Einführung in git

Johannes Gilger & Matthias Lederhofer
Rechen- und Kommunikationszentrum der RWTH Aachen
Network Operation Center

14. Juli 2010

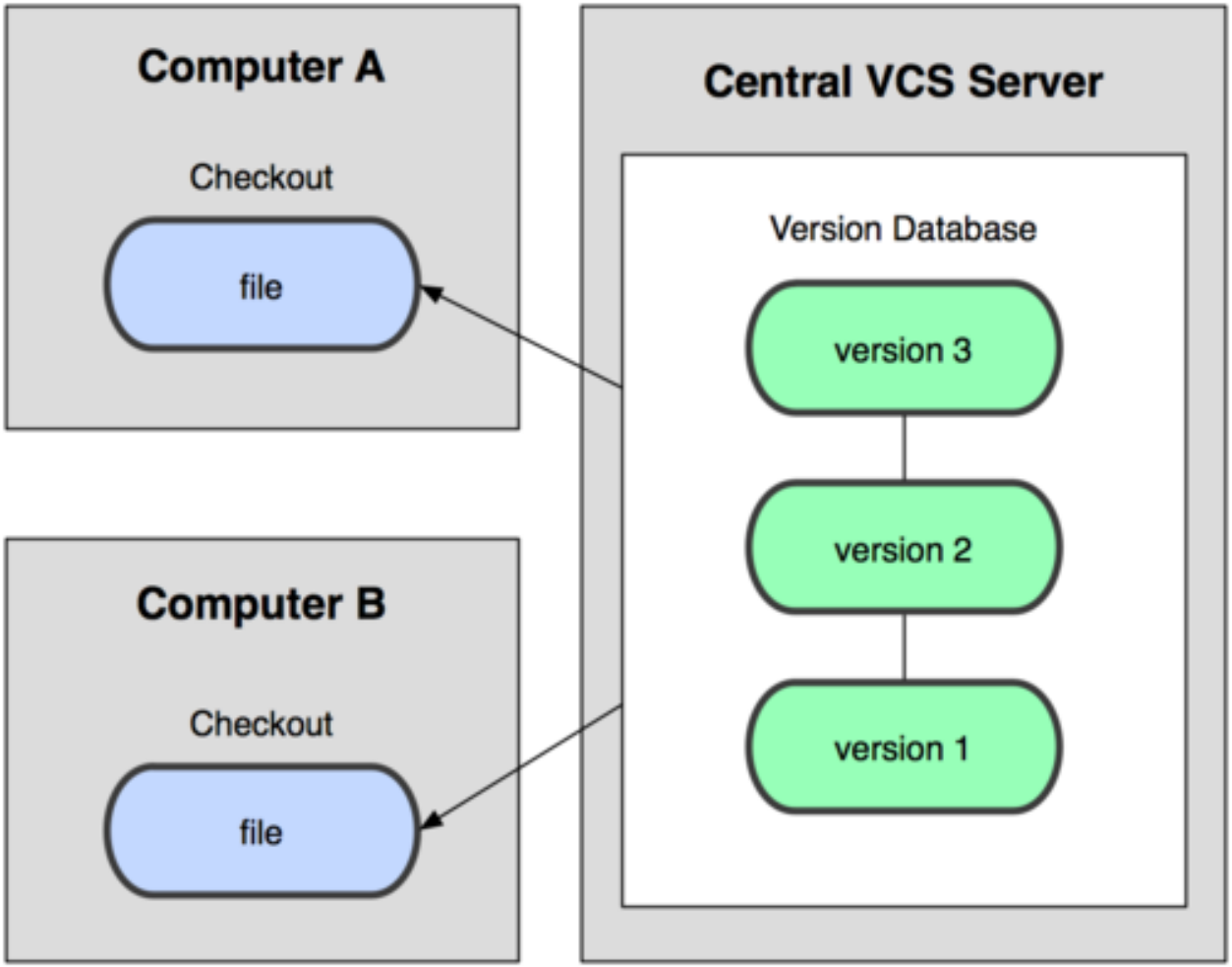
Übersicht

- Begriffe in der Versionsverwaltung
- Unterschiede zentrale und dezentrale VCS
- Warum man git benutzen sollte
- Wie git funktioniert
- Wie wir git benutzen
- Weiterführende Literatur und Software

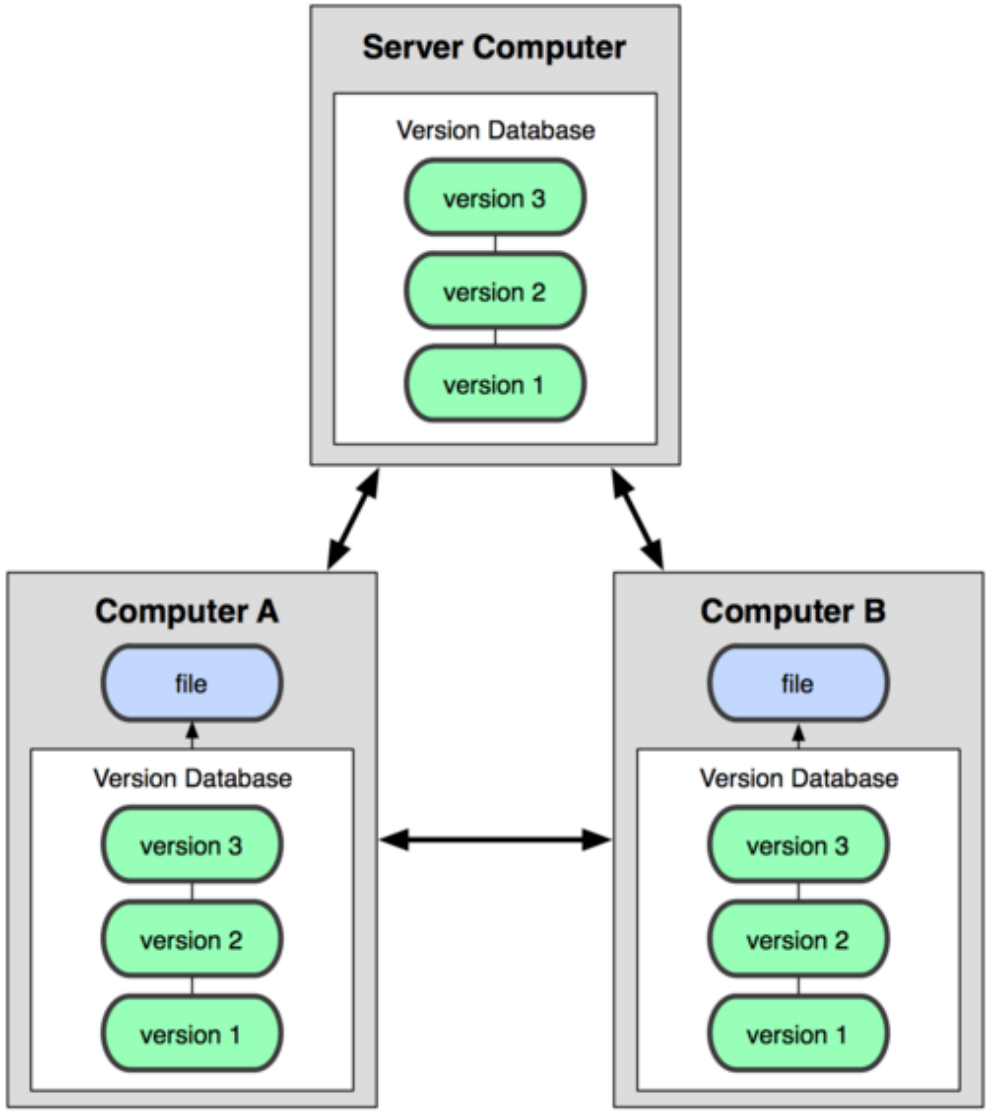
Begriffe

- **DVCS, VCS, SCM** - (Verteilte) Versionsverwaltung
- **Repository** - Enthält die Geschichte (alle Versionen) eines Projekts
- **Commit, Revision** - Identifiziert eine Version
  SVN: 423, git: `b8bba41...`
- **Branch** - Ein Entwicklungszweig in der Geschichte
- **Worktree** - Das aktuelle Arbeitsverzeichnis (Dateien)
- **Checkout** - Die angelegten Dateien die einem bestimmten Stand entsprechen

