Benjamin Kramer David Losch

Versionskontrolle & git

2. April 2014

Versionskontrolle

```
$ ls mein_projekt
List.java
List.bak
List.vorBugfix
List.bak2
Listx.java
List.original
List2.java
...
```

Moderne Variante:
Dropbox

Versionskontrolle: RCS

```
$ ls mein_projekt
List.java,v
$ co -l List.java
List.java,v  -->  List.java
$ vim List.java
$ ci List.java
List.java,v  <--  List.java
new revision: 1.2; previous revision: 1.1
enter log message
>> Doppelte einfuegungen korrigiert.
```

Versionskontrolle: RCS

```
$ cat List.java,v
head   1.2;
…
1.2
date   2014.04.01.13.37.42;    author Benjamin; state Exp;
…
1.2
@Doppelte einfuegungen korrigiert.
…
text
@class List {
```

Versionskontrolle: RCS

- Bei mehreren Nutzern benutzt RCS zur Konfliktlösung Locks

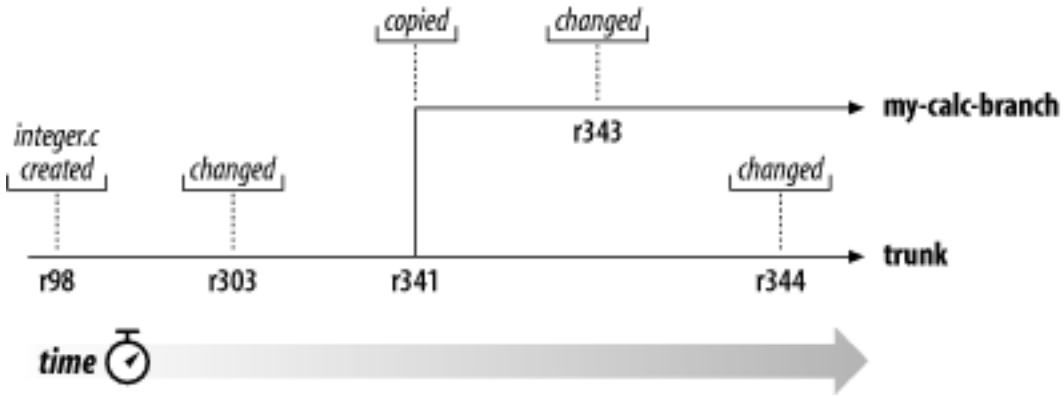
```
co: List.java,v: Revision 1.2 is already locked by
Benjamin
```

- Skaliert nicht.

Versionskontrolle: CVS

- **Jetzt**: Zentraler Server
- Repository (aber immer noch Logs für jede einzelne Datei)
- Konflikte werden automatisch aufgelöst (man Versucht es zumindest)
- **Neu**: Branches und Tags

