Dienstag: Versionskontrolle

5 Versionskontrolle

Projekte

- Dauern lang
- Involvieren mehrere Personen und
- Verzweigen manchmal

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Probleme aus dem täglichen (Projekt-)Leben

- Personen überschreiben gegenseitig ihre Änderungen
- Neuer Code macht alten Code kaputt
- Wer hat eigentlich wann was geändert?
- Wie sah die Datei eigentlich aus, bevor ich Algorithmus X implementiert habe?

Versionsverwaltung

- Protokollieren alle Änderungen
- Verhindern das gegenseitige Überschreiben
- Können automatisch Änderungen von verschiedenen Leuten an der gleichen Datei zusammenführen (manchmal)
- Stellen Unterschiede zwischen Versionen dar

Systeme

Es gibt eine Vielzahl von Versionsverwaltungssystemen:

- CVS (Concurrent Versioning System)
- BitKeeper (proprietär, wurde vom Linux-Kernel verwendet)
- Bazaar (Wird von der Ubuntu-Distribution verwendet)
- SVN (Subversion, als Ersatz für CVS entwickelt)
- git (frei, wird heute für den Linux-Kernel verwendet)

Systeme

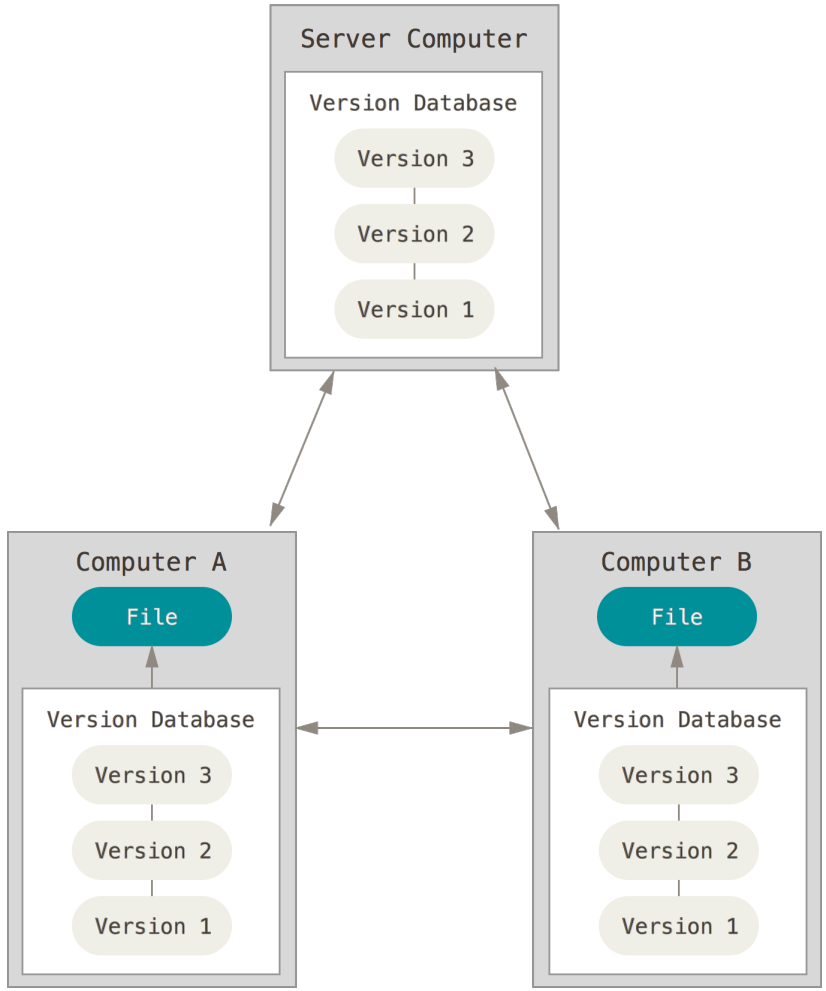
Es gibt eine Vielzahl von Versionsverwaltungssystemen:

- CVS (Concurrent Versioning System)
- BitKeeper (proprietär, wurde vom Linux-Kernel verwendet)
- Bazaar (Wird von der Ubuntu-Distribution verwendet)
- SVN (Subversion, als Ersatz für CVS entwickelt)
- git (frei, wird heute für den Linux-Kernel verwendet)

git

- *verteiltes* System: jeder Benutzer besitzt eine Kopie des gesamten Repositorys
  - stets Zugriff auf komplette Versionsgeschichte
  - ermöglicht *Offline*-Entwicklung
- Dokumentation: `https://git-scm.com/doc`
- man arbeitet stets in seiner Kopie des Repositorys

Übersicht



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Repository

- Man kann entweder selber ein Repository erstellen:

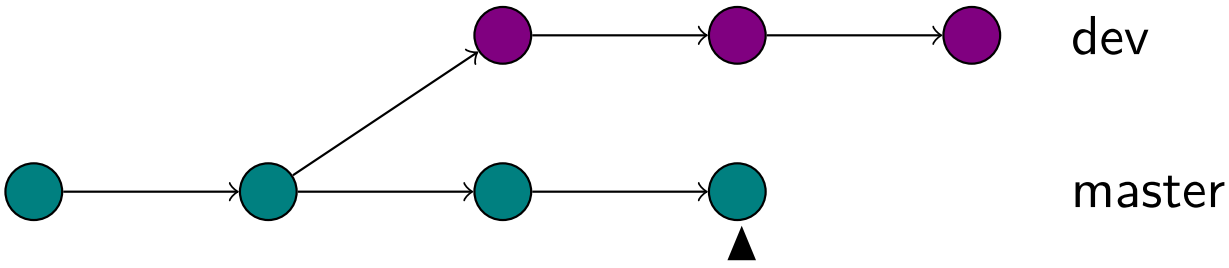
```
:~$ git init <folder>
```

- Oder man “klont” ein vorhandenes Repository:

```
:~$ git clone <url>
```

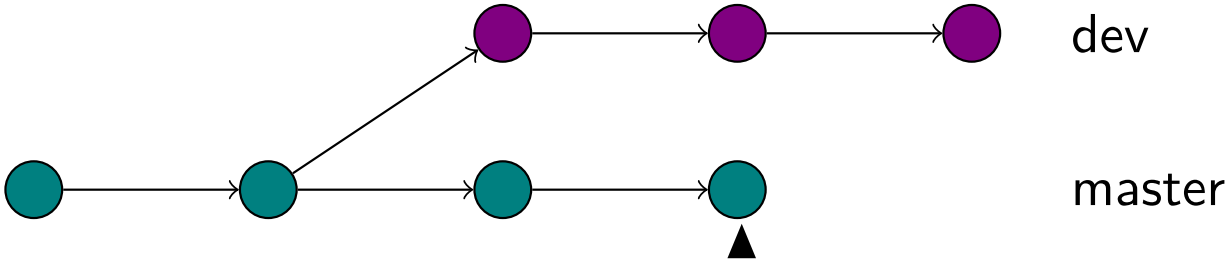
Visualisierung

Remote auf dem Server (**`origin`**):

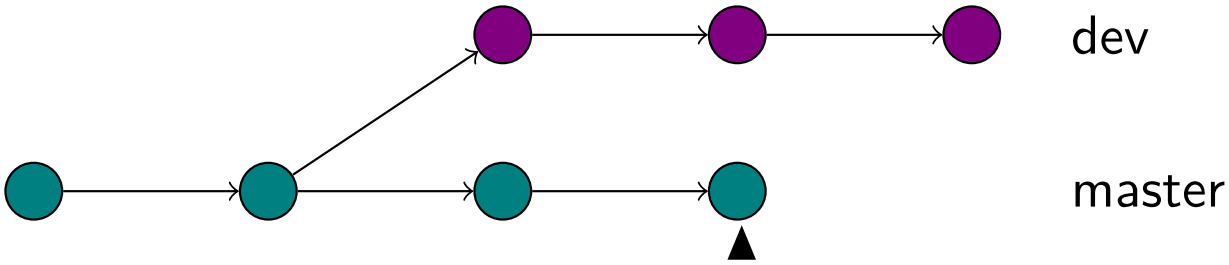


Visualisierung

Remote auf dem Server (**`origin`**):



Lokal nach **`git clone`**:



Paradigma

- Man arbeit in der lokalen Kopie des Repositorys
- Dateien, die man geändert oder hinzugefügt hat, markiert man mit `:~$ git add <filename>`
- Um die so markierten Änderungen zu speichern, benutzt man dann `:~$ git commit`
  - eine Protokollnachricht **muss** angegeben werden! (Mit `-m MESSAGE` oder in einem Editor)

Protokollnachrichten

- Zu jeder Revision eine Protokollnachricht (*log message*)
- Es erhöht die Übersichtlichkeit beträchtlich, etwas sinnvolles anzugeben
- gut: “Methode Klasse123.getABC() hinzugefügt”, “Fehler 123 gefixt”
- schlecht: “Änderungen gemacht”, “Klasse123 repariert”