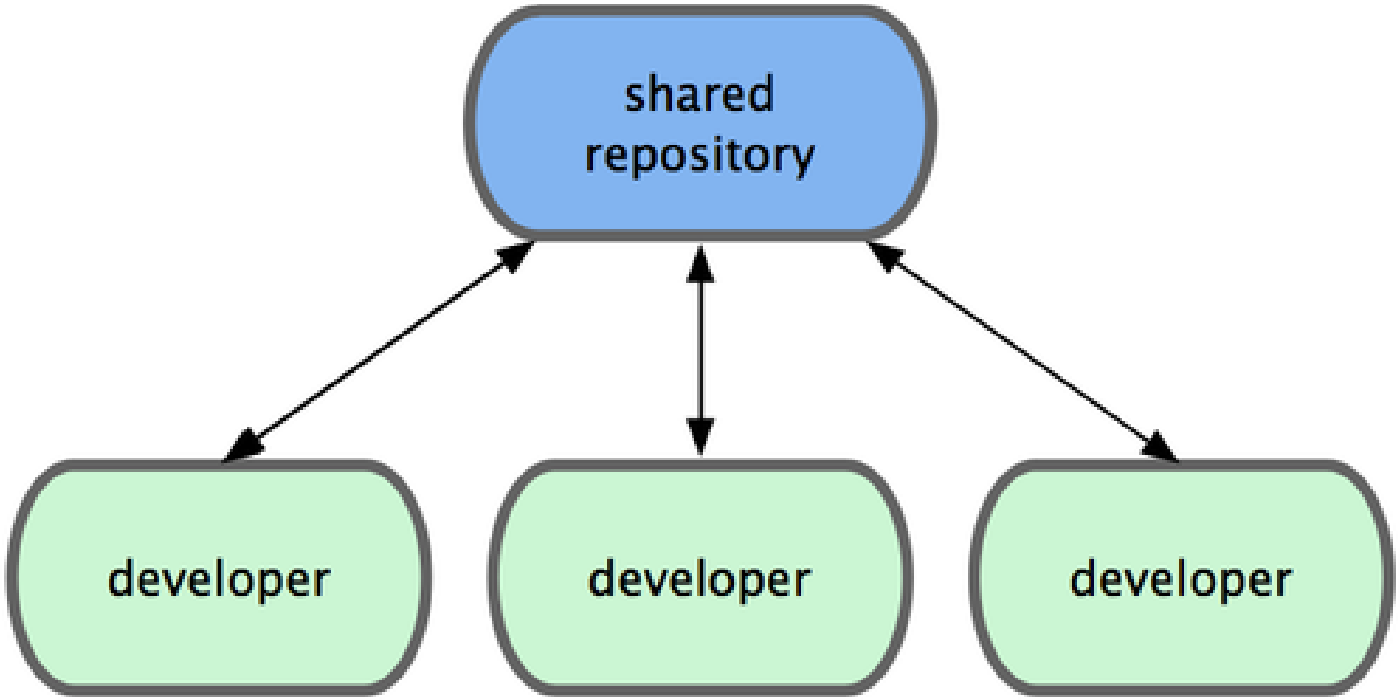
Zentrale Versionskontrolle ...



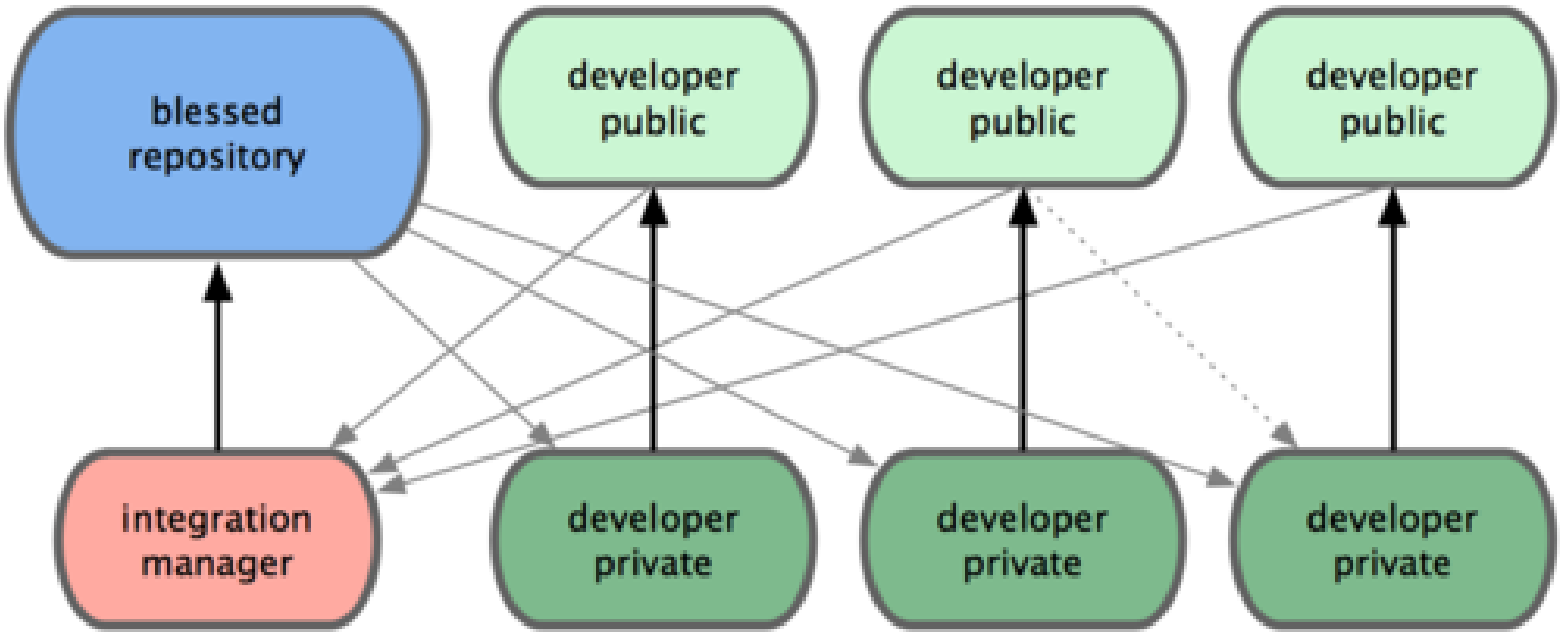
Dezentrale Versionskontrolle



Typischer zentralisierter Workflow



Typischer verteiler Workflow



Festlegung auf Master-Repository ist nur Konvention.

Kurze Geschichte von git

- **2005:** Linus Torvalds schreibt git für die Entwicklung des Linux-Kernels
  Anfangs sehr low-level und kompliziert
- **2005-2010:** Linus gibt die Entwicklung ab, schnell wachsende und aktive Community
- **2010:** git 1.7.1.1 ist aktuell, viele Open-Source Projekte sind inzwischen auf git umgestiegen
  Android, Debian, Eclipse, Fedora, GIMP, Gnome, openSUSE, Perl, Ruby on Rails, Samba, VLC, Wine, X.org

Motivation

”Warum die ganze Arbeit?”

Warum sollte man git benutzen?

- Viele neue Konzepte, anfangs evtl. schwer zu verstehen
  Ungefähr einen Monat Einarbeitungszeit, zahlt sich jedoch schnell aus
- ”SVN funktioniert doch / Viele Leute benutzen noch SVN”
  `git-svn` hilft bei der Umstellung. Die meisten OSS-Projekte wechseln auf ein DVCS
- ”Ich hab nicht vor Open-Source Software zu schreiben”
  git ist genauso hilfreich wenn man nur allein arbeitet
- Andere Gründe?