Branches

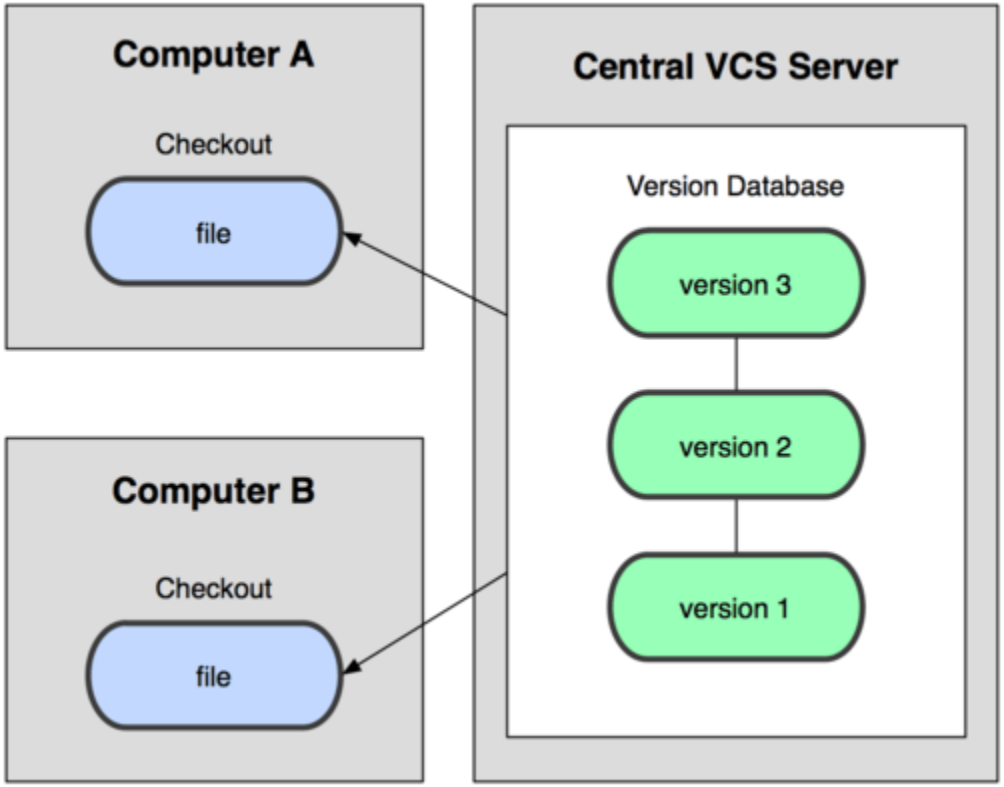


Quelle: http://svnbook.red-bean.com/

Versionskontrolle: Subversion

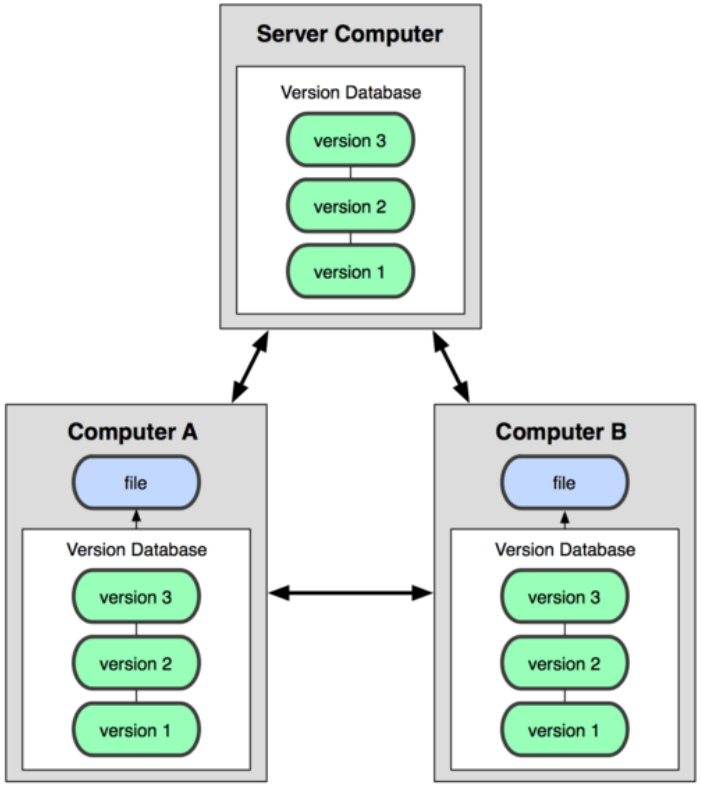
- “CVS done right”
- Jetzt nicht mehr einzelne Logs für Dateien sondern globale Commits

Versionskontrolle: Subversion



Quelle: Scott Chacon - Pro git

Versionskontrolle: git



Quelle: Scott Chacon - Pro git

Versionskontrolle: git

- Ursprünglich von Linus Torvalds für den Linux-Kernel entwickelt
- Entstanden 2005 aus dem Bitkeeper-Fiasko
- Aus dem gleichen Grund entstand auch Mercurial (`hg`)

git installieren

- Linux: verfügbar in allen gängigen Paketquellen, z.B.: `sudo apt-get install git`
- Windows: TortoiseGit

Repository anlegen

```
$ git init
```

- Ausführbar in existierendem Projektordner

```
$ git init --bare
```

- Legt ein neues serverartiges Repository an
- Enthält nur die Inhalte des .git-Ordners, aber keine *working copy*
- Dient als zentrales Repository

Beispiel

```
$ git clone \
  https://name@projekte.itmc.tu-dortmund.de/in/pg583-git.
git
```

alternativ

```
$ git clone \
  git@projekte.itmc.tu-dortmund.de:in/pg583-git.git
```

```
$ git status
$ git add
$ git commit
$ git push
```

Beispiel 2

```
$ git branch experimental
$ git checkout experimental
…
$ git commit -a
$ git push
$ git checkout master
$ git merge experimental
$ git push
$ git branch -d experimental
```