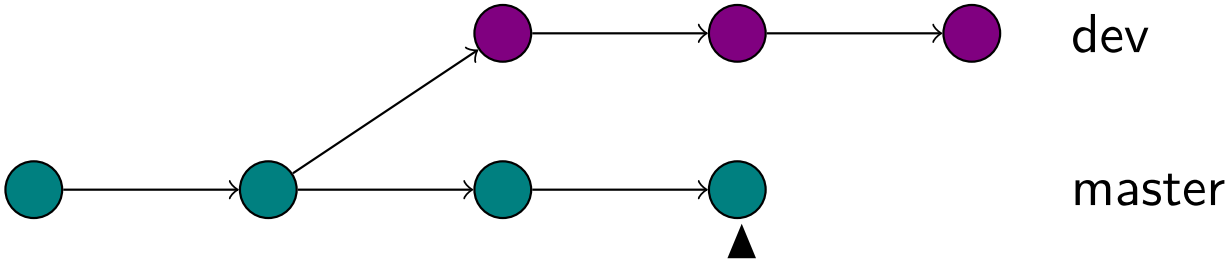
remove

```
:~$ git rm <filename>
```

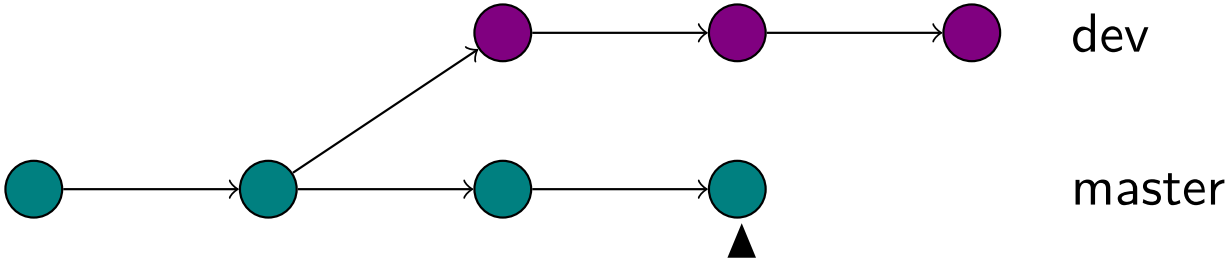
- Dateien unter Versionskontrolle dürfen **nicht** einfach gelöscht werden!
- Dateien müssen per git gelöscht werden

Visualisierung: Comitten

Remote auf dem Server (**`origin`**):

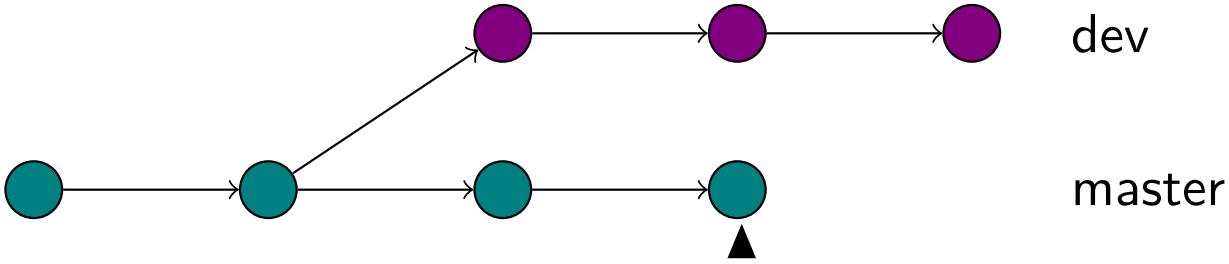


Lokal vor **`git commit`**:

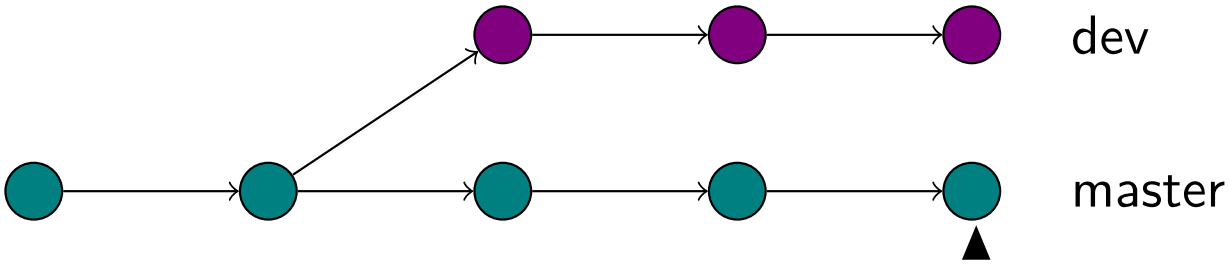


Visualisierung: Comitten

Remote auf dem Server (**`origin`**):



Lokal nach **`git commit`**:



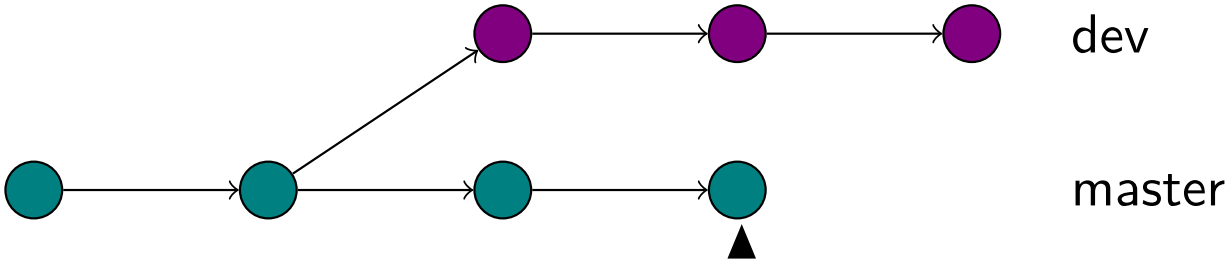
push

```
:~$ git push <remote>
```

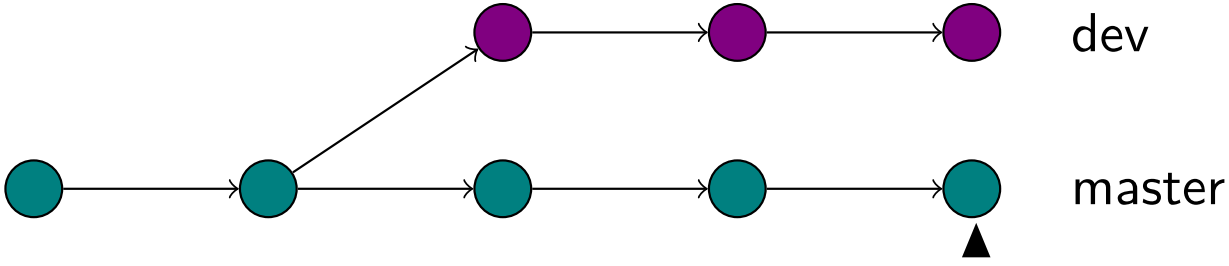
- aktualisiert das Repository names **<remote>** mit dem Status der lokalen Version des Repositorys
- default: **origin**
- Dateikonflikte sind möglich, falls jemand anderes schon etwas aktualisiert hat – dazu kommen wir später