Vorteile von git gegenüber SVN

- **Geschwindigkeit**
  git ist *sehr* schnell und speicherzeffizient, alle Operationen sind lokal (offline)
- **Kein Setup**
  `git init` und sofort loslegen, keine Erstellung von zentralen Projekten nötig
- **Einfache Rechtevergabe**
  Normale Dateisystemberechtigungen, Zugriff lokal oder z.B. über SSH
- **Schnelle parallele Entwicklung**
  Keine Festlegung auf Regeln oder Rollen

Siehe auch `http://www.whygitisbetterthanx.com` ;)

”Nachteile” von git

- **Kein Checkout von Subdirs**
  Kein wirklicher Nachteil, bedeutet dass man Projekte eher in einzelne Repositories aufteilt
- **Keine feine Rechtevergabe für Server**
  Oft nicht nötig durch Workflow (Maintainer, Teams)
- **Potentielles Chaos mit mehreren Leuten**
  Wenn sich nicht auf einen Workflow geeignigt wird

Technischer Hintergrund

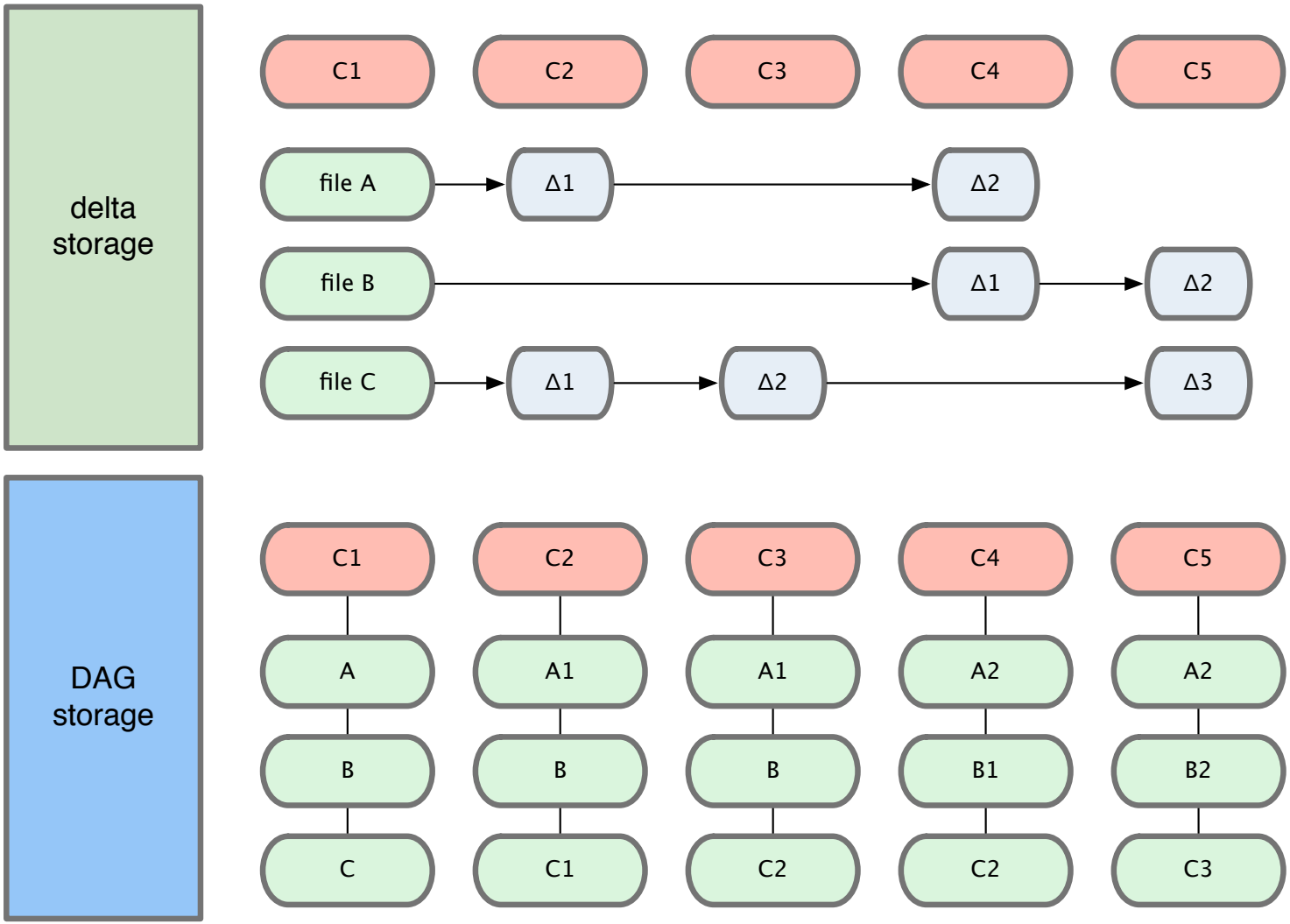
”Behind the curtain. . .”

git Datenstrukturen und Dateilayout

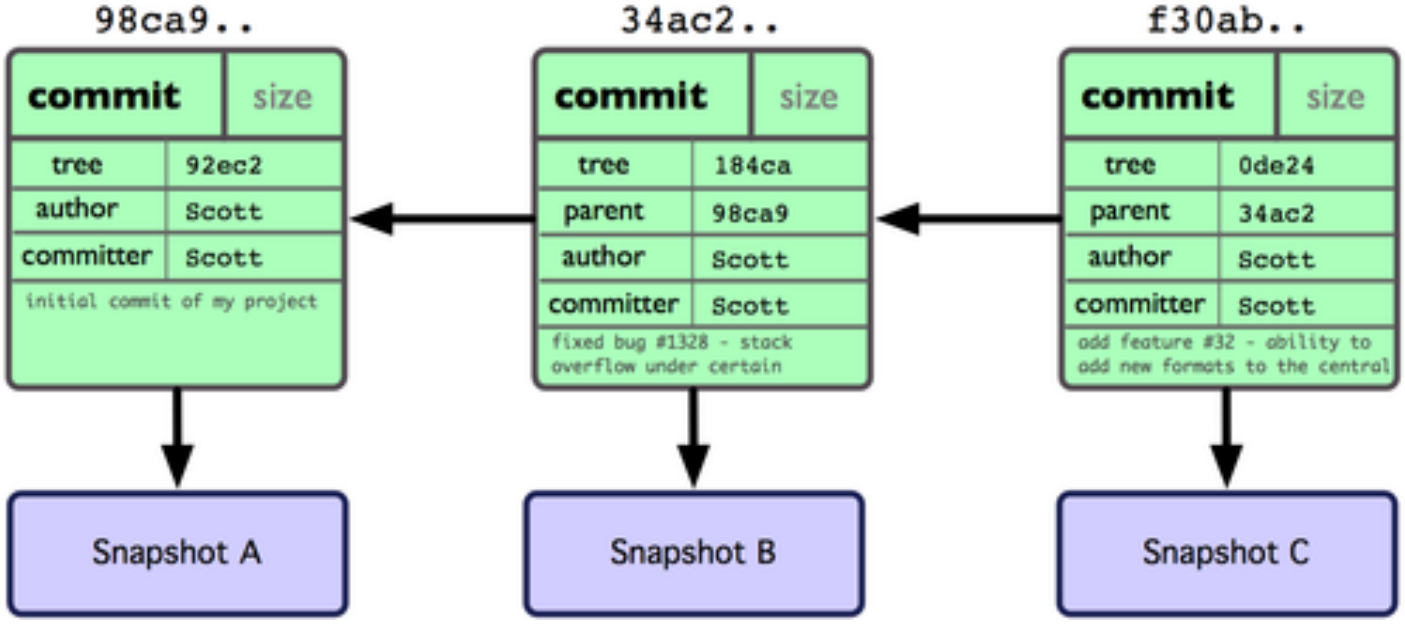
Die Datenstrukturen von git zu kennen...

- Hilft enorm die Funktion der Befehle zu verstehen.
- Vermeidet falsche Arbeitsabläufe.
- Ist kein großer Aufwand.