Beispiel 3

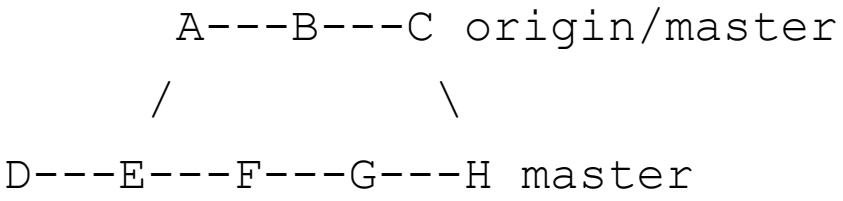
```
$ git remote add origin https://...
```

git clone

- Lädt das komplette Repository herunter
  - Inklusive der History
  - und aller referenzierten Branches
- Der Branch “master” wird als Arbeitskopie genommen
- Der Ursprungsort wird als “origin” eingetragen.
  - origin/master -> master

git pull

- Lädt die neusten Änderungen vom “origin”
- Führt dann `merge` oder `rebase` aus um die Änderungen zu integrieren
  - `git pull --merge` (default)
  - `git pull --rebase`



git add

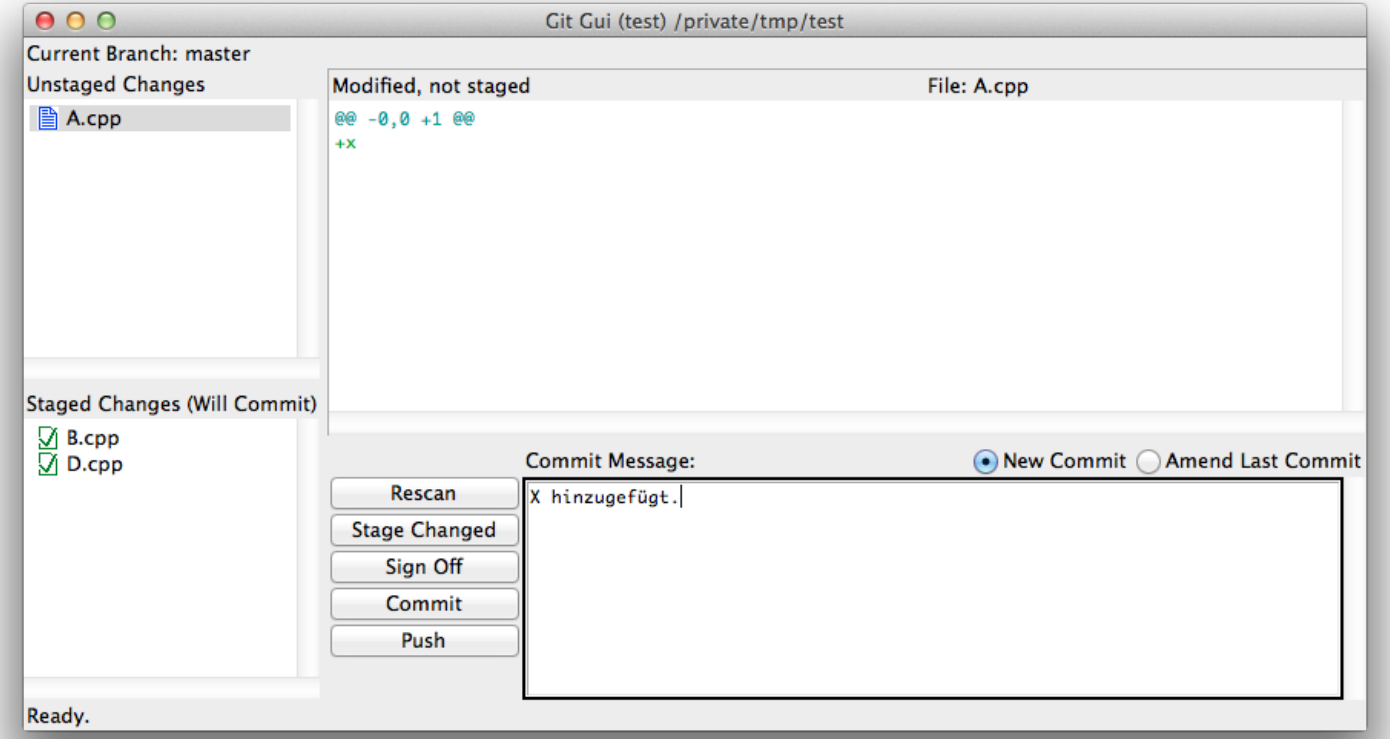
- Fügt eine Datei oder einen Teil einer Datei (Parameter -p) zum Index hinzu
- Der Entwickler wird nicht gezwungen, stets alles zu commiten, sondern kann alles genau entscheiden

```
$ git add -p code.cpp
diff --git a/code.cpp b/code.cpp
index f1bc8ea..84e18fa 100644
--- a/code.cpp
+++ b/code.cpp
@@ -2,11 +2,11 @@


 void doSomething()
 {
-
+       std::cout << "This code prints an interesting text";
 }
 int doSomethingElse()
 {
-       return 0;
+       return 1;
 }
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 2 hunks.
@@ -2,9 +2,9 @@


 void doSomething()
 {
-
+       std::cout << "This code prints an interesting text";
 }


 int doSomethingElse()
 {
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? n
@@ -6,7 +6,7 @@
 }


 int doSomethingElse()
 {
-       return 0;
+       return 1;
 }
Stage this hunk [y,n,q,a,d,/,K,g,e,?]? y
```

