

Praktikum angewandte Systemsoftwaretechnik

Aufgabe 3

Moritz Strübe, Alexander Würstlein, Rainer Müller, Gabor Drescher,
Timo Hönig

Lehrstuhl Informatik 4

November, 2012

Typische Aufgaben eines Versionskontrollsystems sind:

- Transportmedium
- Sichern von alten Zuständen
- Zusammenführung von parallelen Entwicklungen

Idealerweise zusätzlich:

- Unabhängige Entwicklung ohne zentrale Infrastruktur

Versionierung

Typische Aufgaben eines Versionskontrollsystems sind:

- Transportmedium
- Sichern von alten Zuständen
- Zusammenführung von parallelen Entwicklungen

Idealerweise zusätzlich:

- Unabhängige Entwicklung ohne zentrale Infrastruktur



Das Versionskontrollsystem des Linux Kernels

Git wurde 2005 von Linus Torvalds zur Unterstützung für die Linux Kernel Entwicklung geschrieben worden. Es sind viele Erfahrungen im Umgang mit großen Patchmengen und das Vorgängersystem *bitkeeper*. Es unterstützt:

- Dezentrale, parallele Entwicklung
- Koordinierung von Hunderten von Entwicklern
- Visualisierung von Entwicklungszweigen

Wichtige Git Kommandos zum Austauschen von Code

- Initiales *Klonen* der Quellen:

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6
```

- Einspielen von entfernten Änderungen:

```
git pull
```

- Mehrere Repositories registrieren:

```
git remote add 32-stable git://git.kernel.org/.../longterm/linux-2.6.32.y.git
```

- Registrierte Repositories auflisten:

```
git remote -v
```

Wichtige Git Kommandos zum Austauschen von Code (Forts.)

- Alle Remotes nachladen (aktueller Branch wird nicht verändert):

```
git remote update oder git fetch all
```

- Lokalen Branch aus dem neuem "Remote" anlegen:

```
git checkout -b work 32-stable/master
```

- Alle registrierten Zweige anzeigen:

```
git branch -a
```

- Unterschiede zwischen lokalem und entferntem Branch untersuchen:

```
git log ..origin/master
```

Wichtige Git Kommandos zum Austauschen von Code (Forts.)

- Aktuelle Änderungen auf dem entfernten Branch neu aufspielen:

```
git pull --rebase
```

- Die neuste Änderung untersuchen:

```
git show
```

- Den aktuellen Zustand des Repositories anzeigen:

```
git status
```

- Die letzten 2 Änderungen als Patch formatieren:

```
git format-patch HEAD~2
```

 wählt bei einem Merge den **ersten** "Elter"

```
git format-patch HEAD^^
```

 wählt bei einem Merge den **zweiten** "Elter"

```
git format-patch HEAD~4^^
```

 es ist möglich dies zu kombinieren

Wichtige Git Kommandos zum Austauschen von Code (Forts.)

- Sendeziel für Patchversand via E-Mail vorgeben:

```
git config sendemail.to=linux-kernel@i4.informatik.uni-erlangen.de
```

- Patchset mit den letzten 3 Änderungen via E-Mail senden:

```
git send-email --compose HEAD~3
```

- Sonstige Werkzeuge:

- `gitg`, `gitk`, `tig`
- `git gui`
- `git-meld` (globales `git difftool`)
- https://git.wiki.kernel.org/index.php/Interfaces,_frontends,_and_tools

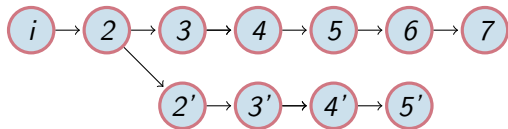
Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:



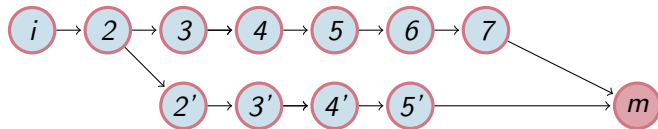
Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:



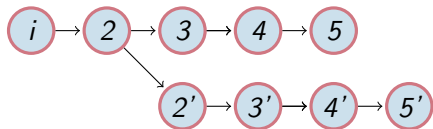
Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:

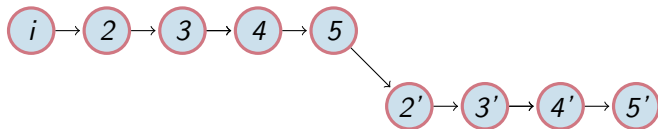


- Git versucht beide Änderungen zusammenzuführen
- Bei nicht eindeutigen Änderungen sind manuelle Eingriffe nötig

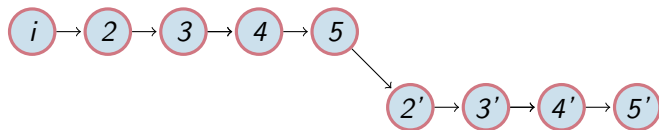
Aufsetzen auf bestehenden Zweigen (rebase)



Aufsetzen auf bestehenden Zweigen (rebase)



Aufsetzen auf bestehenden Zweigen (rebase)



- Patches aus dem “unterem” Zweig werden auf den “oberen” aufgespielt
- Die Historie ist nun linear
- Linearisierte Änderungen lassen sich häufig einfacher bewerten
- **Vorsicht!**
 - Verzweigungen vom alten Zweig können nun nicht mehr zusammengeführt werden
 - Keine gemeinsamen Vorgänger mehr
 - Visualisierung der Historie ist nun bestenfalls verwirrend