Delta-Storage vs. Snapshots

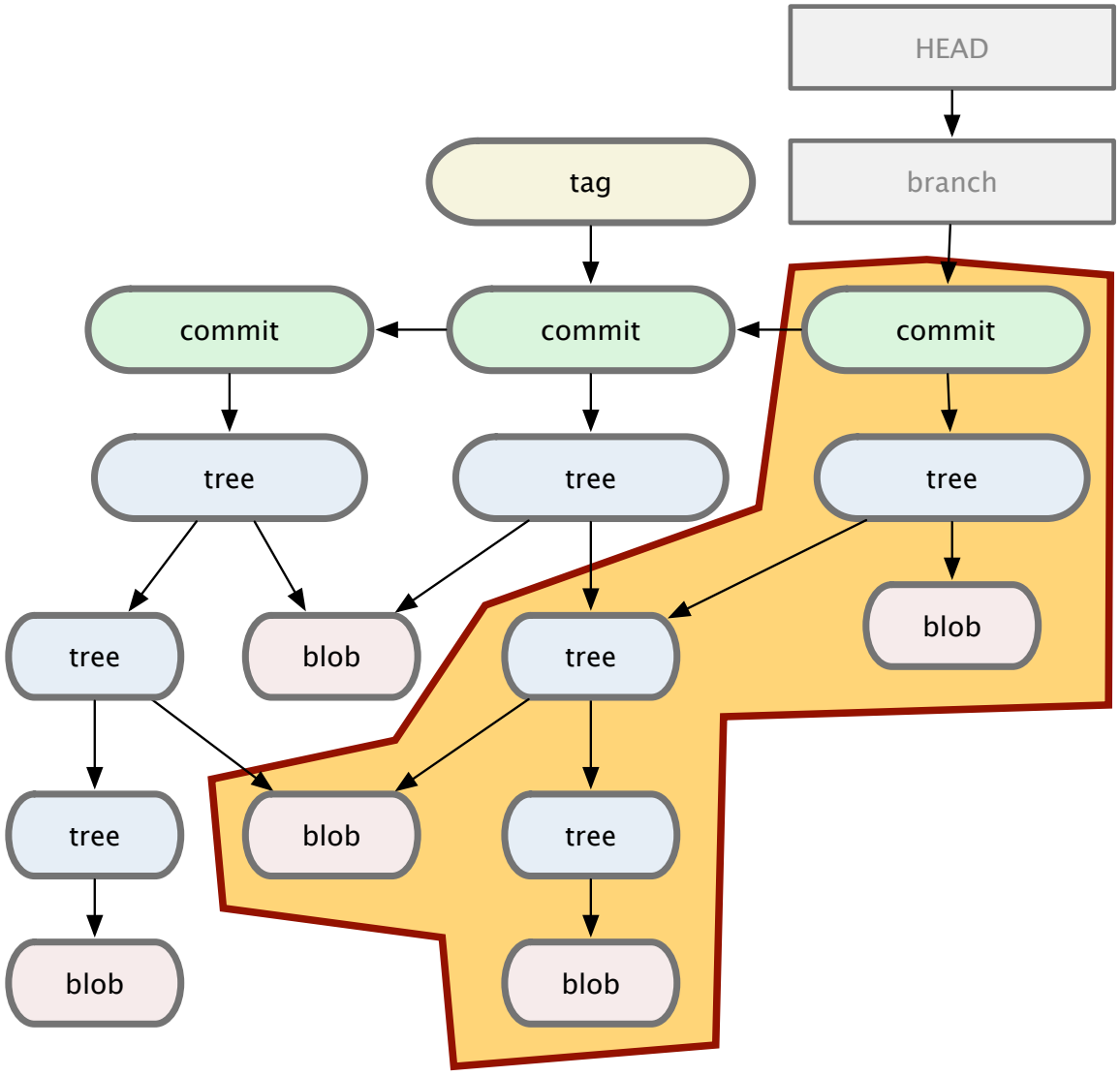


git ist ein Dateisystem mit Versionen

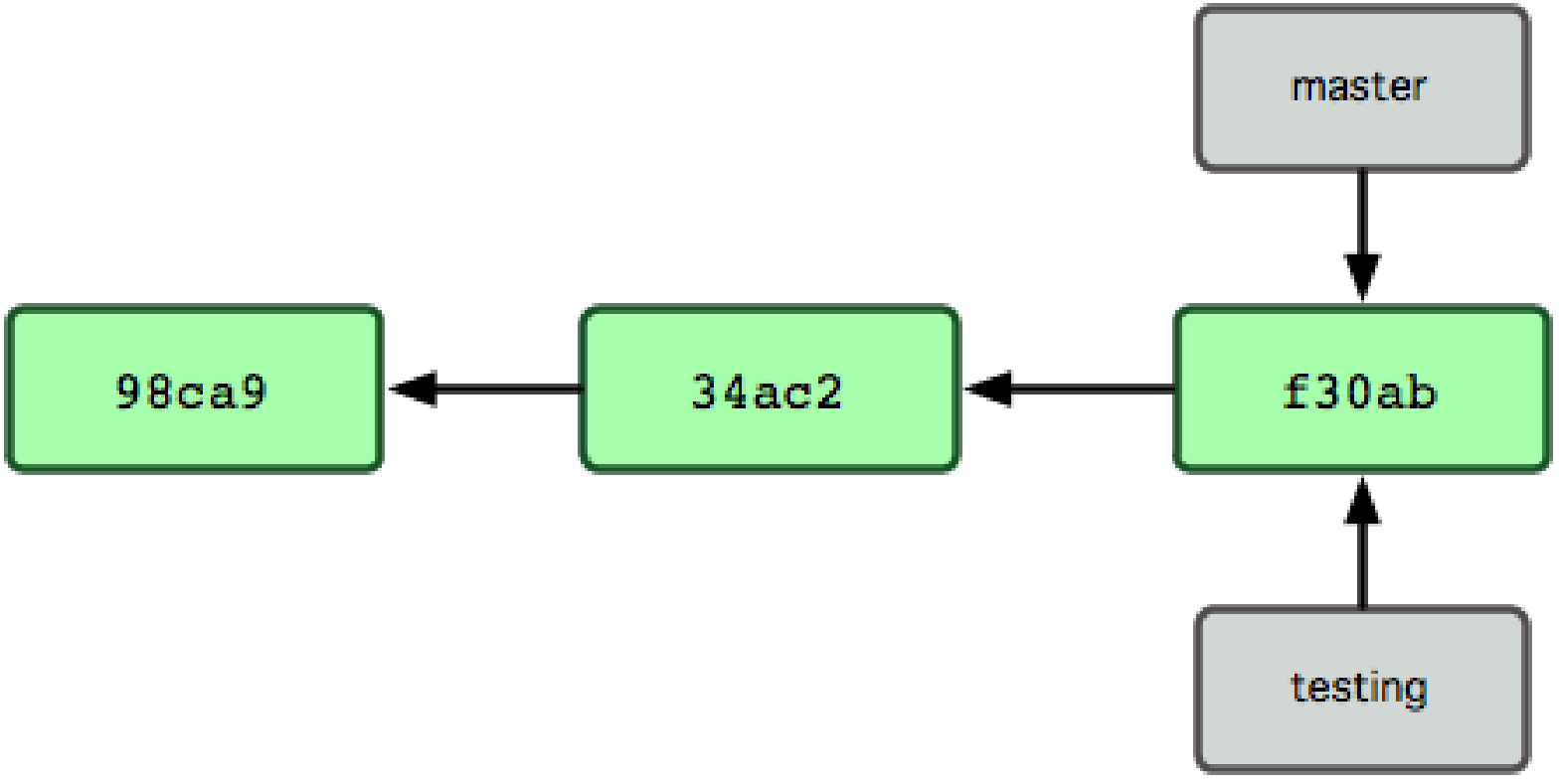


Wichtigstes Konzept: Die History ist eine einfach verlinkte Liste von Commits. Die Links zeigen ”zurück”, zeitlich gesehen.