git commit

- Erzeugt einen neuen Eintrag im Log
- Wird nicht sofort hochgeladen
- Es werden nur Dateien übernommen, die vorher mit `git add` hinzugefügt wurden (index)
- Oder: `git commit -a`, dann wie SVN (alle veränderten Dateien, die git schon kennt)
- `git commit -a -m "Algorithmus 42% schneller gemacht"`

git commit (gui)



git commit --amend

- Ändert den letzten commit

```
$ git commit -m 'initial commit'
$ git add forgotten_file
$ git commit --amend
```

- Ohne Änderung seit letztem commit wird nur letzte commit message angezeigt und kann bearbeitet werden

git push

- Kopiert lokale commits nach “origin”
- Dabei wird Standardmäßig der aktuelle Branch genommen
- `git push origin mybranch`
  - Legt “mybranch” auf origin an und überträgt die Commits.

git: Änderungen zurücknehmen

- `git checkout Datei.cpp`
  - Setzt Datei.cpp auf den Stand des letzten Commits (oder Index)
- `git revert <revision>`
  - Legt einen neuen Commit an, der einen anderen Commit zurücknimmt
- `git reset --hard <revision>`
  - Setzt den kompletten Zustand (Dateien + History) auf einen älteren Stand zurück

git merge

- `git merge branch1 [branch2]`
- Versucht die angegebenen Branches zusammenzuführen und speichert das Ergebnis im aktuellen Branch
- Bei Konflikten muss der Benutzer diese eventuell per Hand auflösen (resolve)

Merge conflict

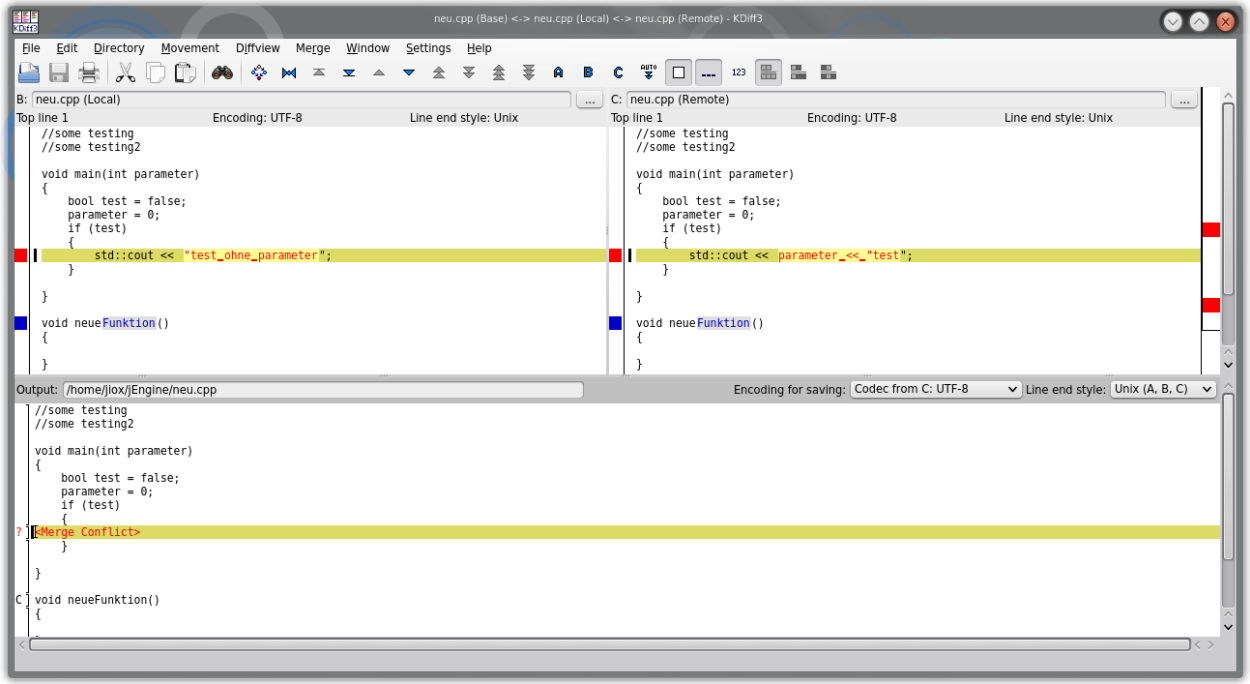
```
CONFLICT (content): Merge conflict in code.cpp
Automatic merge failed; fix conflicts and then commit the result.
```

