Praktikum angewandte Systemsoftwaretechnik

Aufgabe 3

Benjamin Oechslein, Daniel Lohmann, Jens Schedel, Michael Gernoth,
Moritz Strübe, Reinhard Tartler, Timo Hönig

Lehrstuhl Informatik 4

Mai, 2011

Zusammenfassung "Mini-Aufgabe" Distributions-Kernel

⇒ Unterschiedliche Herangehensweise der Distributionen
- Auswahl der Patches basierend auf einem Regelwerk (*policy*)
- Spannweite von konservativ (Gentoo) bis progressiv (SUSE, Ubuntu)

- Anzahl der unterschiedlichen Kernel. Abhängig von:
  - Menge der unterstützten Architekturen
  - Angebotene Geschmacksrichtungen (*flavors*), z.B. Desktop, Server

- Abweichung von Mainline aus Sicht der Anwender
  - Vorteil: Portierung wichtiger Änderungen (Sicherheit, Treiber)
  - Nachteil: Patches, die auf Mainline-Kernel anwendbar sind, brechen

Zusammenfassung Distributions-Kernel

- Abweichung von Mainline aus Sicht der Distribution
  - Vorteil: Optimale Anpassung des Kernel auf die Distribution
  - Nachteil: Immenser Aufwand für Wartung der Patches (*maintenance*)
- Distributionen bewerten Lizenzproblematik sehr unterschiedlich
  - Lizenzkritischer Programmcode wird nicht von Mainline akzeptiert
  - Grauzone: Laden nicht-offener Kernel-Module (z.B. Nvidia) $\rightarrow$ „*tainted*"
- Upstream-Involvierung der Distributionen
  - Hohes Gefälle: von Ignoranz bis aktive Teilnahme
  - Upstream-Engagement spart mehr Ressourcen als es kostet

Versionierung

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 Remotes untersuchen:

```
git remote -v
```

Wichtige Git Kommandos zum Austauschen von Code (Forts.)

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

```
git remote update
```

- 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
```

- Aktuelle Änderungen auf dem entfernten Branch neu aufspielen:

```
git pull --rebase
```

Wichtige Git Kommandos zum Austauschen von Code (Forts.)

- Die neuste Änderung untersuchen:

```
git show
```

- Die letzten **3** Änderungen als Patch formatieren:

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

- 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:

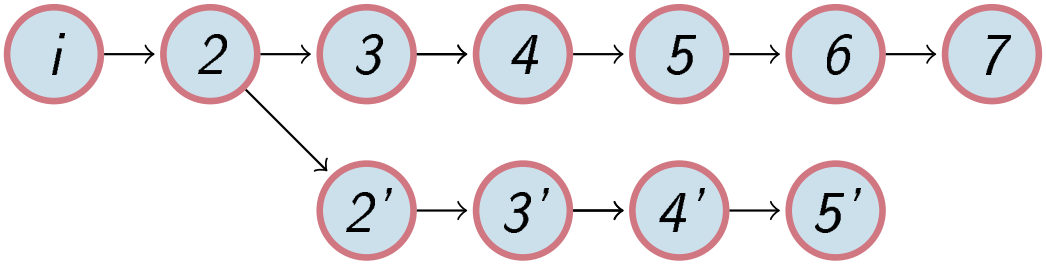
```
git send-email --compose HEAD~~
```

Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:

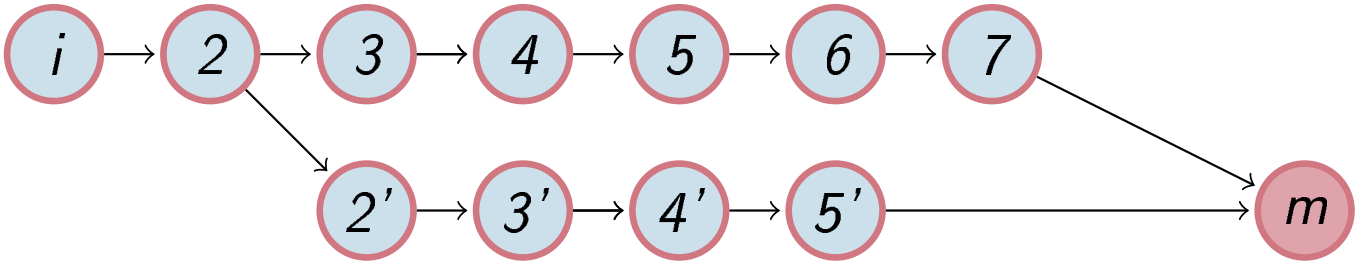
Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:



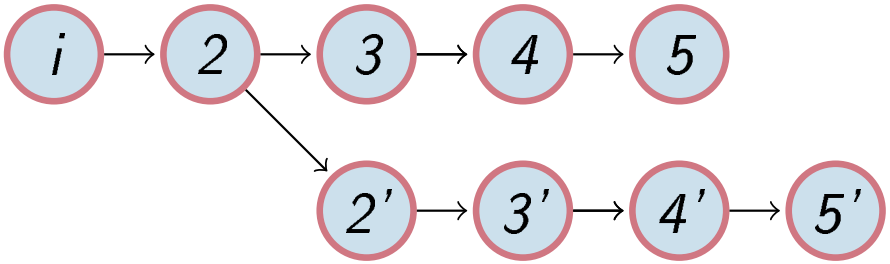
Verzweigungen und Zusammenführungen

Beispiel für parallele Entwicklung:

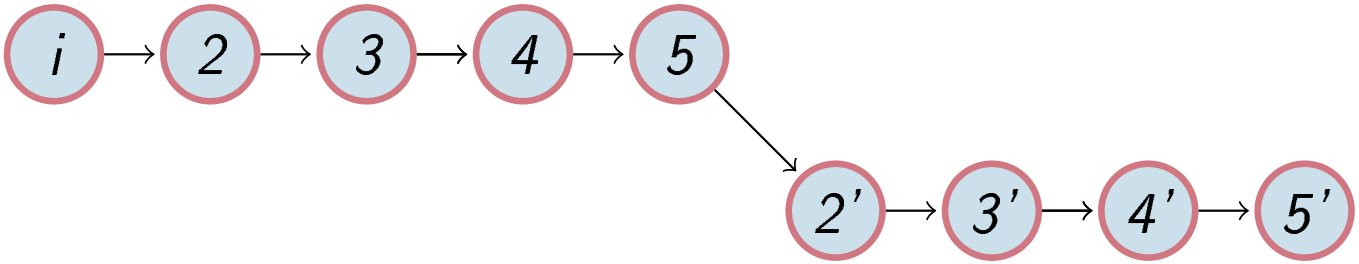


- Git versucht beide Änderungen zusammenzuführen
- Bei nicht eindeutigen Änderungen sind manuelle Eingriffe nötig

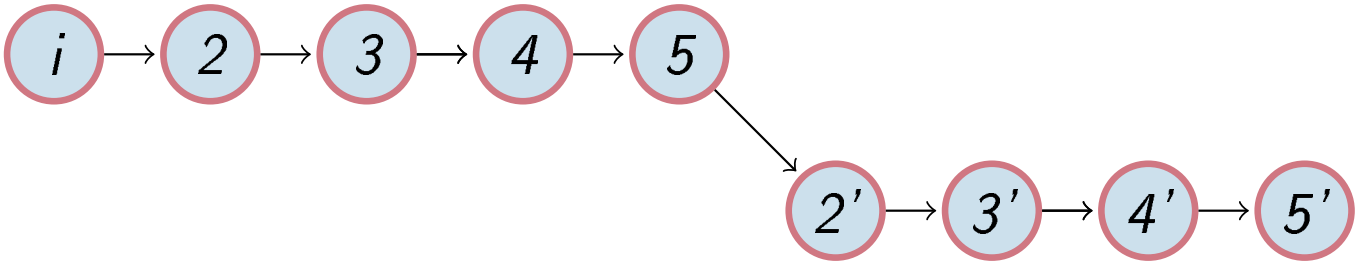
Aufsetzen auf bestehenden Zweigen (rebase)



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