git ist eine Objekt-Datenbank

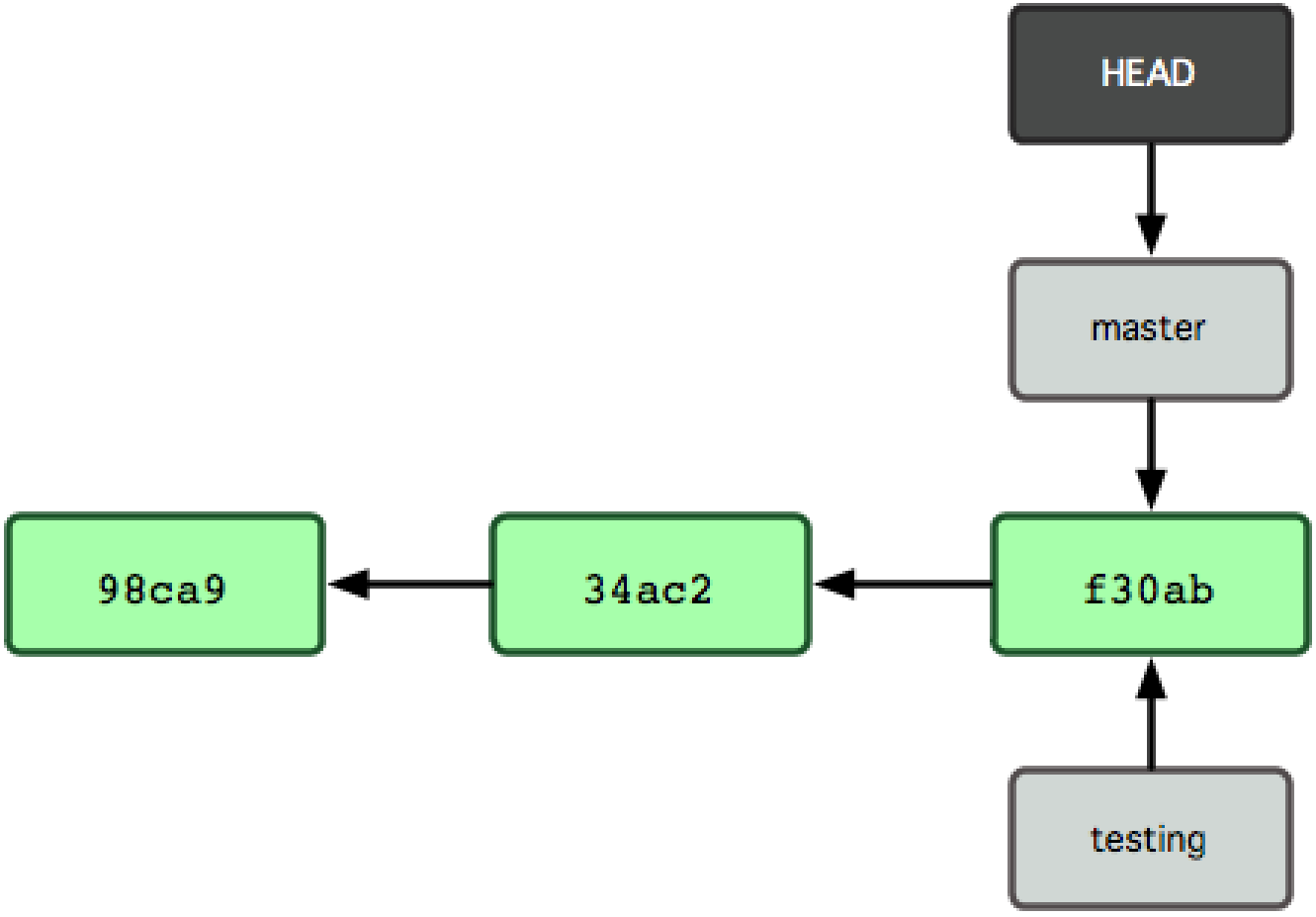


Commits und Branches



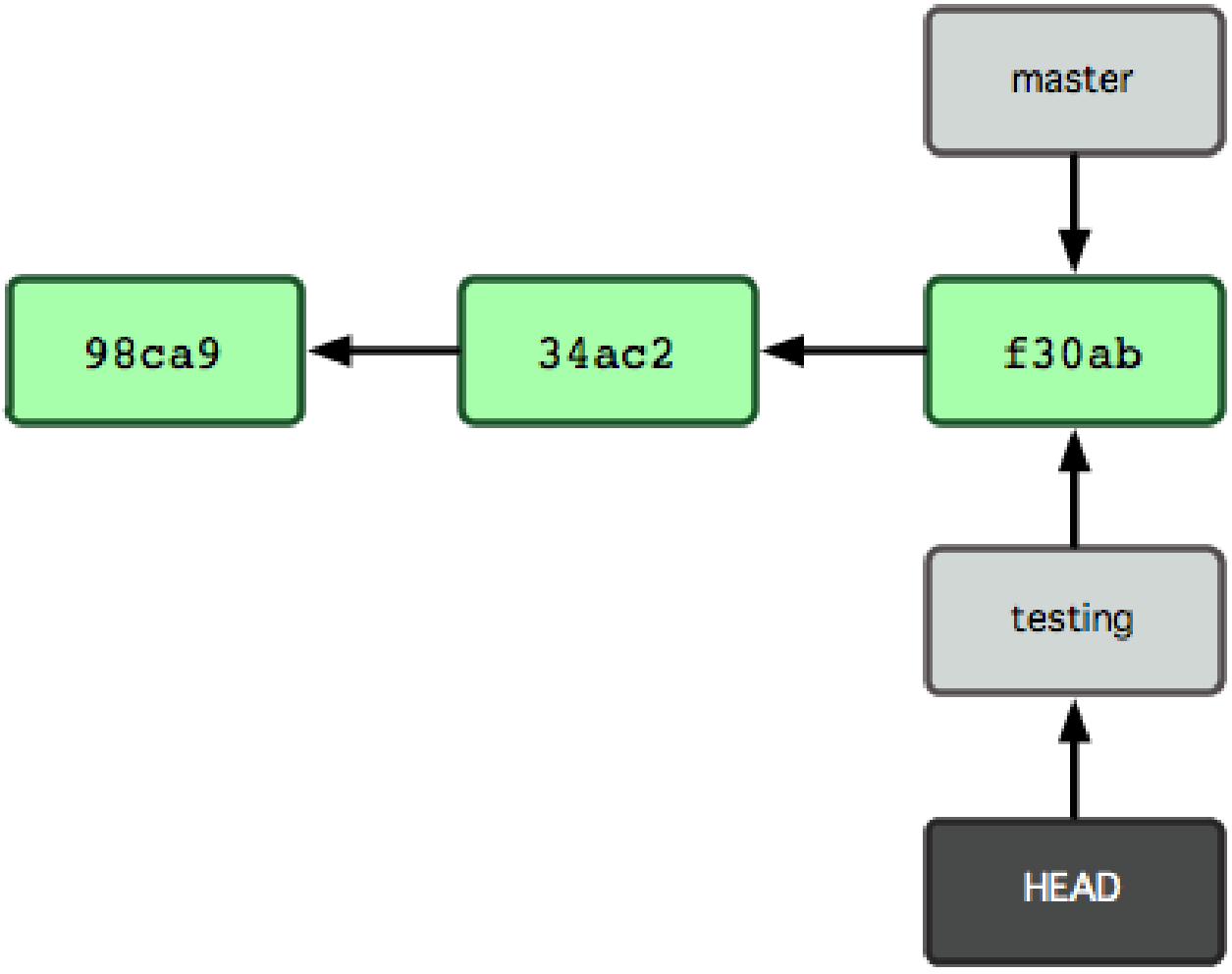
Branches sind nur Zeiger (”references”) auf existierende Commits