Visualisierung: push

`origin` vor `git push origin`:

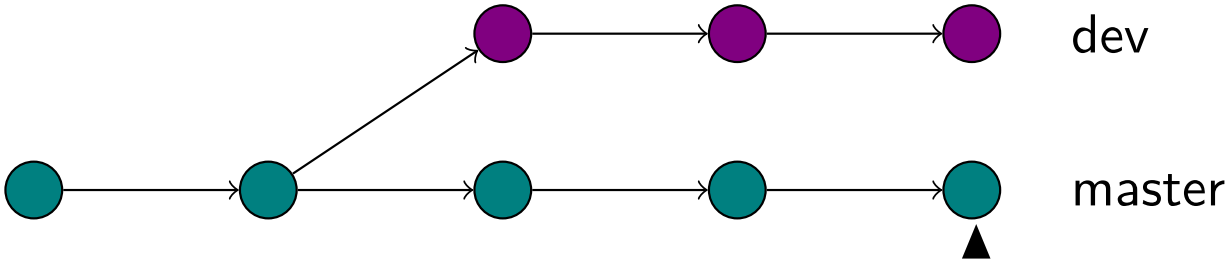


Lokal:

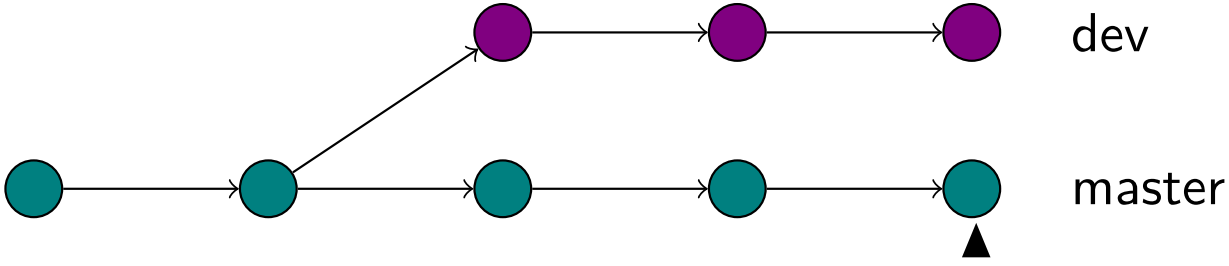


Visualisierung: push

`origin` nach `git push origin`:



Lokal:



fetch

```
:~$ git fetch <remote>
```

- aktualisiert lokale Information über **<remote>**
- Dateien werden noch nicht geändert!

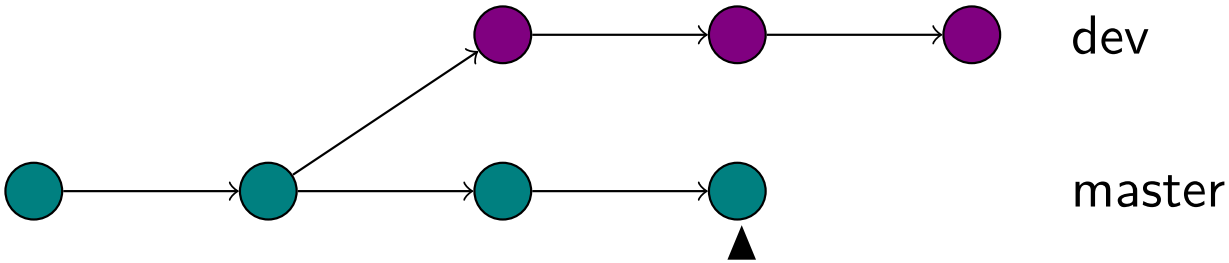
merge

```
:~$ git merge <remote>/<branch>
```

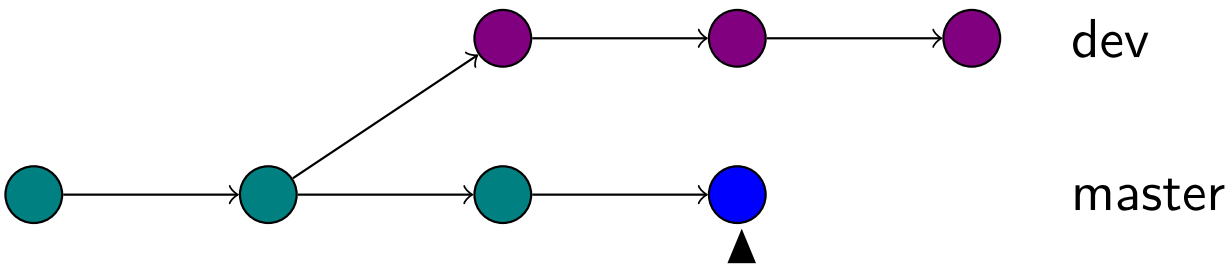
- fügt angegebenen Branch und aktuellen zusammen
- eventuell müssen Dateikonflikte aufgelöst werden
- falls keine Konflikte: es wird automatisch comittet!

