Git – Eine Einführung

LinuxTag 2013, Berlin

Julius Plenz

22. Mai 2013

Ablauf

- Versionskontrolle: Zentral vs. Dezentral
- Historischer Kurzabriss zu Git

- Das Objektmodell – wie *funktioniert* Git?
- Merge vs. Rebase
- Koordinationsmöglichkeiten der Entwickler

- Wie lerne ich Git?

Vor- und Nachteile verteilter Versionskontrollsysteme

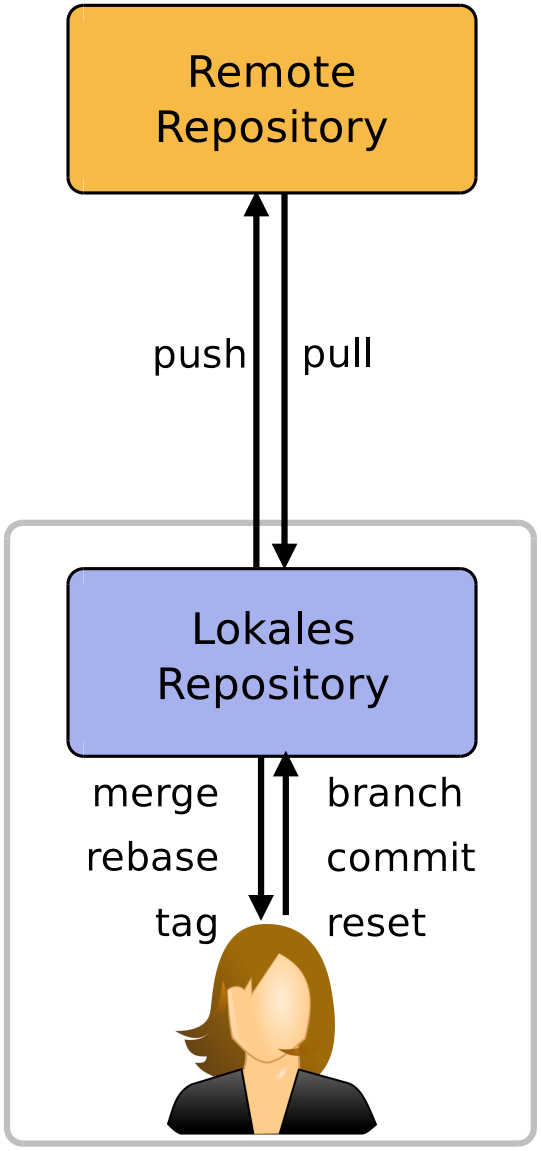
Vorteile:

- Jeder Entwickler besitzt eine *komplette* Kopie der Versionsgeschichte
  - Offline-Arbeit möglich
  - Impliziter Schutz vor Manipulation
- Es gibt keinen »single point of failure«
  - Serverausfall, Hack, wütender Entwickler, ...
- Kein Streit um Commit-Rechte
- Delegation von Aufgaben ist leichter

Nachteile:

- Viel Freiheit: Policies müssen geschaffen werden
- Komplexeres Setup als zentralisierte Systeme

Autonomie des eigenen Repositories



- *Remote* und lokales Repository sind gleichberechtigt
- Austausch zwischen Repositories via Push/Pull
  - *Push*: Eigene Änderungen hochladen
  - *Pull*: Änderungen herunterladen
- Alle anderen Aktionen passieren zunächst *nur* lokal

Die Geschichte von Git

- 2005: Der Linux-Kernel darf *BitKeeper* nicht mehr kostenfrei verwenden
- 3. April: Linus Torvalds beginnt, ein eigenes VCS zu schreiben: *Git*
- 7. April: Git verwaltet nun seinen eigenen Quellcode
- 16. Juni: Das erste Kernel-Release wird mit Git organisiert
- Februar 2007: Git 1.5.0
- Februar 2010: Git 1.7.0
- Oktober 2012: Git 1.8.0

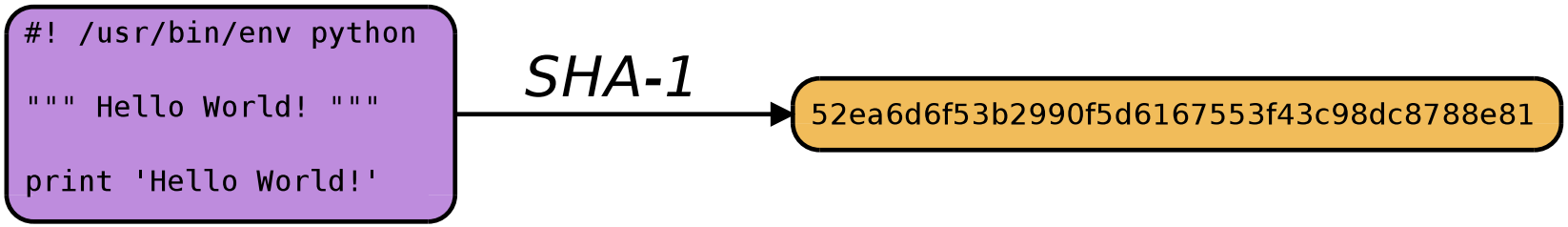
Wesentliche Merkmale von Git

- Die Architektur von Git ist ausgelegt auf:
  - Schnelligkeit
  - Integrität
  - Simplizität
- Git ist ein typisches Unix-Tool
  - »Alles ist Text«
  - Viele kleine Subkommandos
  - GUIs sind meist nur ein *Überbau*

Das Objektmodell

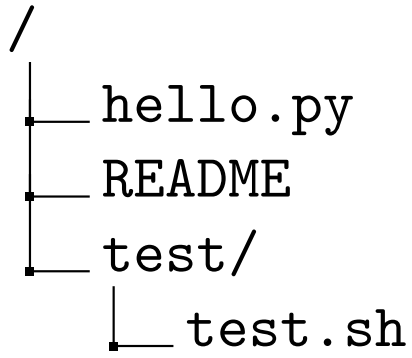
SHA-1-Summe

- *SHA-1* ist eine weit verbreitete *Hash-Funktion*
  - Eingabe: Bit-Sequenz mit max. Länge $2^{64} - 1$ ($\approx$2 Exbibyte)
  - Ausgabe: Hexadezimal-Zahl der Länge 40 (d. h. 160 Bits)
  - Resultat ist eine von 2^{160} ($\approx 1.5 \cdot 10^{49}$) möglichen Zahlen
  - Ziemlich einzigartig ($\rightarrow$ *Kollisionssicherheit*)