HEAD - der aktuelle Branch



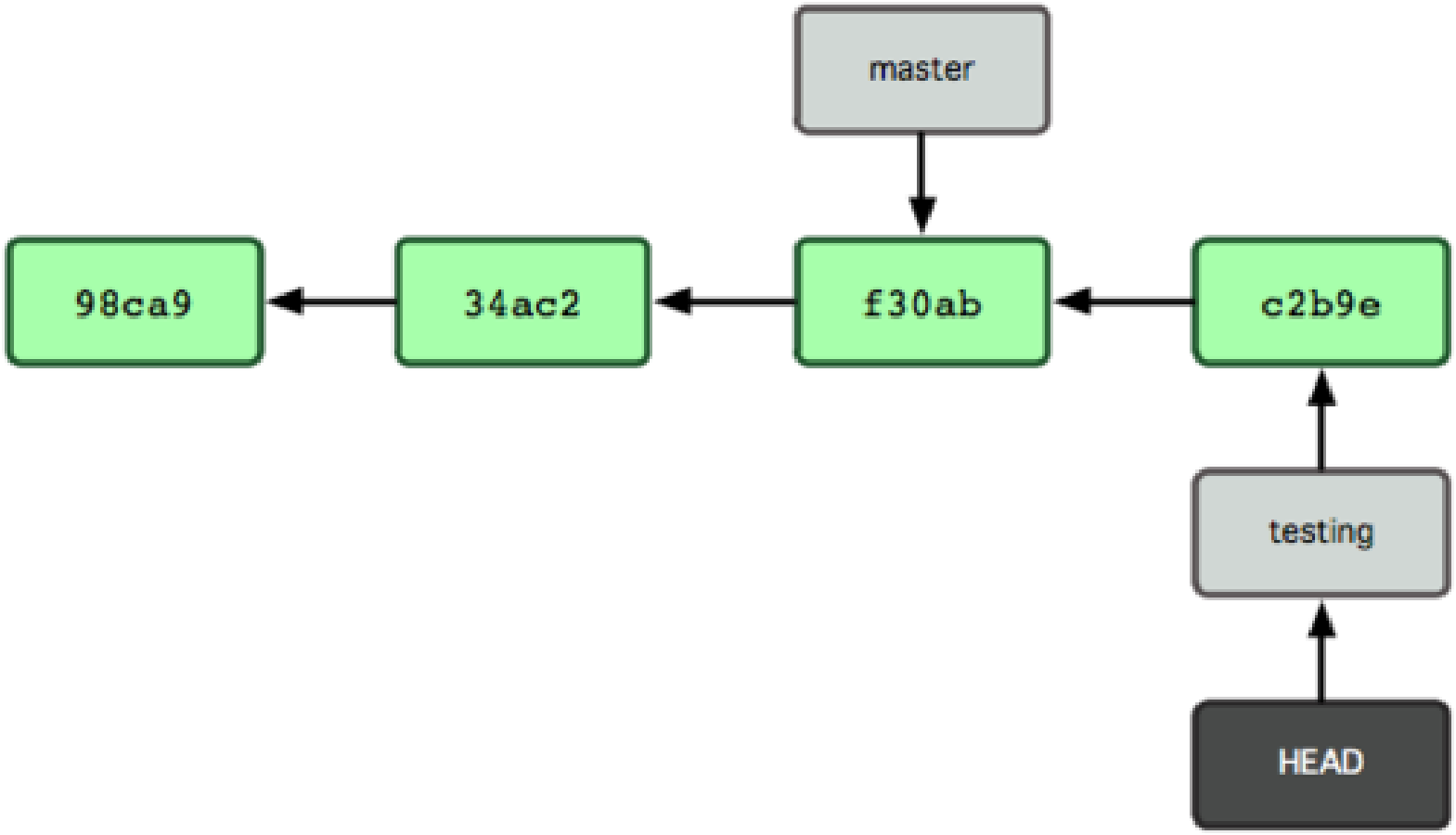
HEAD zeigt an auf welchem Branch aktuell Commits gespeichert werden

git checkout - den aktuellen Branch ändern



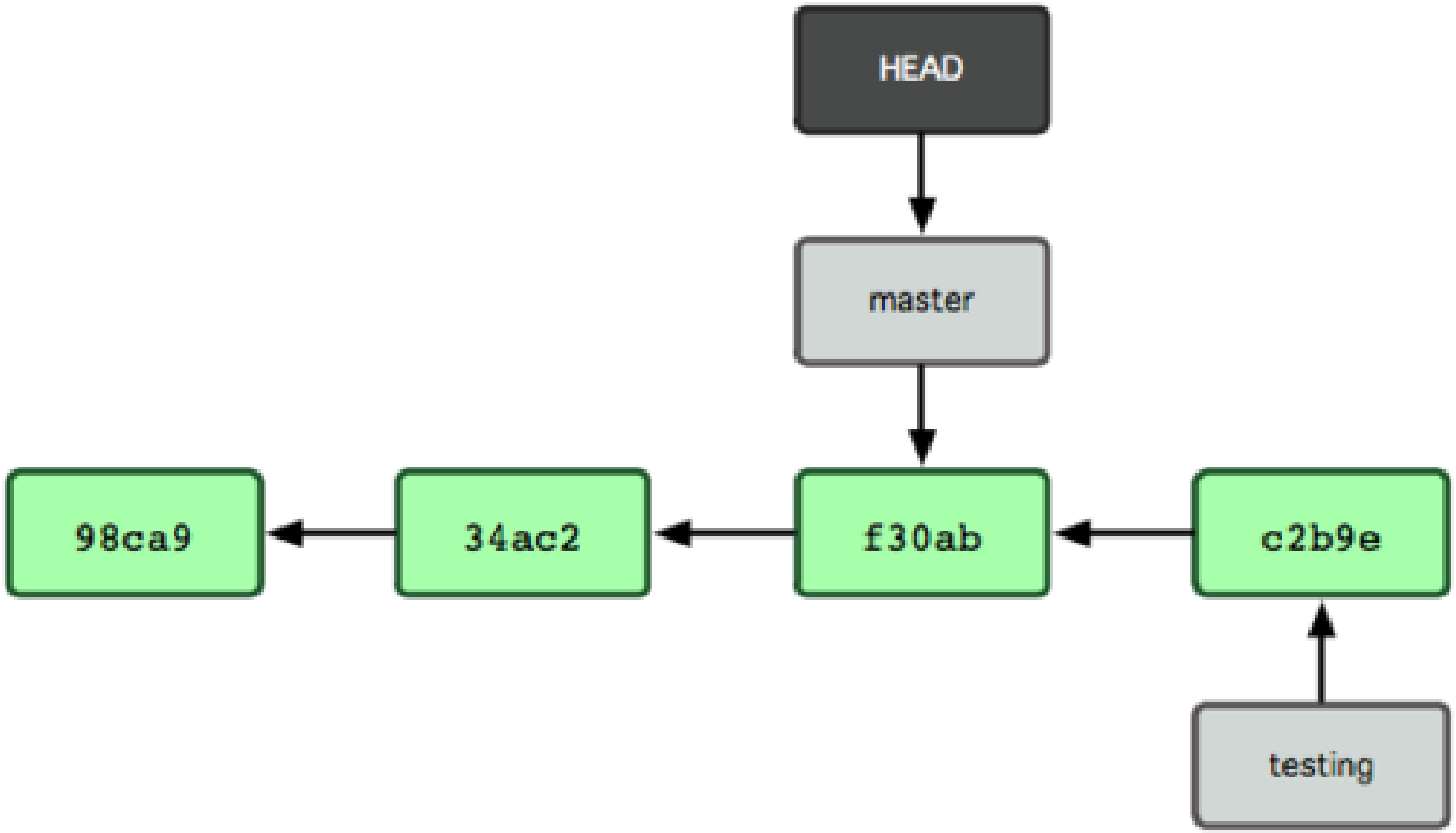
Ein Checkout aktualisiert den Worktree und den HEAD-Zeiger

