `mist.txt` gestaget, aber noch nicht commitet, die ich gar nicht im repo haben möchte.

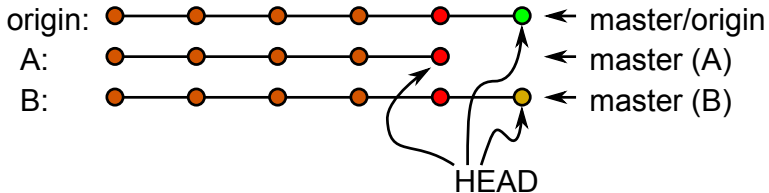
```
git rm --cached mist.txt
```

(git rm `--cached` nimmt sie nur aus dem Stage-Bereich ("Index") raus)

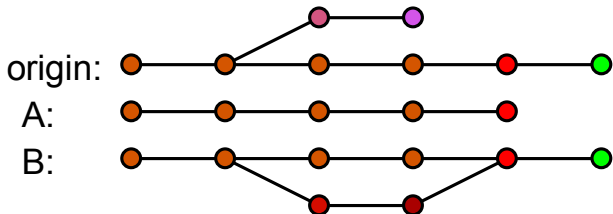
Git - Branches

HEAD und branch

Bisher keine Verzweigungen (*branches*). Bezeichnungen:



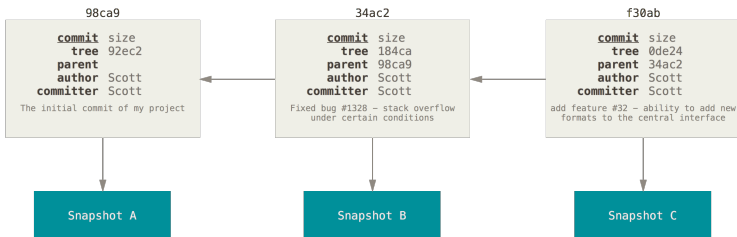
Wir wollen nun auch Versionen verzweigen können:



Git - Branches

HEAD und branch

Genauer Bild des lokalen master branches:



Git - Branches

HEAD und branch

Unix-
Praktikum

Dirk Frettlöh

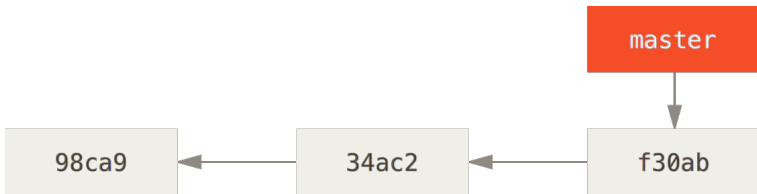
Git

Verteiltes Arbeiten

Rückgängig

Branches

Größeres Bild des lokalen master branches:



Git - Branches

HEAD und branch

Unix-
Praktikum

Dirk Frettlöh

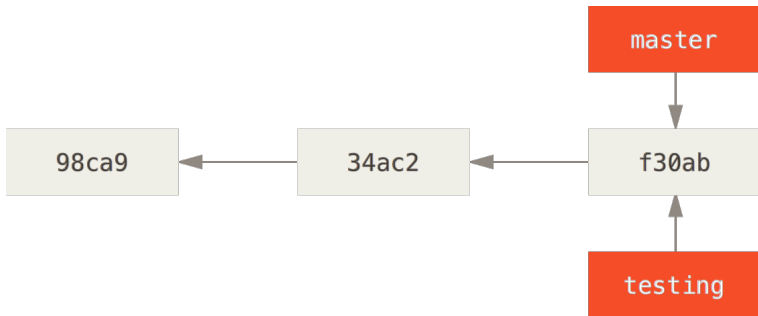
Git

Verteiltes Arbeiten

Rückgängig

Branches

Neuen Branch erzeugen: `$ git branch testing`

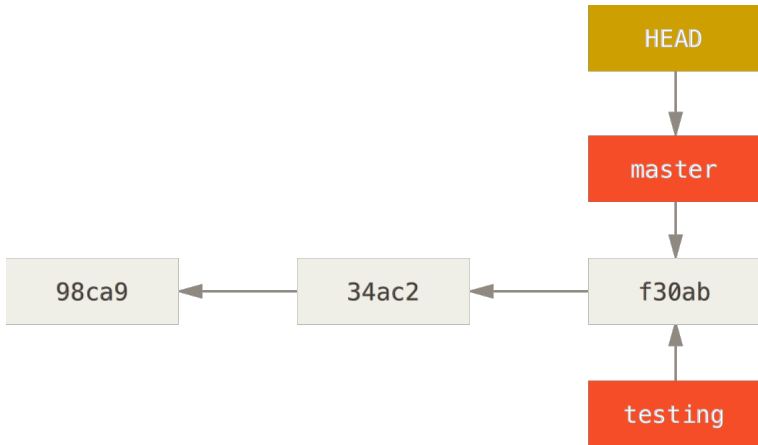


Es wird also keine Kopie aller Snapshots angelegt, sondern nur ein neuer Zeiger erzeugt.