code.cpp:

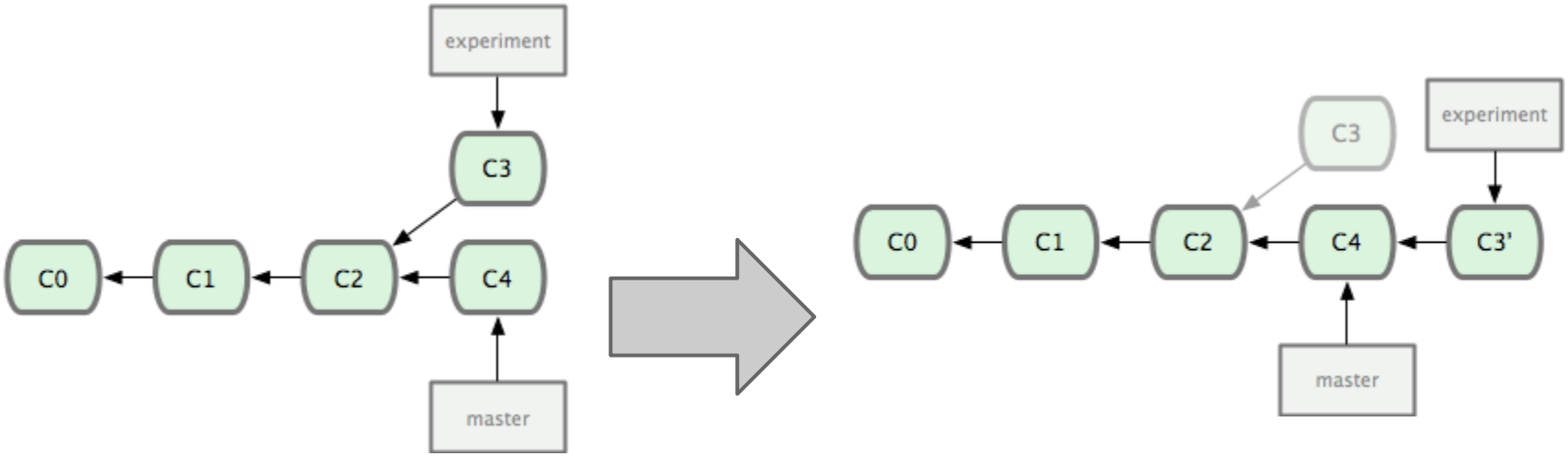
```
<<<<<<< HEAD
        std::cout << "a";
=======
        std::cout << "b";
>>>>>>> experimental
```