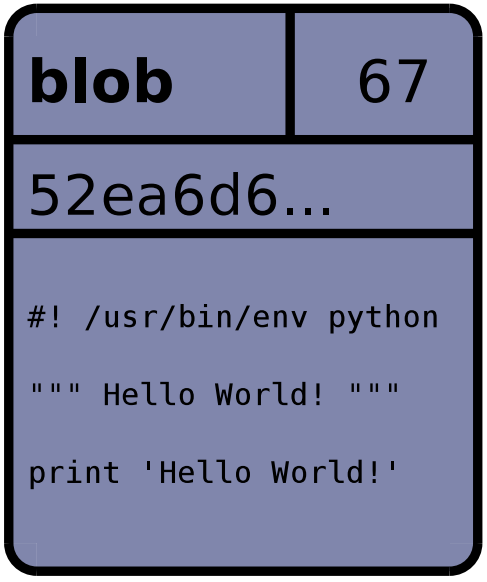
Inhalt eines Repositories

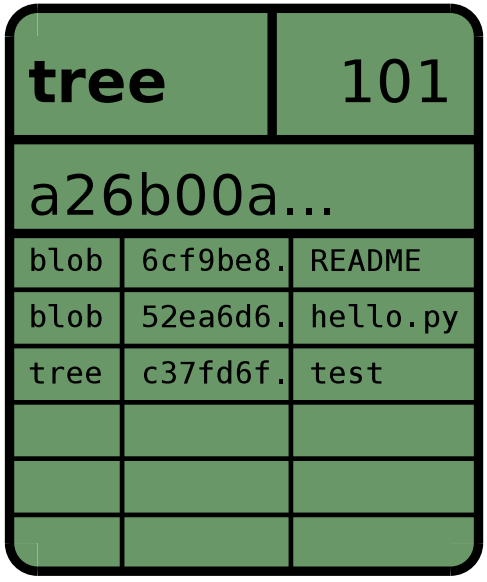
Angenommen, wir wollen folgendes Verzeichnis speichern:



Git-Objekte

- *Blob*: Enthält den Inhalt einer Datei
- *Tree*: Eine Sammlung von Tree- und Blob-Objekten
- *Commit*: Besteht aus einer Referenz auf einen Tree mit zusätzlichen Informationen
  - *Author* und *Commiter*
  - *Parents*
  - *Commit-Message*





Objektverwaltung

- Alle Objekte werden von Git in der *Objektdatenbank* (genannt Repository) gespeichert
- Ein Objekt wird nach der SHA-1-Summe seines *Inhaltes* benannt
- Dadurch sind Objekte durch ihre SHA-1-ID *eindeutig* adressierbar

- Für jede Datei erzeugt Git ein Blob-Objekt
- Für jedes Verzeichnis erzeugt Git ein Tree-Objekt
- Ein Tree-Objekt enthält die Referenzen (SHA-1-IDs) auf die in dem Verzeichnis enthaltenen Dateien

Objektdatenbank

```
$ git cat-file commit e2c67ebb6d2db2aab831f477306baa44036af635
tree a26b00aaef1492c697fd2f5a0593663ce07006bf
parent 8e2f5f996373b900bd4e54c3aefc08ae44d0aac2
author Valentin Haenel <valentin.haenel@gmx.de> 1294515058 +0100
committer Valentin Haenel <valentin.haenel@gmx.de> 1294516312 +0100

Kommentar fehlte
```

```
$ git ls-tree a26b00aaef1492c697fd2f5a0593663ce07006bf
100644 blob 6cf9be8017a937ca9f442290bcc8b2db13f12ab4    README
100644 blob 52ea6d6f53b2990f5d6167553f43c98dc8788e81    hello.py
040000 tree c37fd6f7d4f9619448f0feafec09ef5d18b58712    test
```

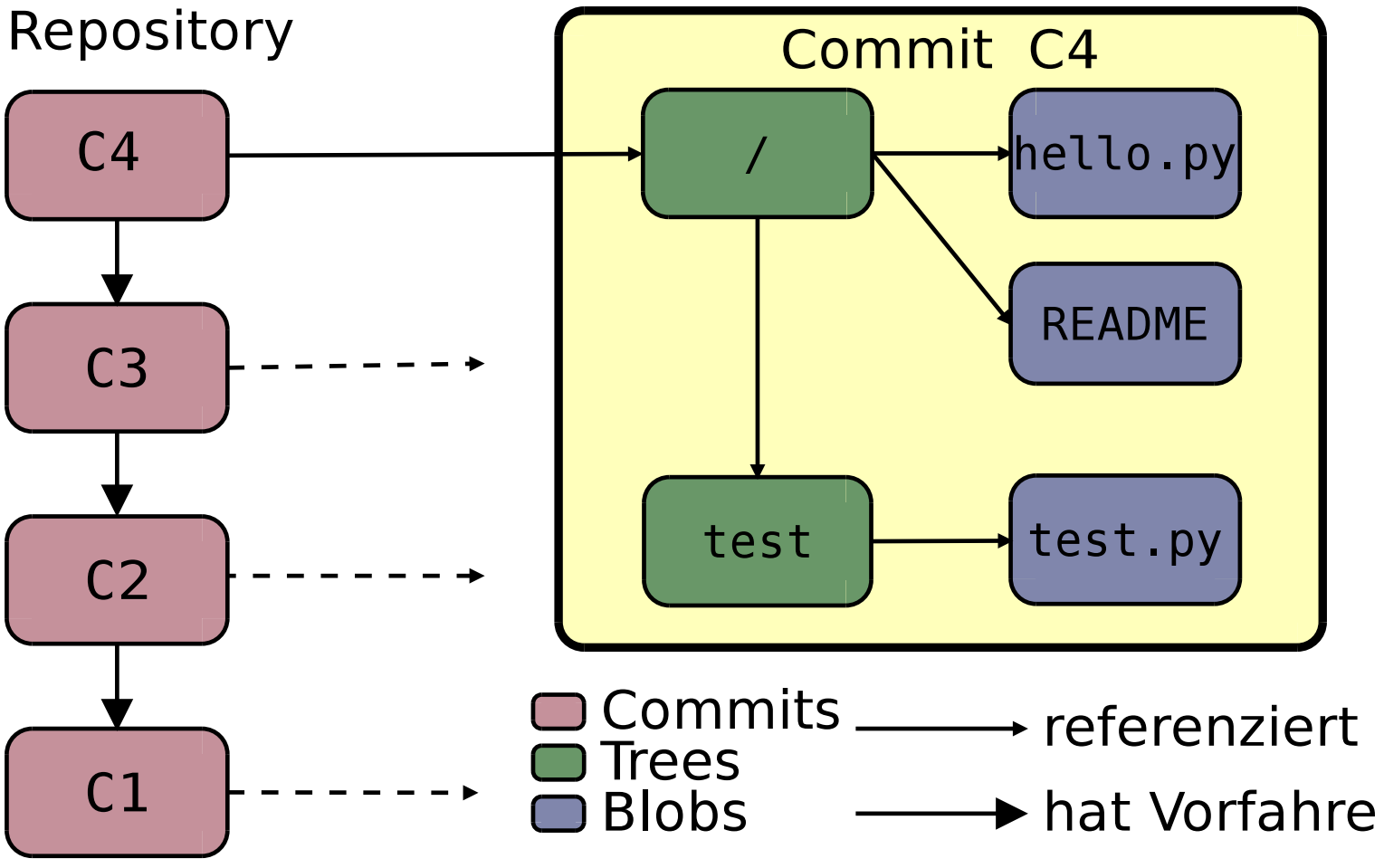
```
$ git cat-file blob 52ea6d6f53b2990f5d6167553f43c98dc8788e81
#!  /usr/bin/env python

""" Hello World!  """

print 'Hello World!'
```