Git - Branches

HEAD und branch

Wohin zeigt HEAD?

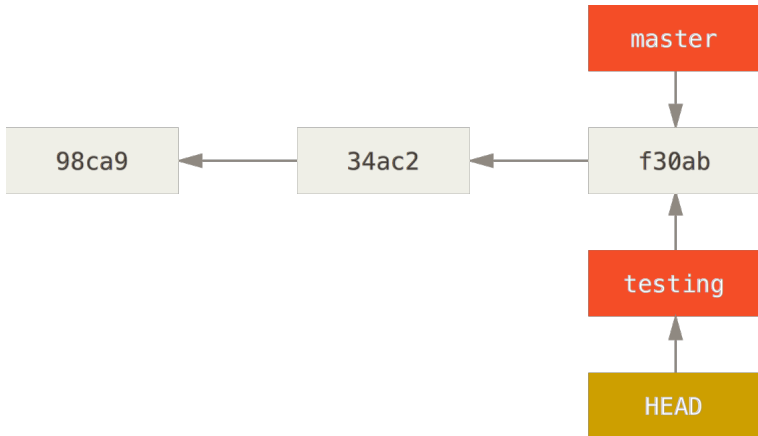


Wir "befinden" uns noch im Branch master.

Git - Branches

HEAD und branch

```
$ git checkout testing
```

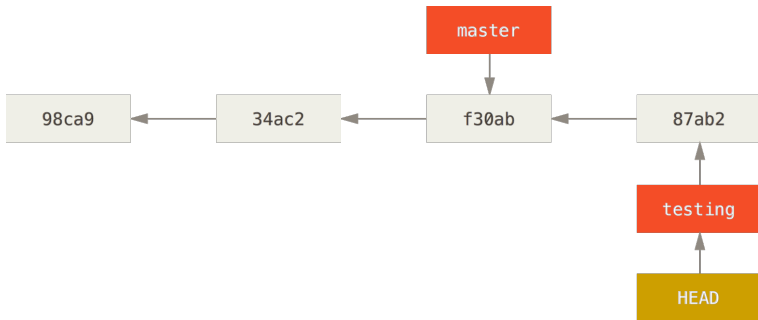


Jetzt "befinden" wir uns im Branch testing.

Git - Branches

HEAD und branch

```
$ emacs test.rb  
$ git commit -a -m 'test.rb geändert'  
$ git push --set-upstream origin testing  
(dem remote-repo mitteilen: wir arbeiten aktuell in testing)
```

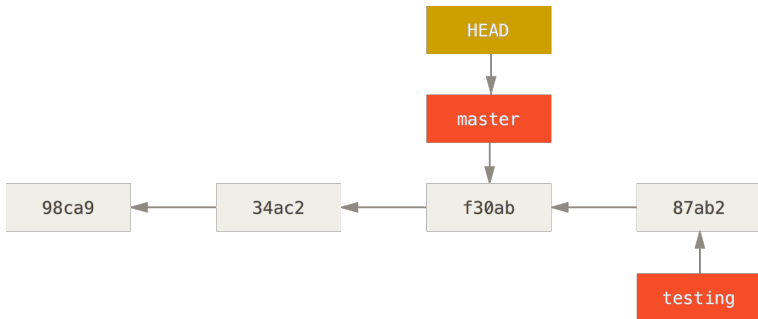


... und arbeiten im Branch testing.

Git - Branches

HEAD und branch

```
$ git checkout master
```



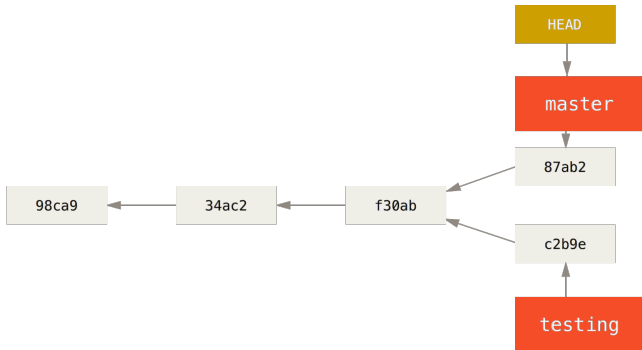
Wir wechseln zurück zu master.

Git - Branches

HEAD und branch

```
$ emacs test.rb
```

```
$ git commit -a -m 'test.rb verbessert'
```



... und arbeiten nun im Branch master. Usw.