Merge tools

```
$ git mergetool code.cpp
```



git: rebase

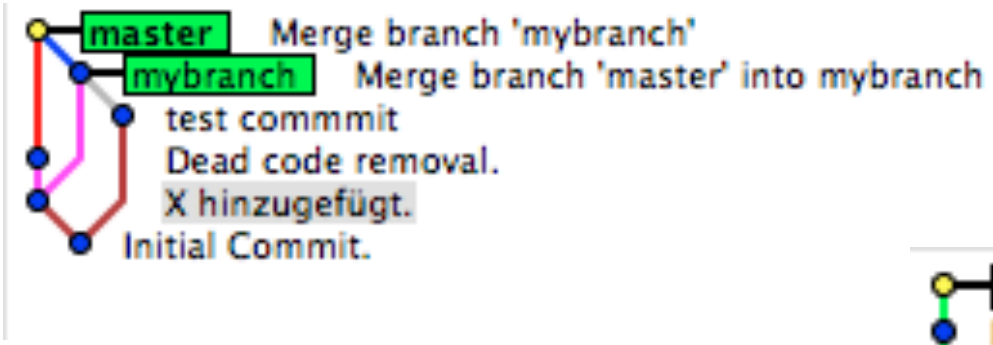


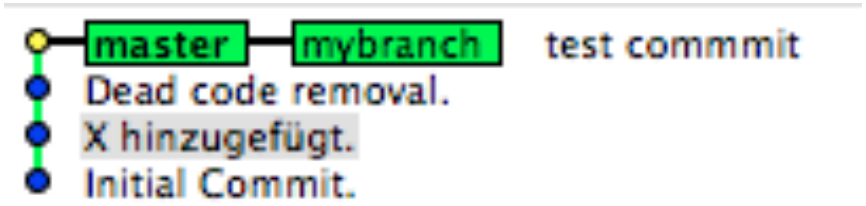
Achtung: Ändert die History

Nicht benutzen wenn C3 schon gepusht war

git: rebase

- Wozu, wenn es so gefährlich ist?
- Schönere History





Szenario

- Man entwickelt an einem Branch, aber ein anderer Mitarbeiter fordert ein gewisses Feature im gleichen Branch an, das er so schnell wie möglich benötigt
- Dazu bietet git ein Werkzeug: *stash* speichert den aktuellen Zustand in einen Zwischenspeicher und setzt den Stand des aktuellen Verzeichnisses wieder auf HEAD zurück

git stash

```
$ git stash
Saved working directory and index state WIP on master:
2ee70f4 Commit message
HEAD is now at 2ee70f4 Commit message
$ git stash list
stash@{0}: WIP on master: 2ee70f4 Commit message
…
$ git commit -a; git push
$ git stash apply stash@{0}
```