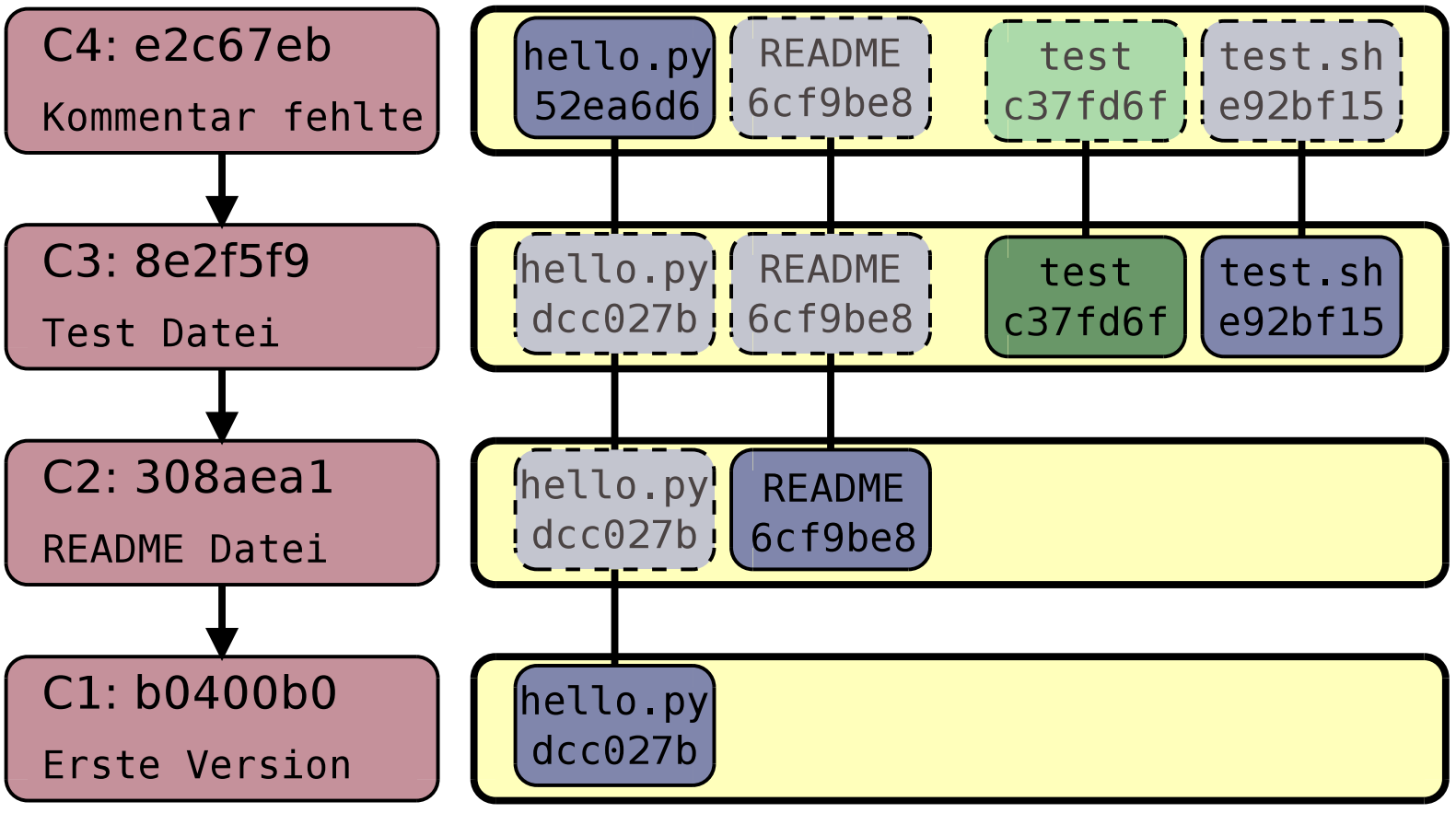
Zusammenfassung

Ein Git-Repository enthält Commits; diese wiederum referenzieren Trees und Blobs, sowie ihren direkten Vorgänger



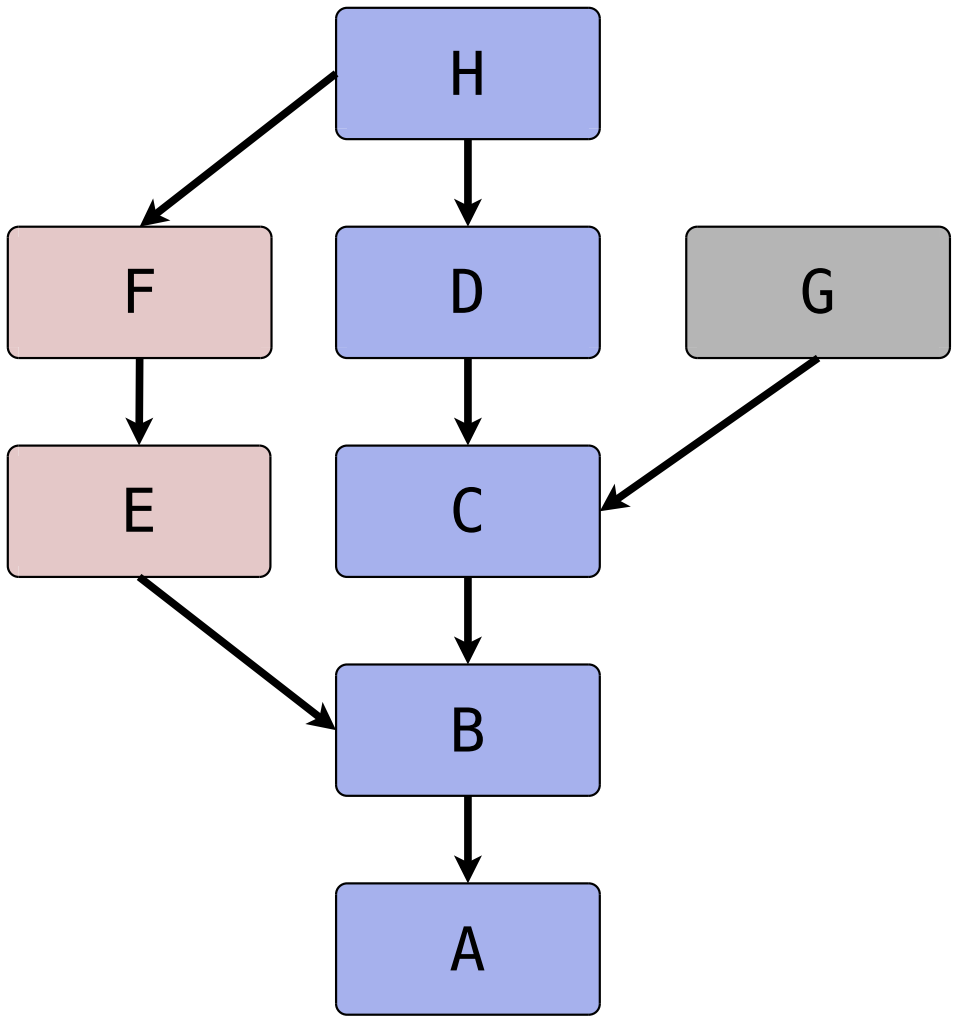
Commit = Dateibaum

Ein Commit hält den Zustand *jeder* Datei zum gegebenen Zeitpunkt fest. (Auch den der nicht geänderten.)