Git - Branches

HEAD und branch

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Eine primitive Anzeige bietet git log:

```
$ git log --oneline --decorate --graph --all
```

```
$ git log --oneline --decorate --graph --all
* c2b9e (HEAD, master) made other changes
| * 87ab2 (testing) made a change
|/
* f30ab add feature #32 - ability to add new formats
* 34ac2 fixed bug #1328 - stack overflow under certain
* 98ca9 initial commit of my project
```

Git - Branches

merge branches

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Verschiedene branches können gemerget werden:

```
$ git checkout master (in branch master wechseln)
```

```
$ git merge testing (testing in master einfügen)
```

Jetzt stimmen master und testing überein, also kann testing auch wieder gelöscht werden:

```
$ git branch -d testing
```

Git - Branches

merge branches

Ein Beispielszenario (vgl. Pro Git, Kapitel 3):

- ▶ A arbeitet an einer Webseite.
- ▶ Erzeugt dazu einen neuen branch `iss53` und arbeitet darin
- ▶ Es kommt ein Anruf: dringend etwas an der eigentlichen Webseite reparieren
- ▶ A wechselt zurück zu `master` und erzeugt da einen neuen branch `hotfix`
- ▶ Repariert in `hotfix` den Fehler, testet, merget `hotfix` in `master`
- ▶ Arbeitet weiter in `iss53`

Git - Branches

merge branches

Unix-
Praktikum

Dirk Frettlöh

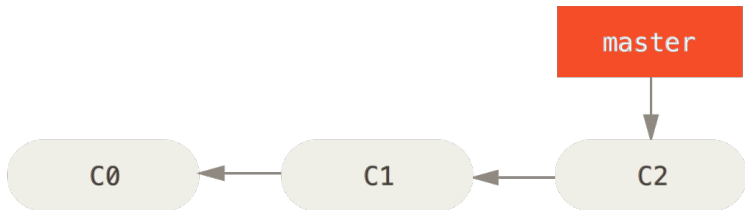
Git

Verteiltes Arbeiten

Rückgängig

Branches

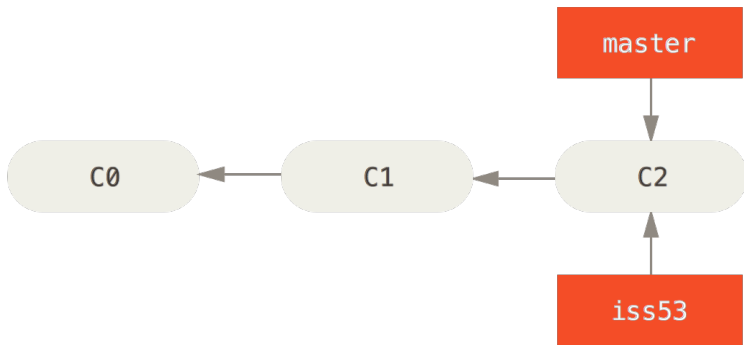
Zustand vorher:



Git - Branches

merge branches

```
$ git checkout -b iss53 (kurz für git branch und git checkout)
```



Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Git - Branches

merge branches

Unix-
Praktikum

Dirk Frettlöh

Git

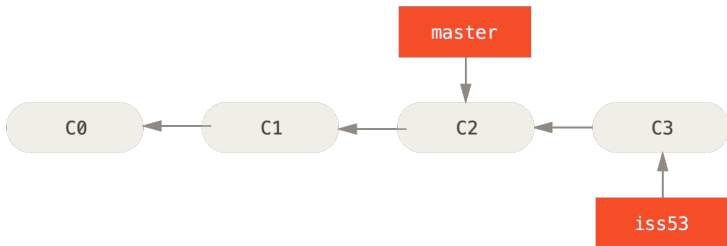
Verteiltes Arbeiten

Rückgängig

Branches

```
$ emacs index.html
```

```
$ git commit -a -m 'added [issue 53]'
```



Git - Branches

merge branches

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

A arbeitet in iss53. Nun kommt der Anruf.

```
$ git checkout master (in master wechseln)
```

Beachte: Das wd sieht nun so aus wie master!