bzw. `$ git stash drop stash@{0}`

Dateien ignorieren

.gitignore-Datei:

```
$ cat .gitignore

*~
*.exe
tmp/*
bin/*
```

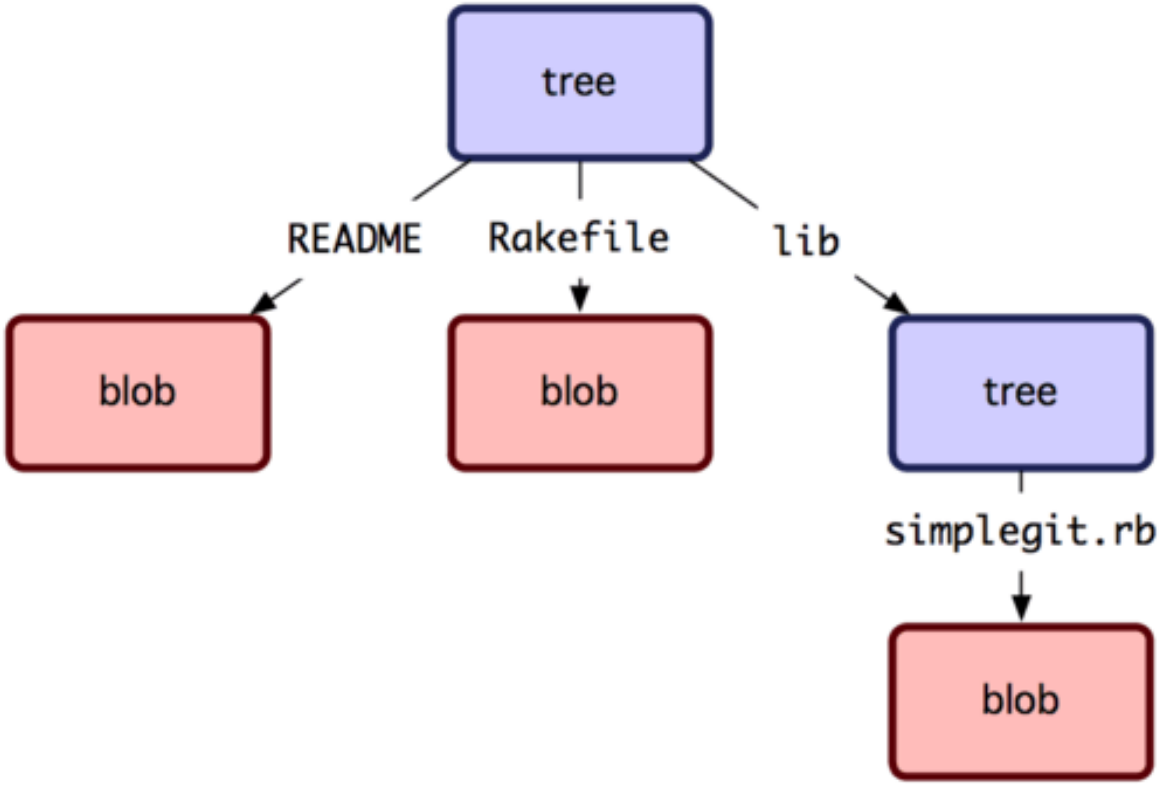
- Unterstützt auch Wildcards

Tags

- Tag: Markiert einen Stand als besonders wichtig in der Entwicklung

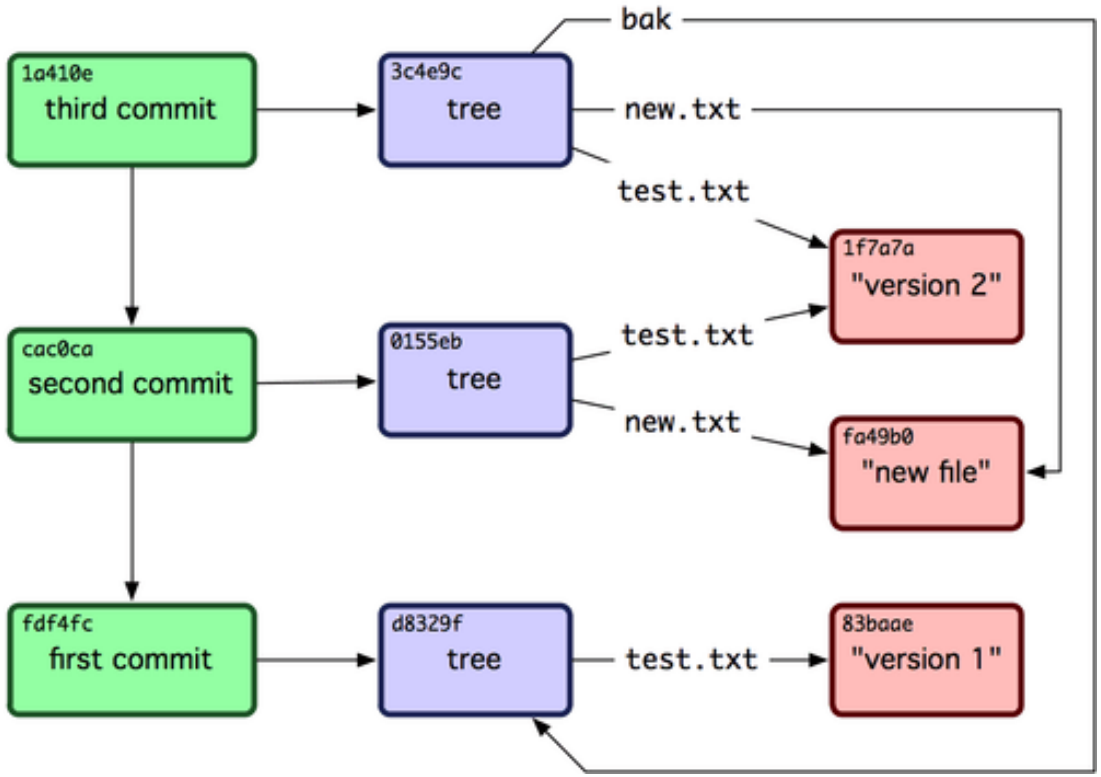
```
$ git tag -a v1.4 -m 'Version 1.4 mit Feature A und B'
$ git tag
v0.1
v1.3
v1.4
```

Bonus Slide: git-Datenstrukturen



Quelle: Scott Chacon - Pro git

Bonus Slide: git-Datenstrukturen



Quelle: Scott Chacon - Pro git