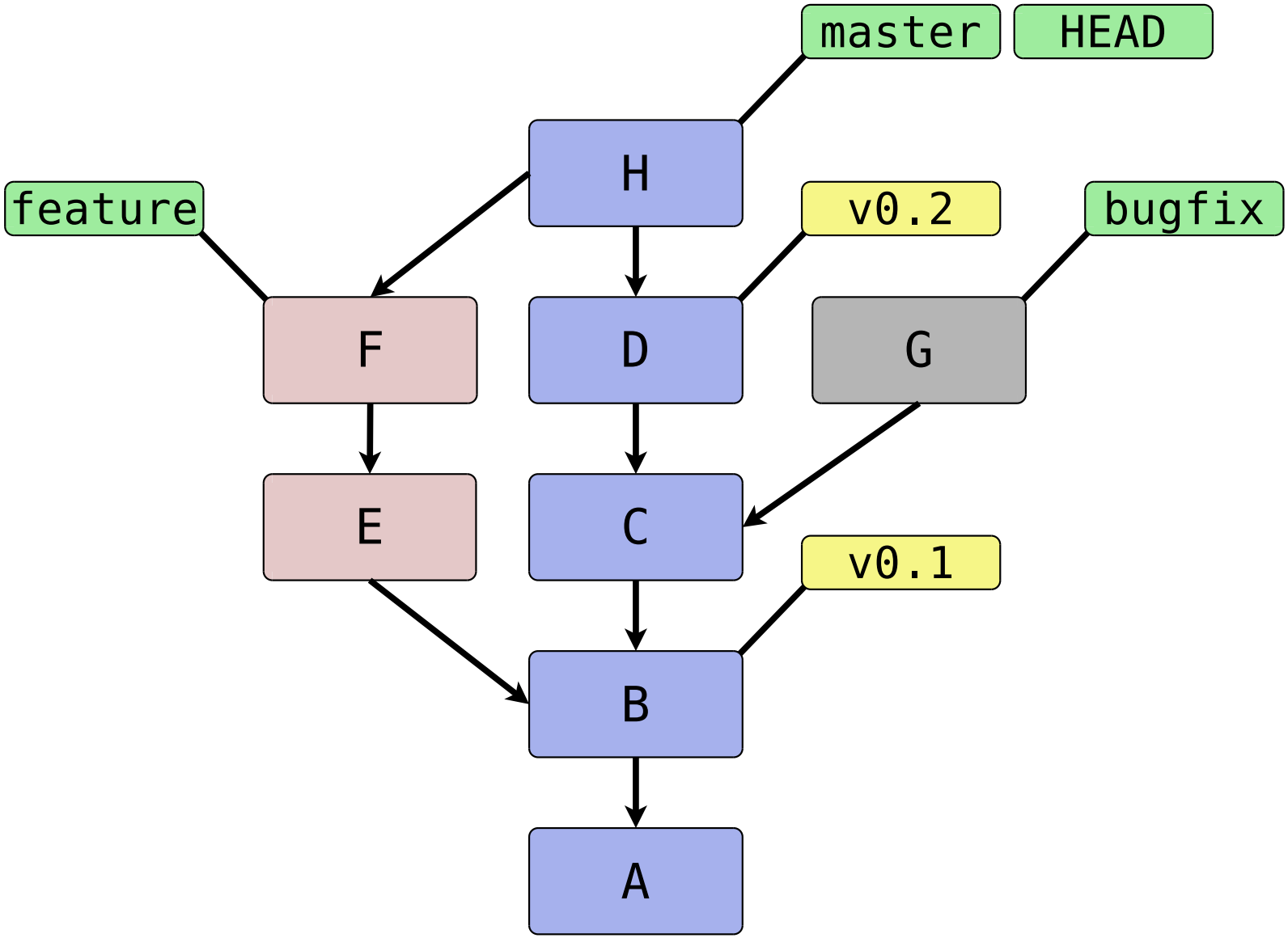
Commit Graph

Ein Repository ist ein *Gerichteter Azyklischer Graph*
Engl.: Directed Acyclic Graph (DAG)



Branches und Tags

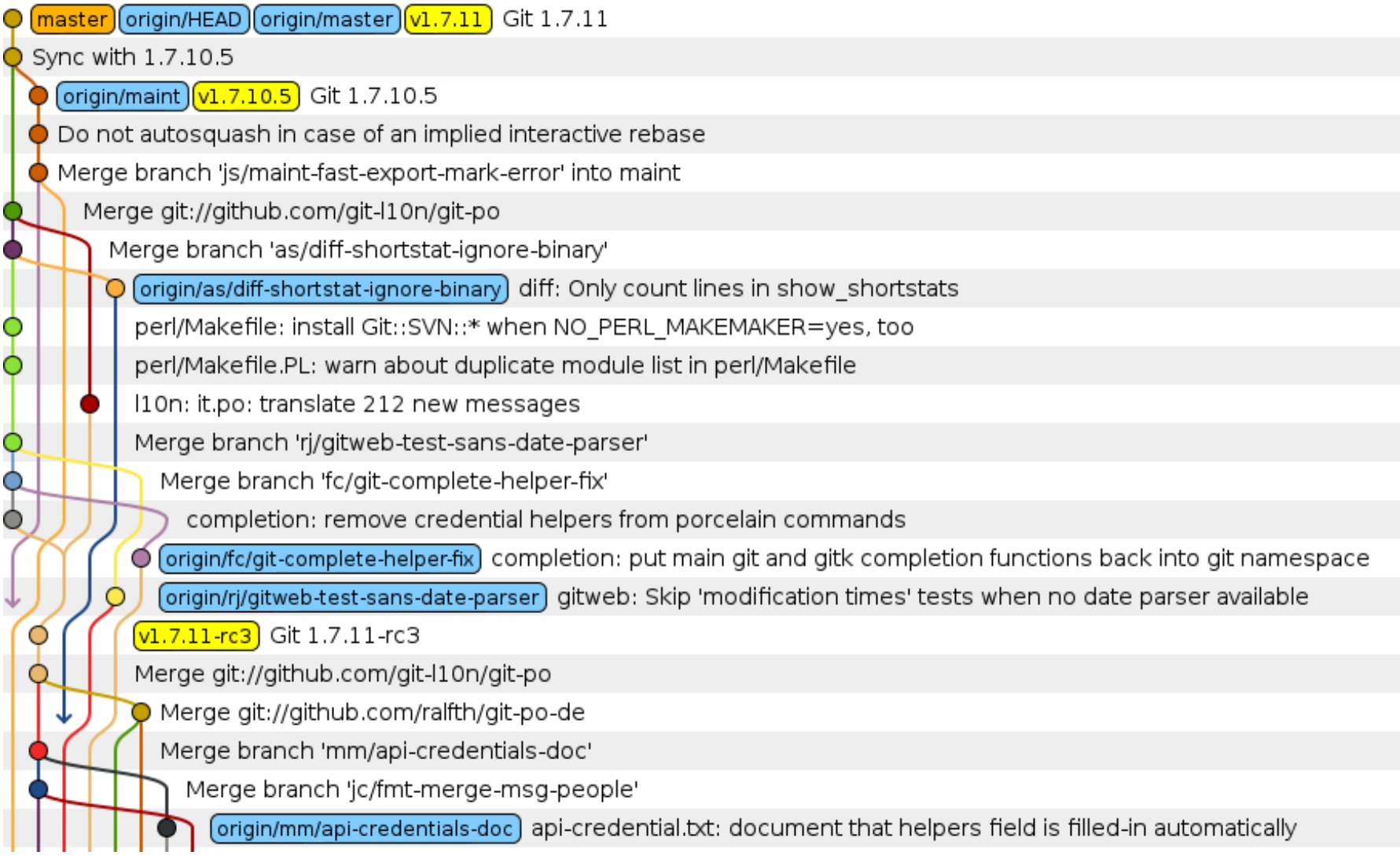
Branches und Tags sind Zeiger auf Knoten in dem Graphen.