A erzeugt einen neuen branch, behebt dort den Fehler:

Git - Branches

merge branches

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

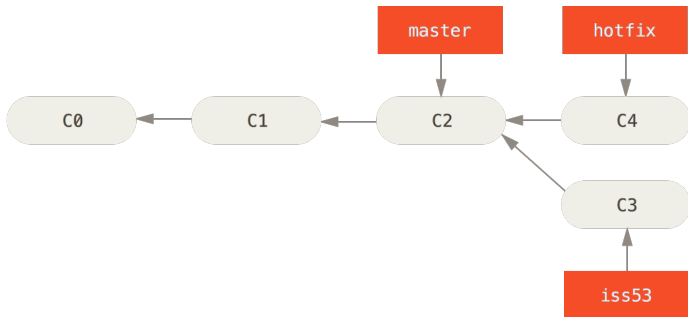
Rückgängig

Branches

```
$ git checkout -b hotfix
```

```
$ emacs index.html
```

```
$ git commit -a -m 'index.html fixed'
```

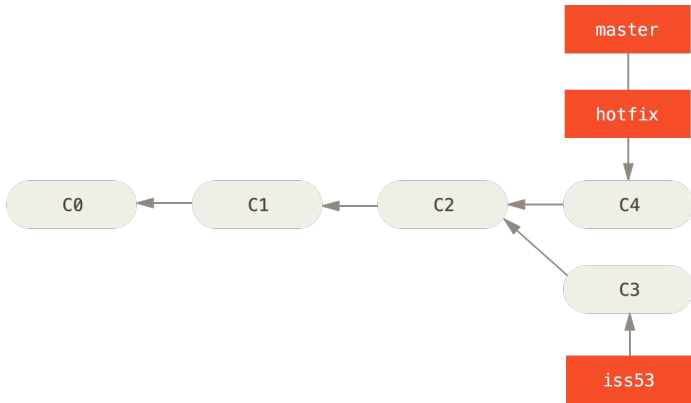


Git - Branches

merge branches

```
$ git checkout master
```

```
$ git merge hotfix (hotfix in master einfügen)
```



(Kein Konflikt, “fast-forward” nach hotfix)

Git - Branches

merge branches

Unix-
Praktikum

Dirk Frettlöh

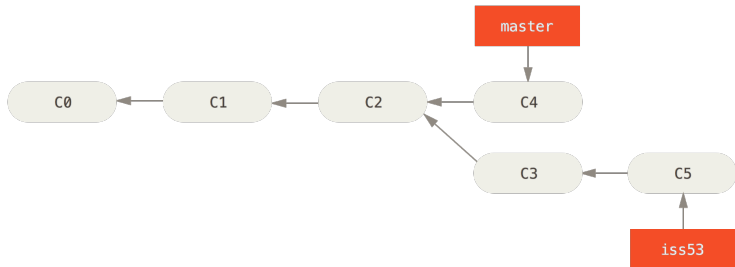
Git

Verteiltes Arbeiten

Rückgängig

Branches

```
$ git branch -d hotfix (hotfix löschen)  
$ git checkout iss53 (nach iss53 wechseln)  
$ emacs index.html (weiterarbeiten)  
$ git commit -a -m 'finished [issue 53]'
```



Git - Branches

merge branches

Unix-
Praktikum

Dirk Frettlöh

Git

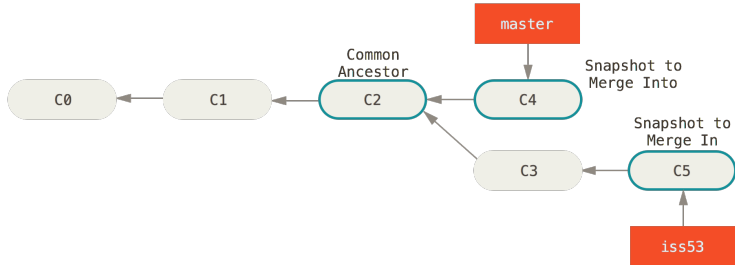
Verteiltes Arbeiten

Rückgängig

Branches

```
$ git checkout master (nach master wechseln)
```

```
$ git merge iss53 (iss53 in master einfügen)
```

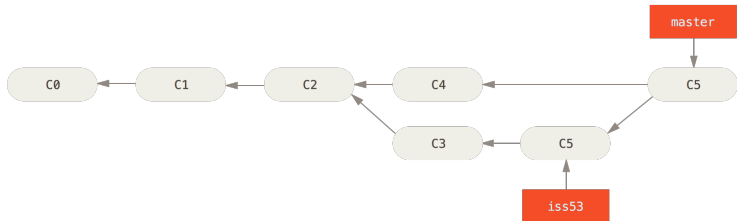


Git - Branches

merge branches

master und iss53 sind zwei verschiedene Zweige, “fast-forward” geht hier nicht.

Also mergen, Konflikte lösen, siehe oben.



Git II - Überblick

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

`git clone`

ein repo auf den eigenen
Rechner holen

`git pull, fetch, push`

Änderungen holen / sichern

`git rm, revert, reset...`

Änderungen rückgängig machen

Branches:

parallele Versionen des Projekts

Ende der heutigen Vorlesung

Unix-
Praktikum

Dirk Frettlöh

Git

Verteiltes Arbeiten

Rückgängig

Branches

Vielen Dank fürs Zuhören!

Bis nächste Woche!