Visualisierung: fetch und merge

`origin`:

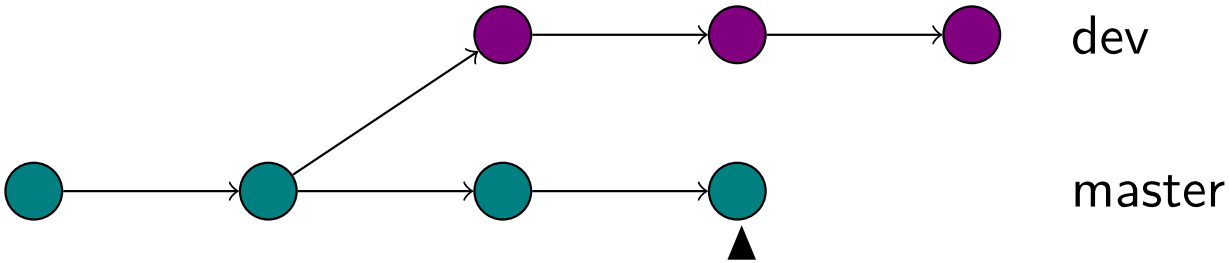


Lokal vor `git fetch origin`:

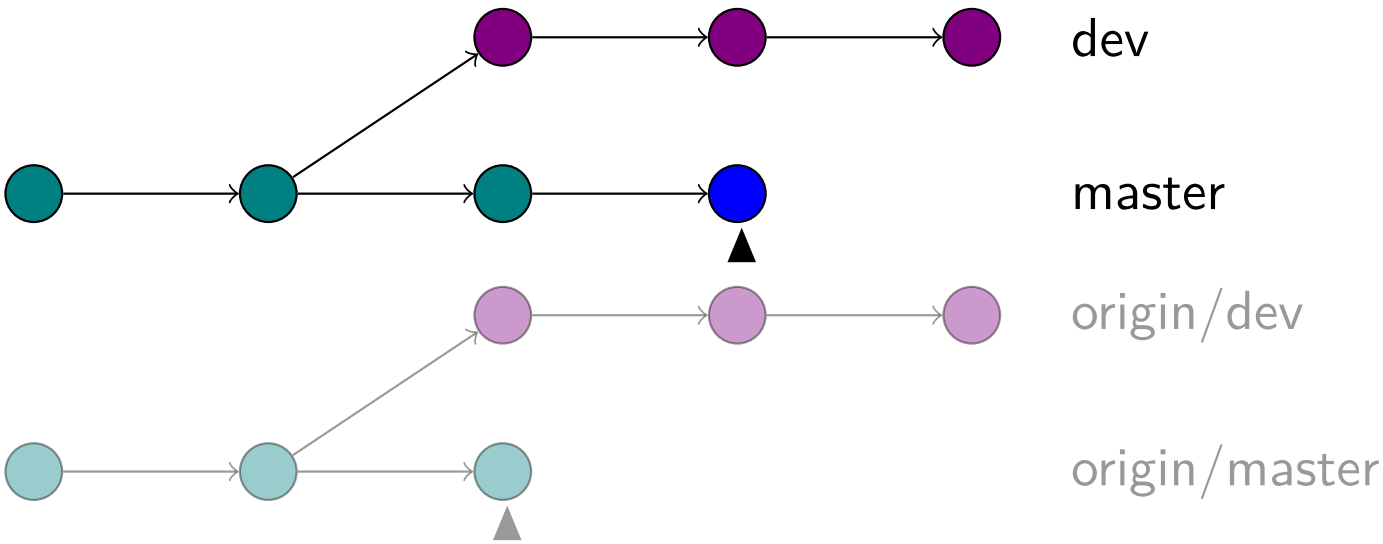


Visualisierung: fetch und merge

`origin`:

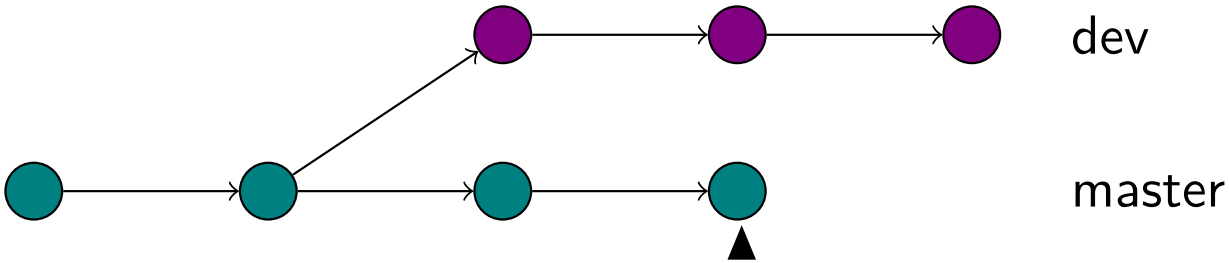


Lokal vor `git fetch origin`:

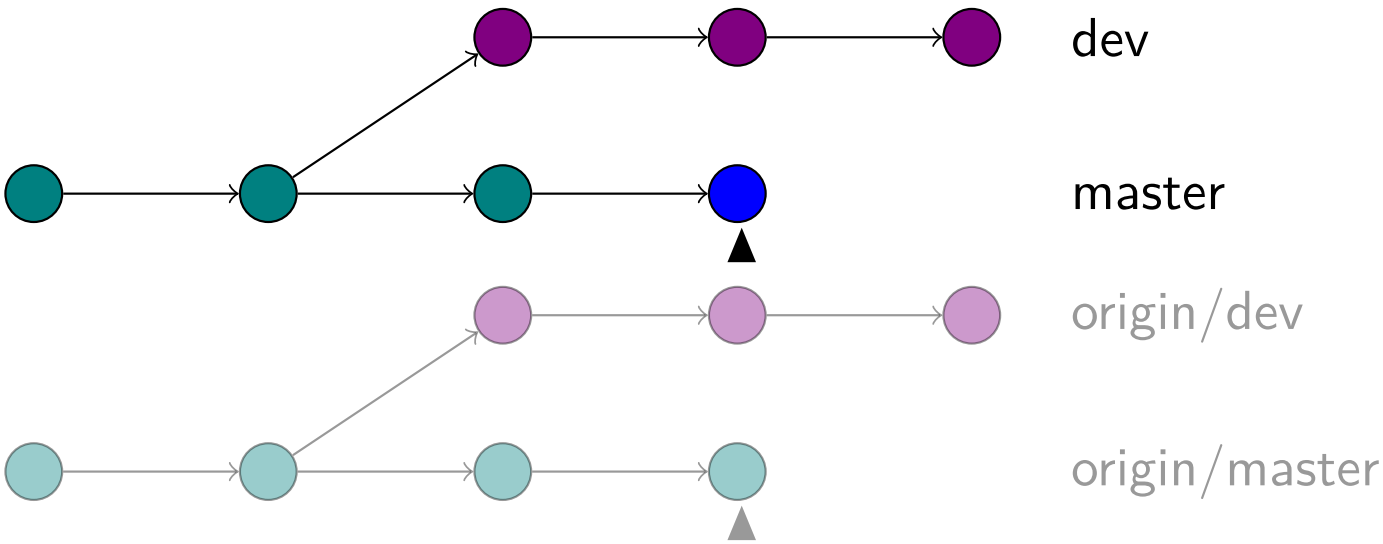


Visualisierung: fetch und merge

`origin`:

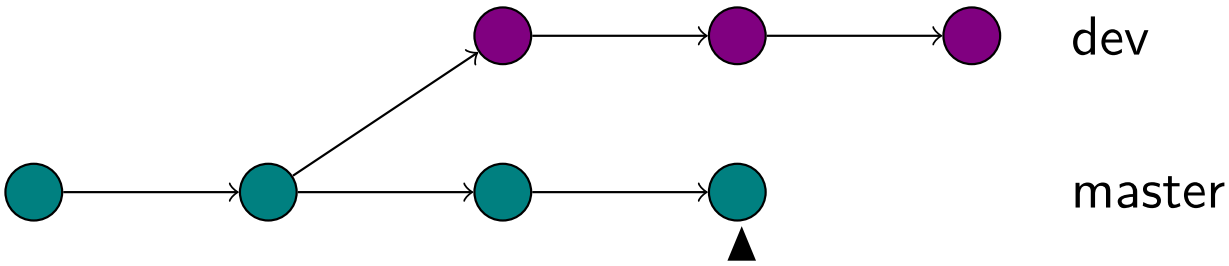


Lokal nach `git fetch origin`:

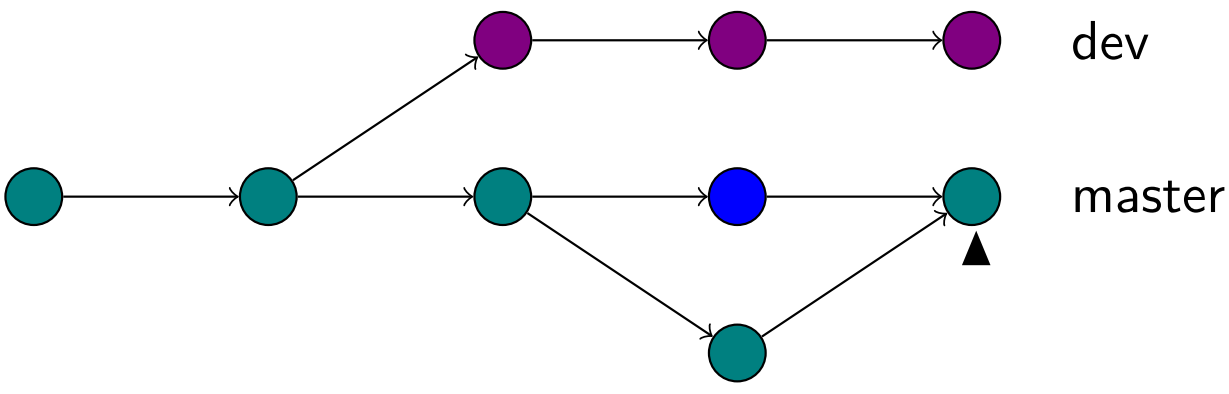


Visualisierung: fetch und merge

`origin`:



Lokal nach `git merge origin/master`:



pull

```
:~$ git pull <remote>/<branch>
```

- ruft zunächst `:~$ git fetch` auf, danach `:~$ git merge`

Branchmanagement

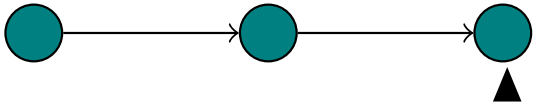
- Manchmal möchte man komplexe Dinge ausprobieren, die nicht mit einer Änderung getan sind
- Dafür kann man einen Branch anlegen
- `:~$ git branch <branch>` legt Branch namens **<branch>** an
- `:~$ git checkout <branch>` wechselt in Branch **<branch>**
- `:~$ git branch -d <branch>` löscht Branch **<branch>**
- der "Standardbranch" ist **master**

Merging von Branches

- Workflow:
  1. `:~$ git branch <branch>`
  2. `:~$ git checkout <branch>`
  3. Arbeiten durchführen und comitten
  4. `:~$ git checkout master`
  5. `:~$ git merge <branch>`
  6. `:~$ git branch -d <branch>`
- Dateikonflikte können auftreten, falls an dem Branch, von dem abgezweigt wurde, weitergearbeitet wurde

Visualisierung: Branches

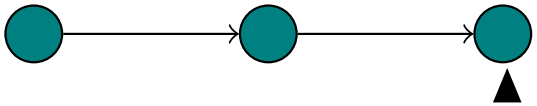
Lokal ursprünglich:



master

Visualisierung: Branches

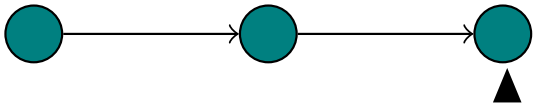
Lokal nach `git branch dev`:



master, dev

Visualisierung: Branches

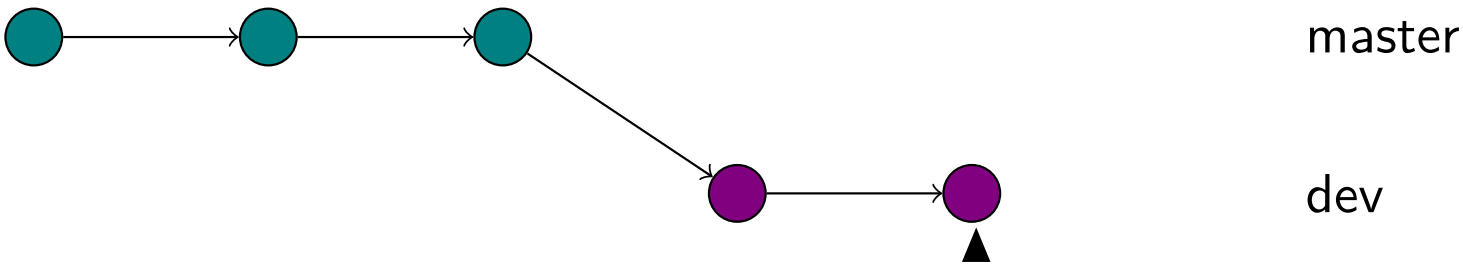
Lokal nach `git checkout dev`:



master, dev

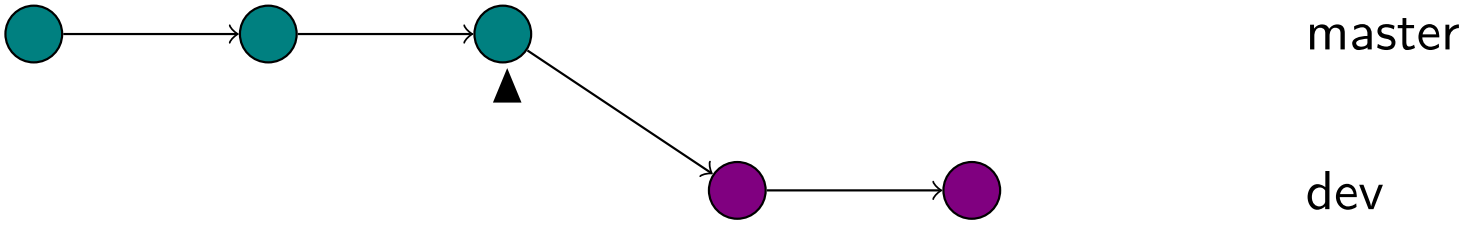
Visualisierung: Branches

Lokal nach Änderungen und mehreren `git commit`:



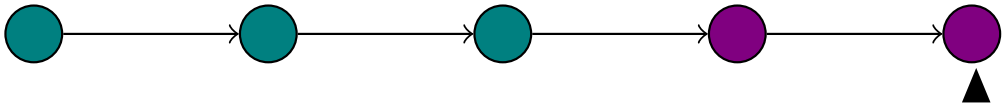
Visualisierung: Branches

Lokal nach `git checkout master`:



Visualisierung: Branches

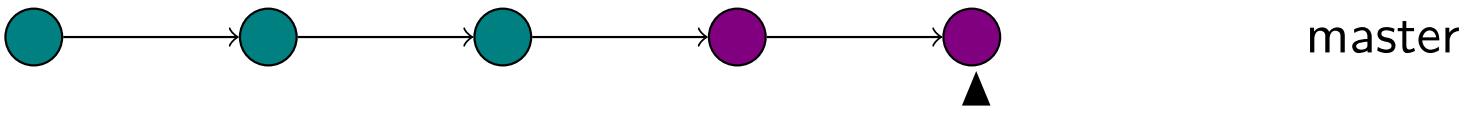
Lokal nach `git merge dev`:



master, dev

Visualisierung: Branches

Lokal nach `git branch -d dev`:



Dateikonflikte auflösen I

- Beim Arbeiten mit verschiedenen Branches und bei der Interaktion mit anderen Versionen des Repositorys können Dateikonflikte auftreten
- `git status` – Konflikte werden angezeigt
- Entweder:
  - in die Datei gehen, Konflikte manuell auflösen oder
  - `git mergetool` benutzen

Dateikonflikte auflösen II

- `:~$ git add <filename>` ausführen, um anzuzeigen, dass die Konflikte aufgelöst wurden
- Zuletzt committen:

```
:~$ git commit -m 'Resolved conflict between ...'
```

Hilfe und Status

```
:~$ git help
```

- Zeigt eine Übersicht der verfügbaren Befehle an

```
:~$ git help <command>
```

- Zeigt Hilfe zu bestimmten Kommandos an

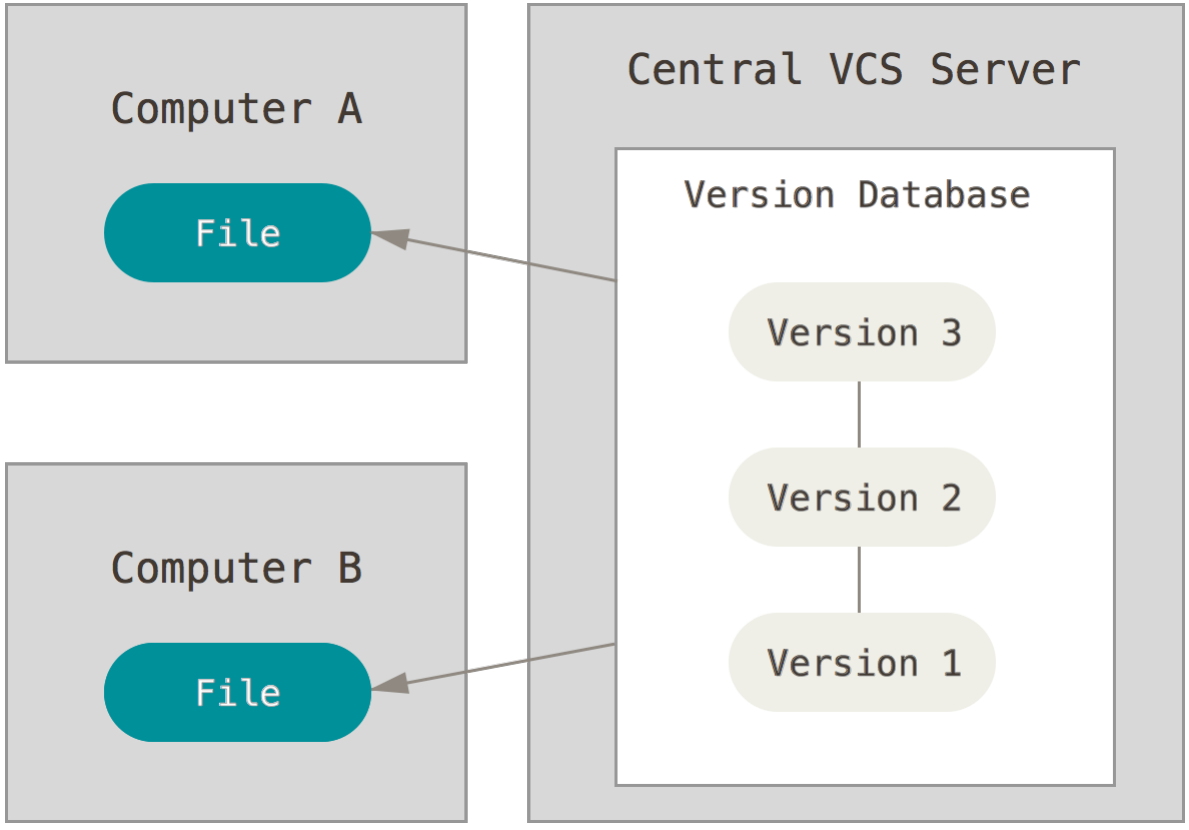
```
:~$ git status
```

- Zeigt den aktuellen Status an

Subversion

- Zentralisiertes System
- Eine Versionsnummer für das ganze Repository
- Dokumentation: `http://svnbook.org`
- Man arbeitet NIE im Repository

Übersicht



https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

Subversion: Befehle

- Größter Unterschied: git ist verteilt, svn ist zentralisiert
- Viele Ähnliche Befehle wie git, aber mit anderer Bedeutung
  - `svn checkout` ist ungefähr `git clone`
  - `svn commit` kombiniert `git add`, `git commit` und `git push`

Übung 6