Graph-Struktur

- Die gerichtete Graph-Struktur entsteht, da in jedem Commit Referenzen auf direkte Vorfahren gespeichert sind
- Integrität kryptographisch gesichert
- Git-Kommandos manipulieren die Graph-Struktur

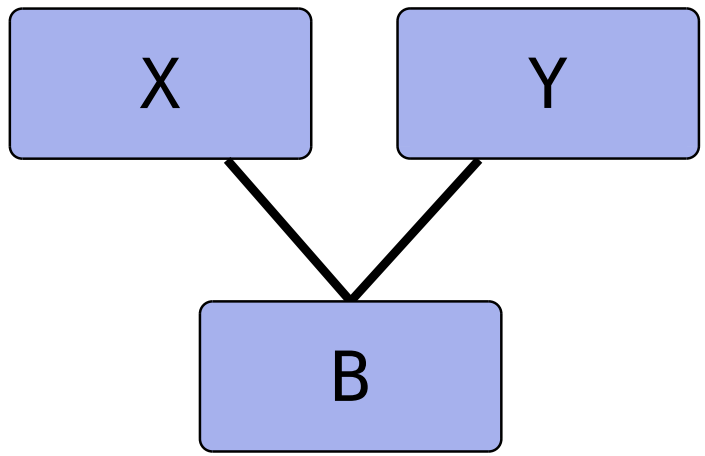
Visualisierung des Git-Graphen



Projekt-Organisation

Parallele Änderungen

- Branches entstehen in Git auf ganz »natürliche« Weise
  - Commit X hat Vorgänger B
  - Commit Y hat Vorgänger B
- Git bietet zwei Möglichkeiten, Änderungen zu integrieren
  - Merge
  - Rebase

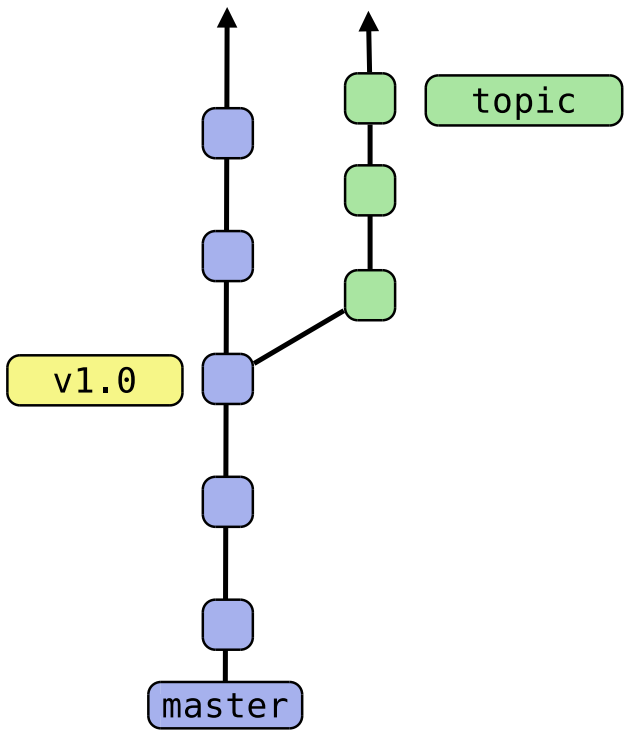


Merges?

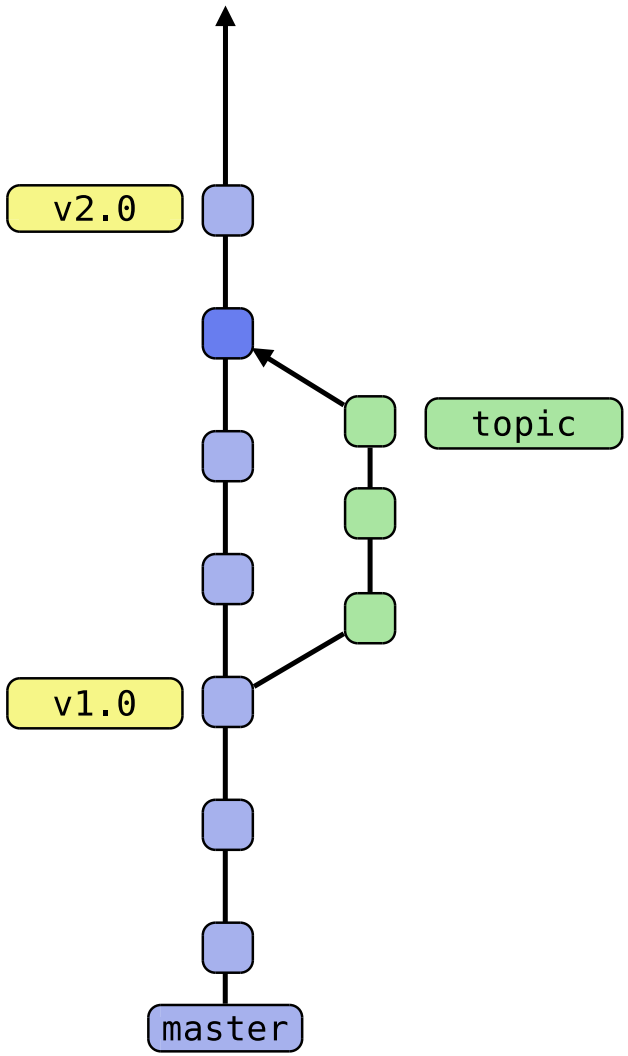
> Merging changes sounds simple enough, but in practice it can become a headache.
>
> — Book *Version Control with Subversion*, O'Reilly

> I just think a lot of people merge without even thinking about all the other things it involves, just because git made it *so* easy to do.
>
> — Linus Torvalds, LKML, May 2008