git commit - einen Commit erstellen



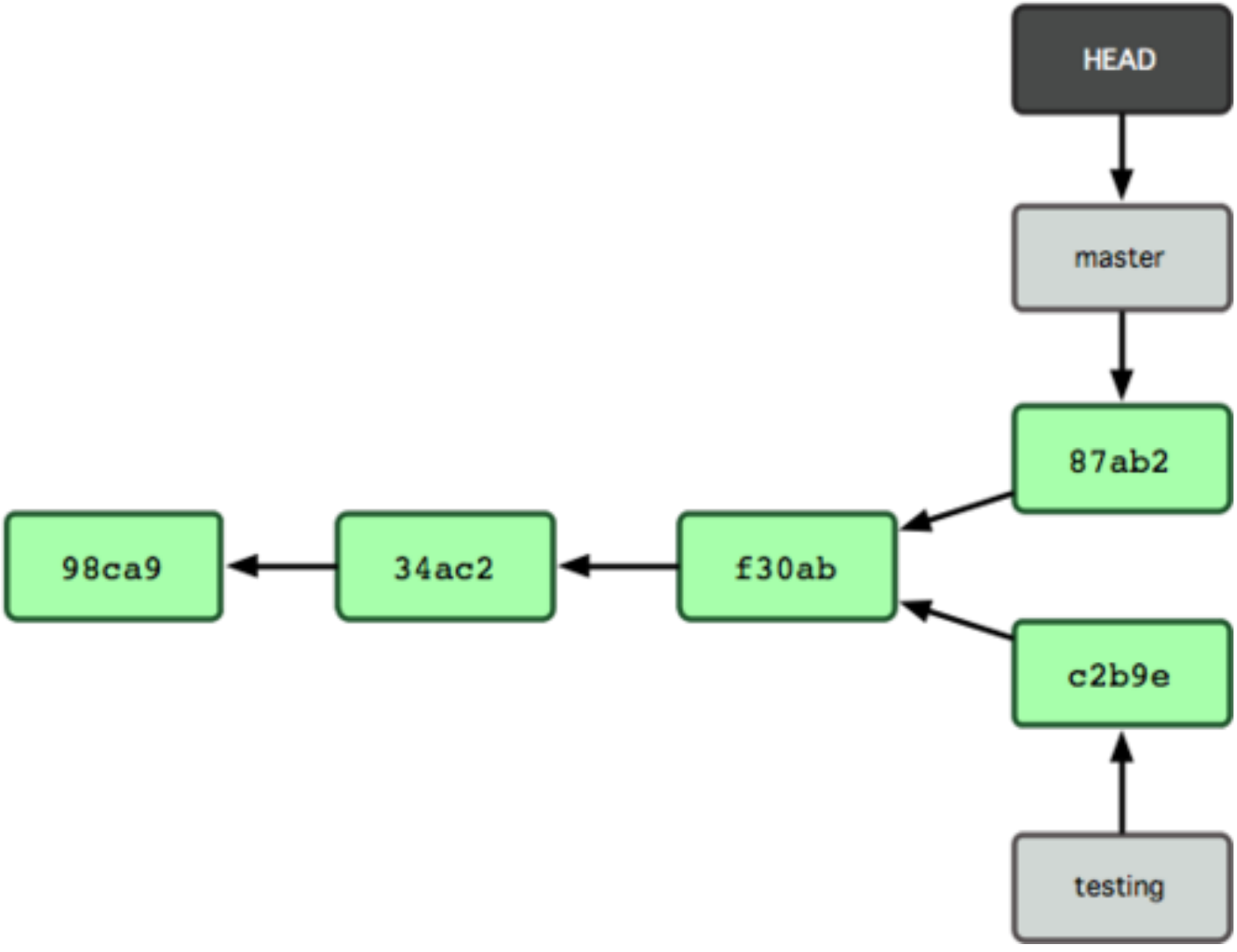
Da HEAD auf ”testing” zeigte wurde hier der Commit angelegt

git checkout - zum ersten Branch zurück

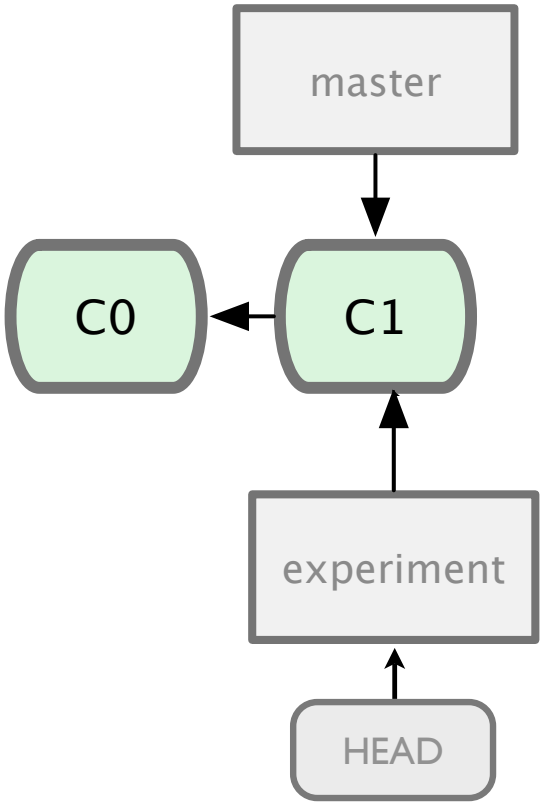


Es müssen währenddessen noch Änderungen am Hauptentwicklungszweig vorgenommen werden