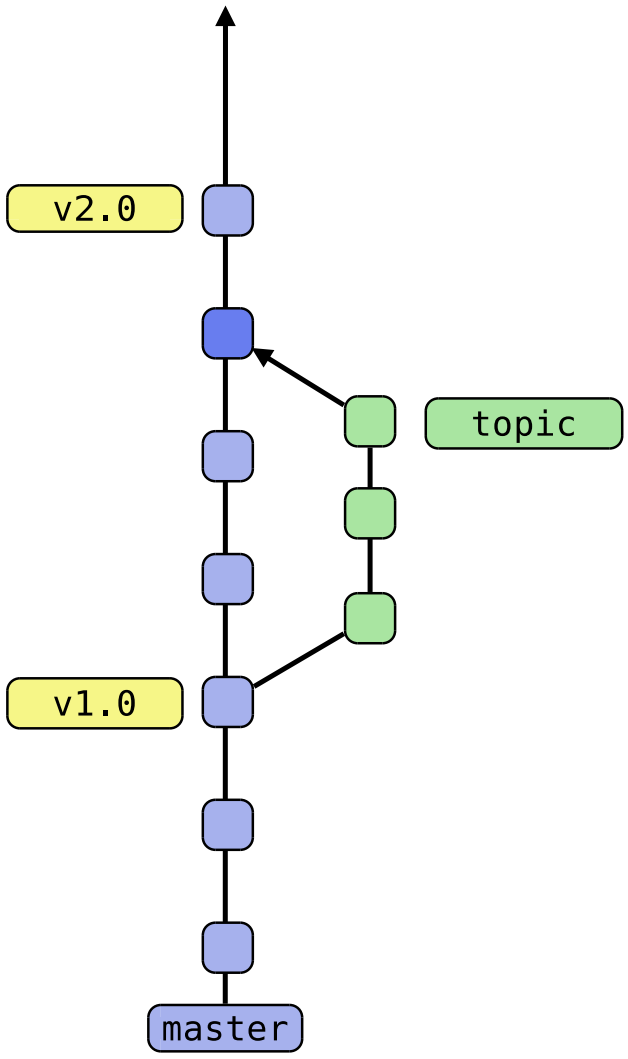
Merge



Merge

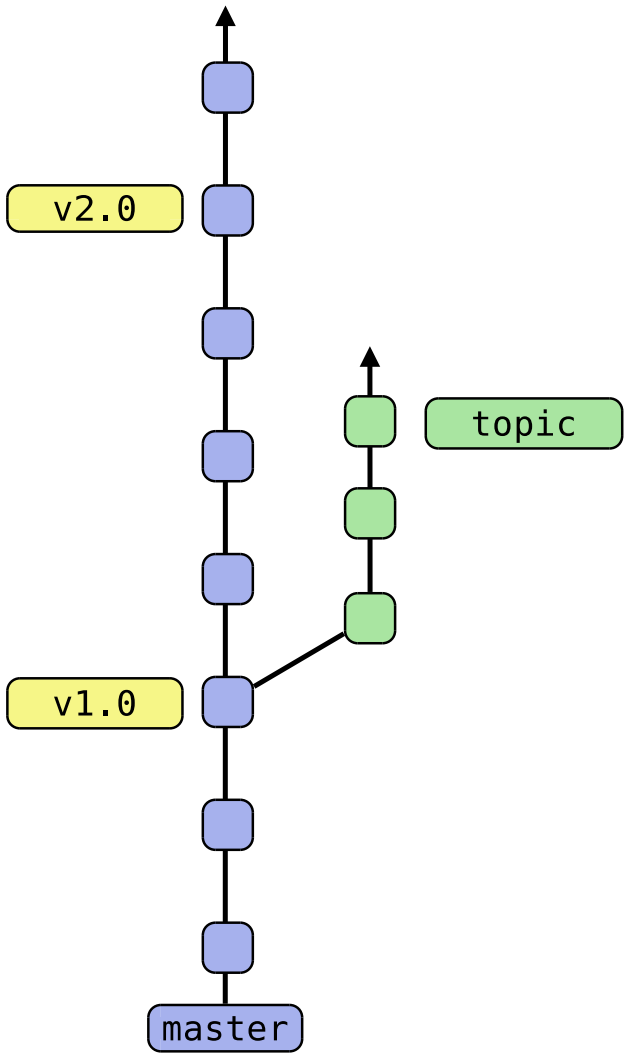


Merge

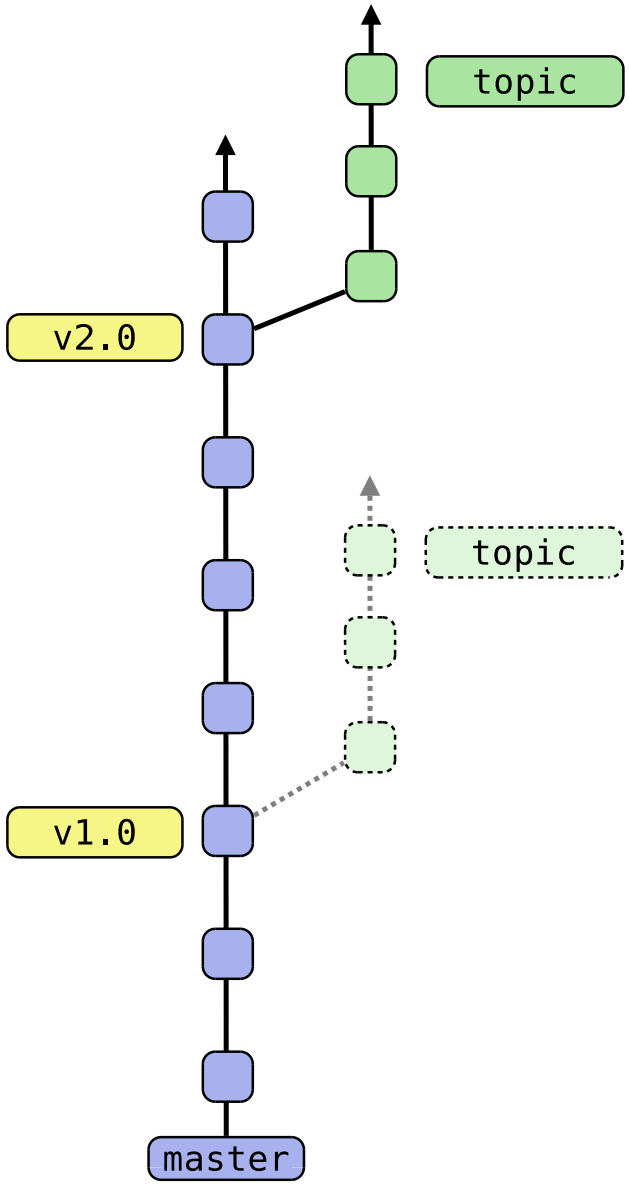


- Der *Merge-Commit* hat mehrere Vorgänger
- Die Änderungen beider Branches sind in `v2.0` enthalten
- Zeitpunkt der Integration des Features wird deutlich
- Die Branches können nach wie vor unterschieden werden

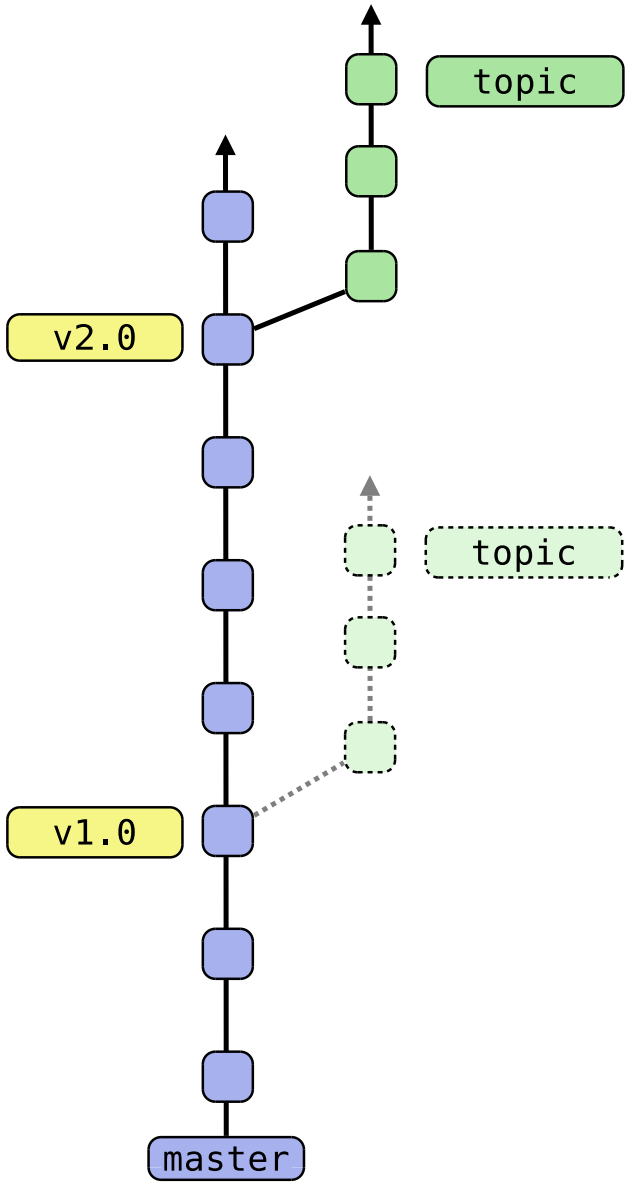
Rebase



Rebase



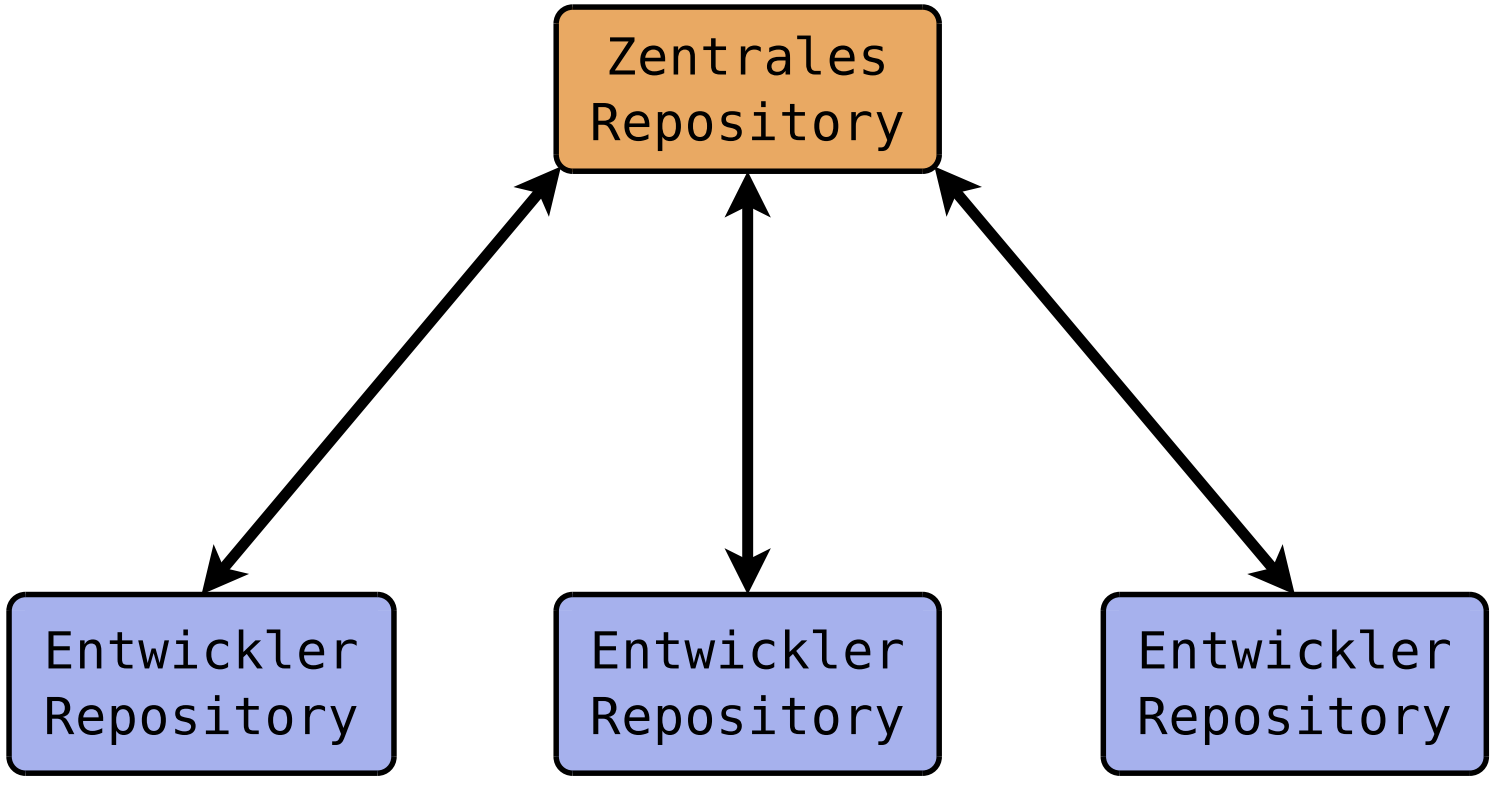
Rebase



- Commits aus `topic` werden auf eine *neue Basis* aufgebaut
- Ihre Vorgänger enthalten nun schon die Änderungen in `v2.0`
- Ideal für noch unfertige Entwicklungen
- Vorwärts- oder Rückwärtsportierung von Änderungen

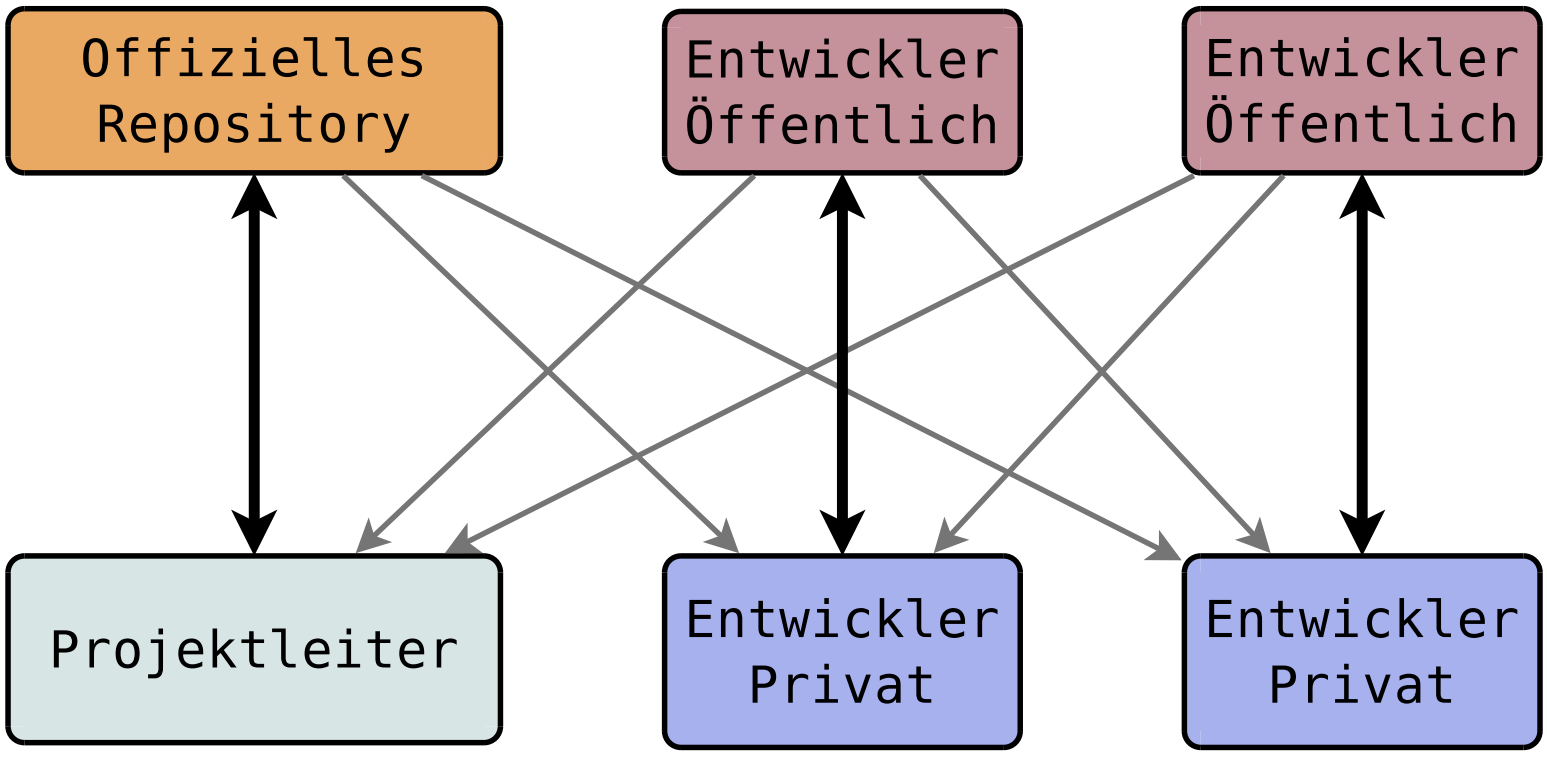
Zusammenarbeit der Entwickler

Zentralisiert



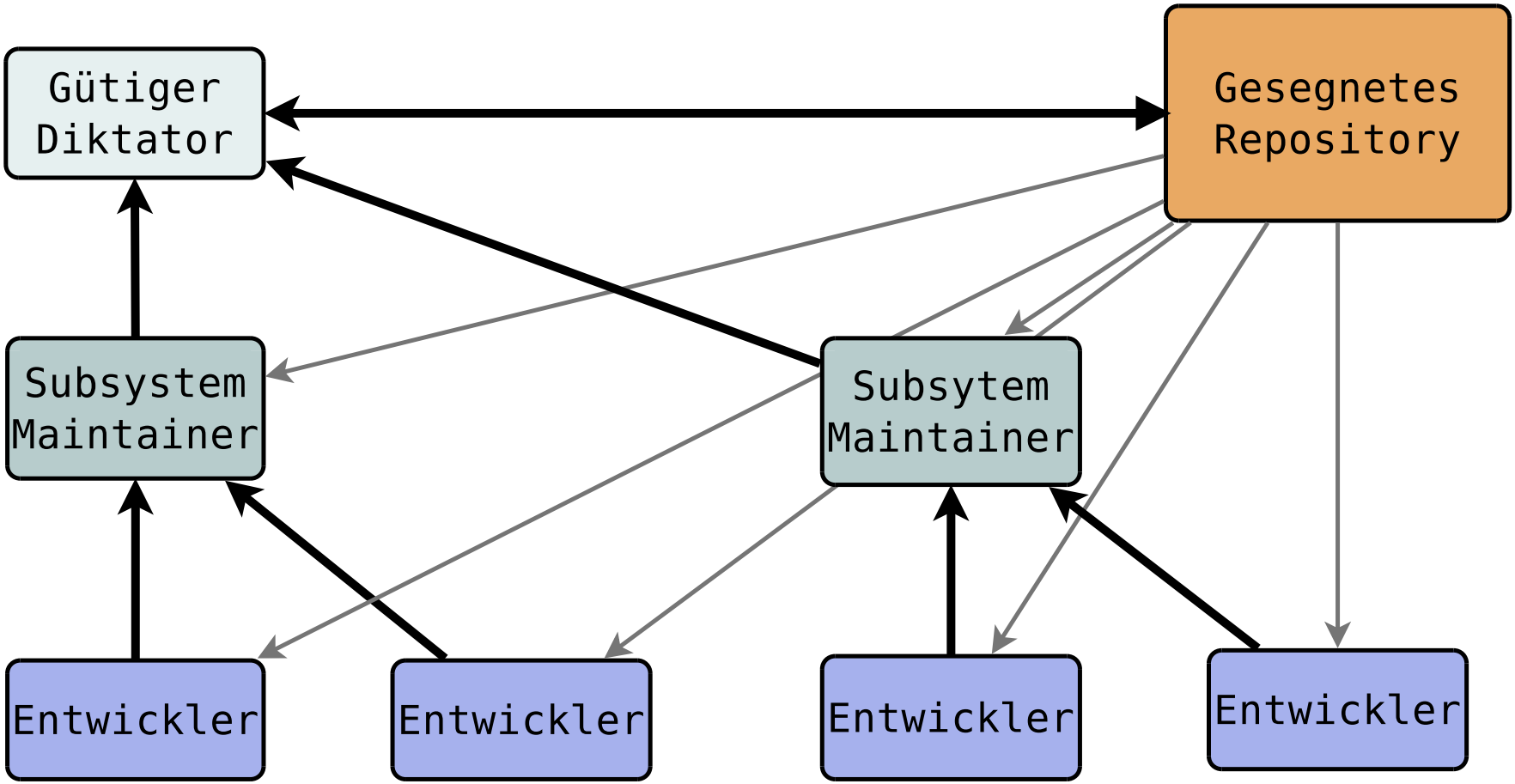
- Ein einziges zentrales Repository
- Alle Entwickler haben Schreibzugriff
- Aber: Commit $\neq$ Synchonisation

Öffentliche Entwickler-Repositories



- Ein öffentliches Repository pro Entwickler
- Der Projektleiter integriert Verbesserungen

Patch-Queue per Email



- Mehrere Hierarchie-Ebenen
- Stark vom Linux-Kernel und Git selbst verwendet

Ausblick

- Der Git-Index (*Staging Area*)
- Interaktives Rebase (»Code aufräumen«)
- Automatisierte Fehlersuche (`git bisect`)

Git lernen

Das Git-Buch

Git – Verteilte Versionskontrolle für Code und Dokumente, 328 Seiten, Open Source Press, 2011

Inhaltsverzeichnis und Leseprobe unter `http://gitbu.ch/`

Insgesamt ist es den Autoren gelungen, ein übersichtliches Git-Kompendium zusammenzustellen, das alle wichtigen Anwendungsfälle abdeckt.

— Rezension *t3n*-Magazin, 2011/09

»Try Git«

- `http://try.github.io/`
- »Git Magic«-Tutorial
- »Pro Git« von Scott Chacon

- Aber am besten: Einfach anfangen, ein kleines Projekt in Git verwalten. Und sei es nur ein Shell-Script.

Danke!

Fragen?