git commit - ein paralleler Branch

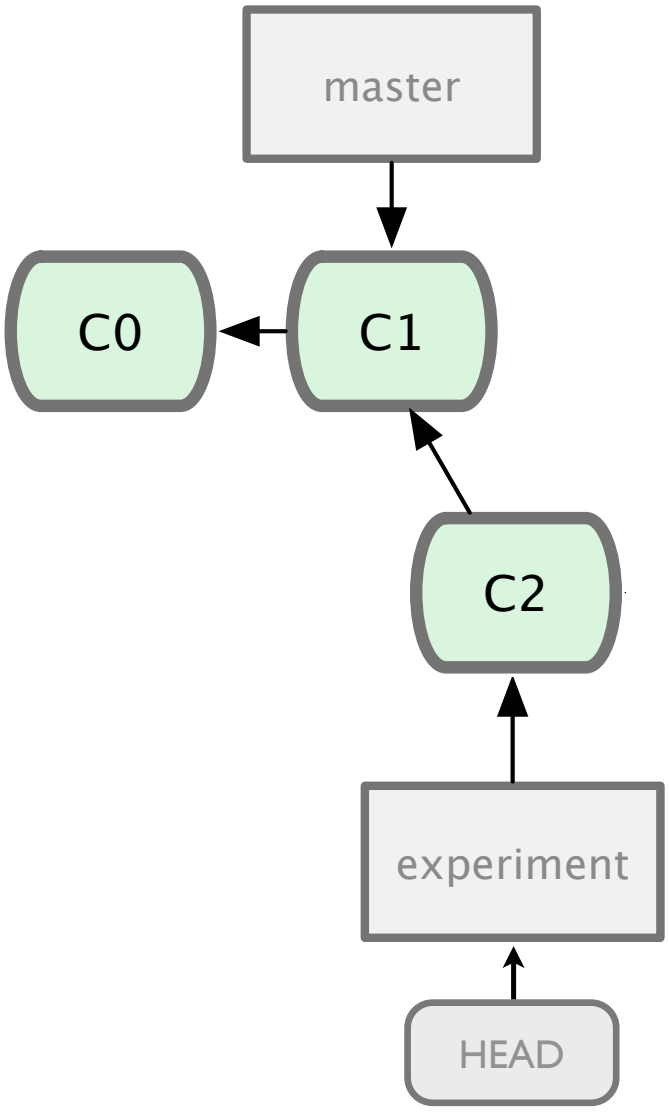


Branching und Merging



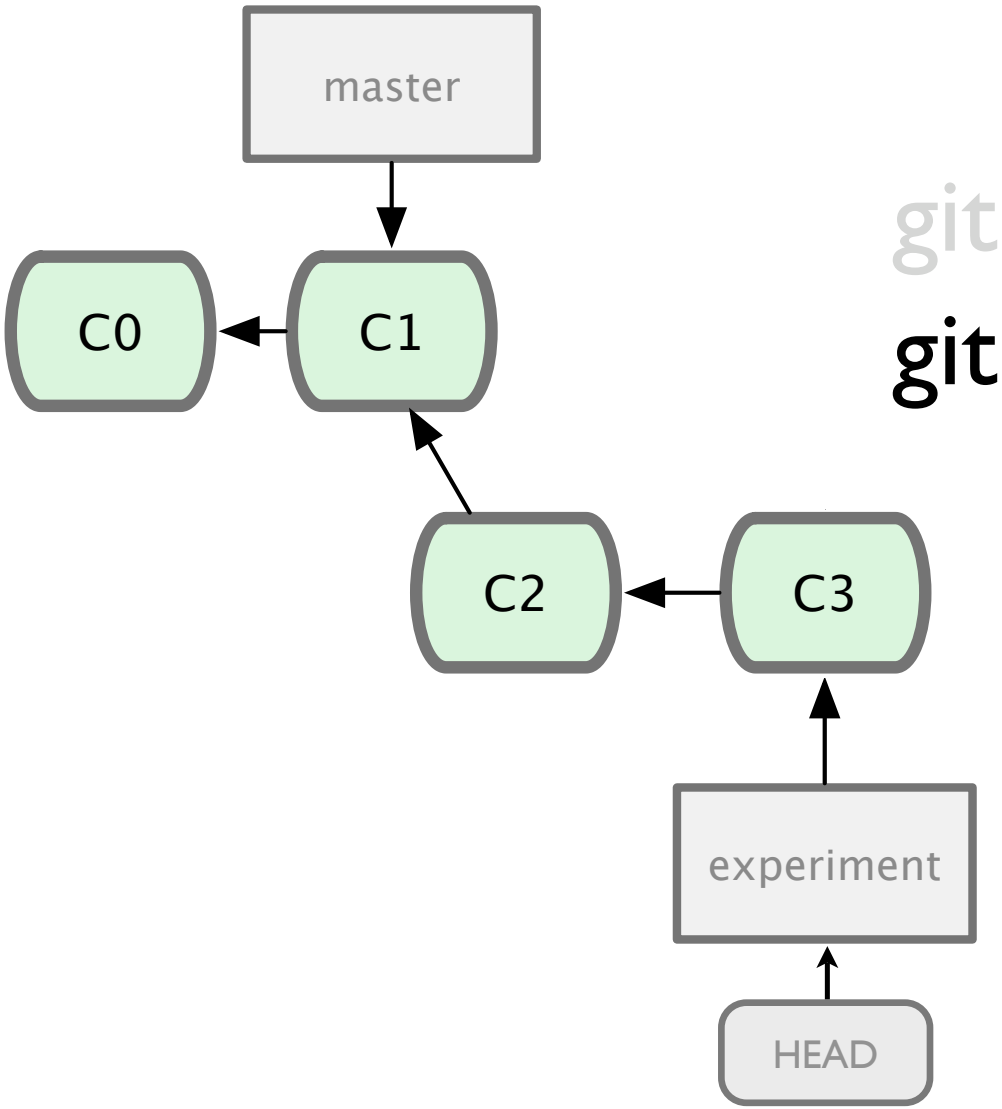
git checkout -b experiment

Branching und Merging



git commit

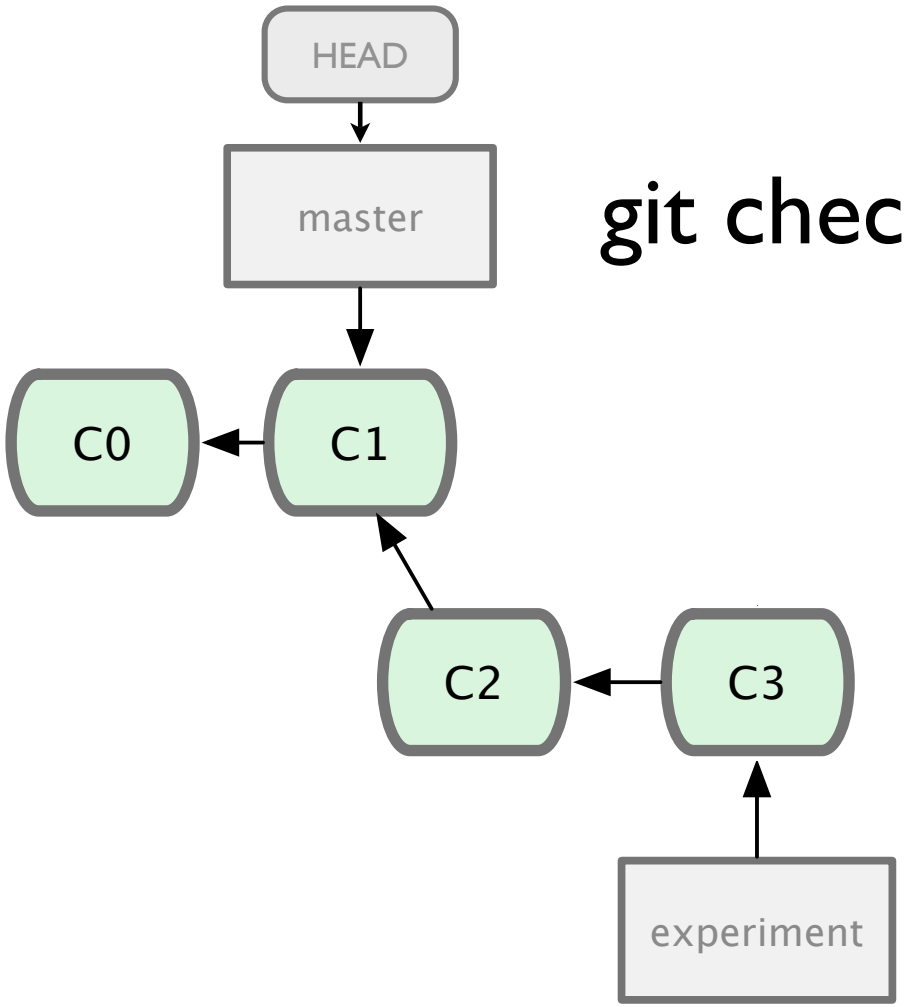
Branching und Merging



git commit

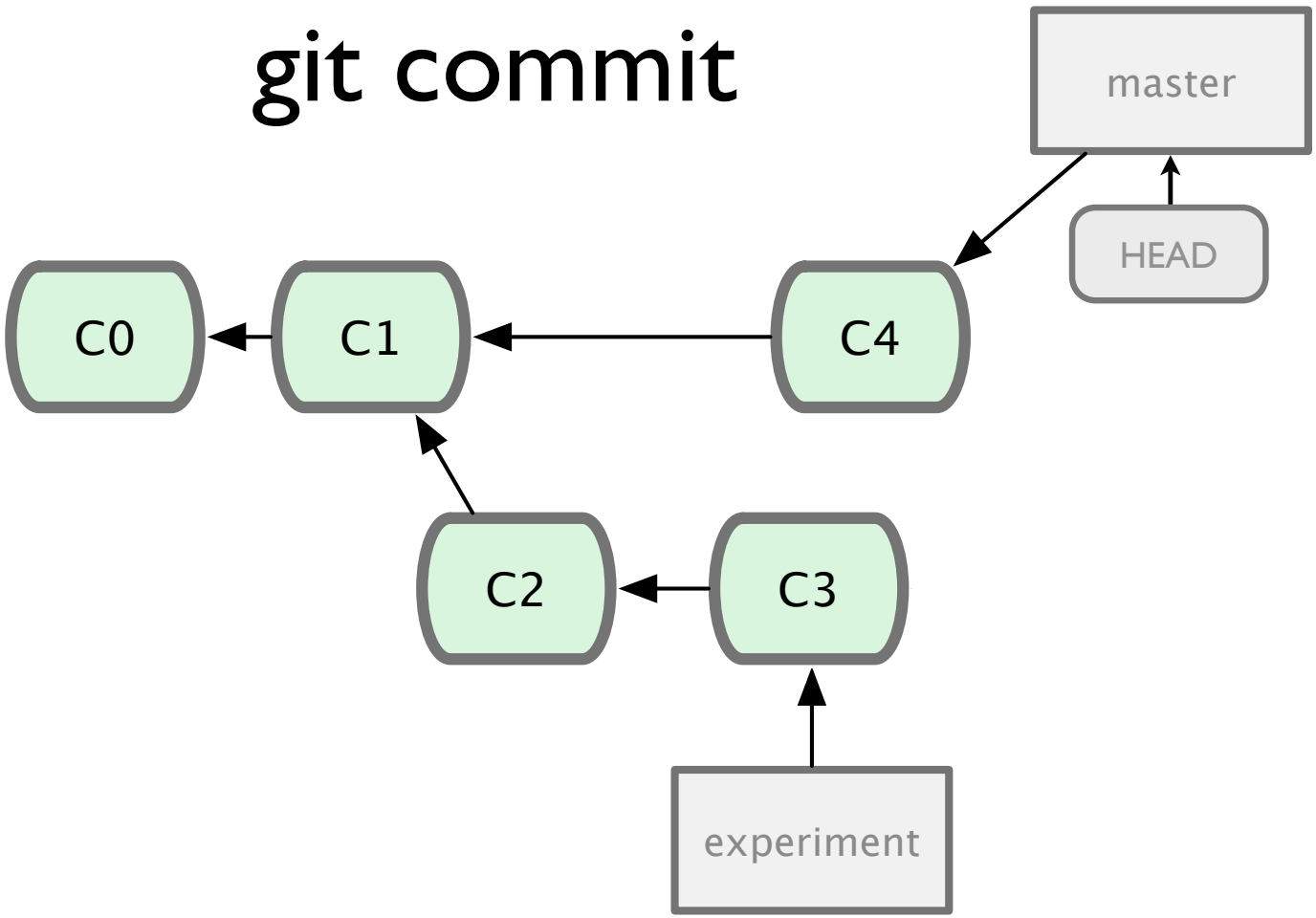
git commit

Branching und Merging

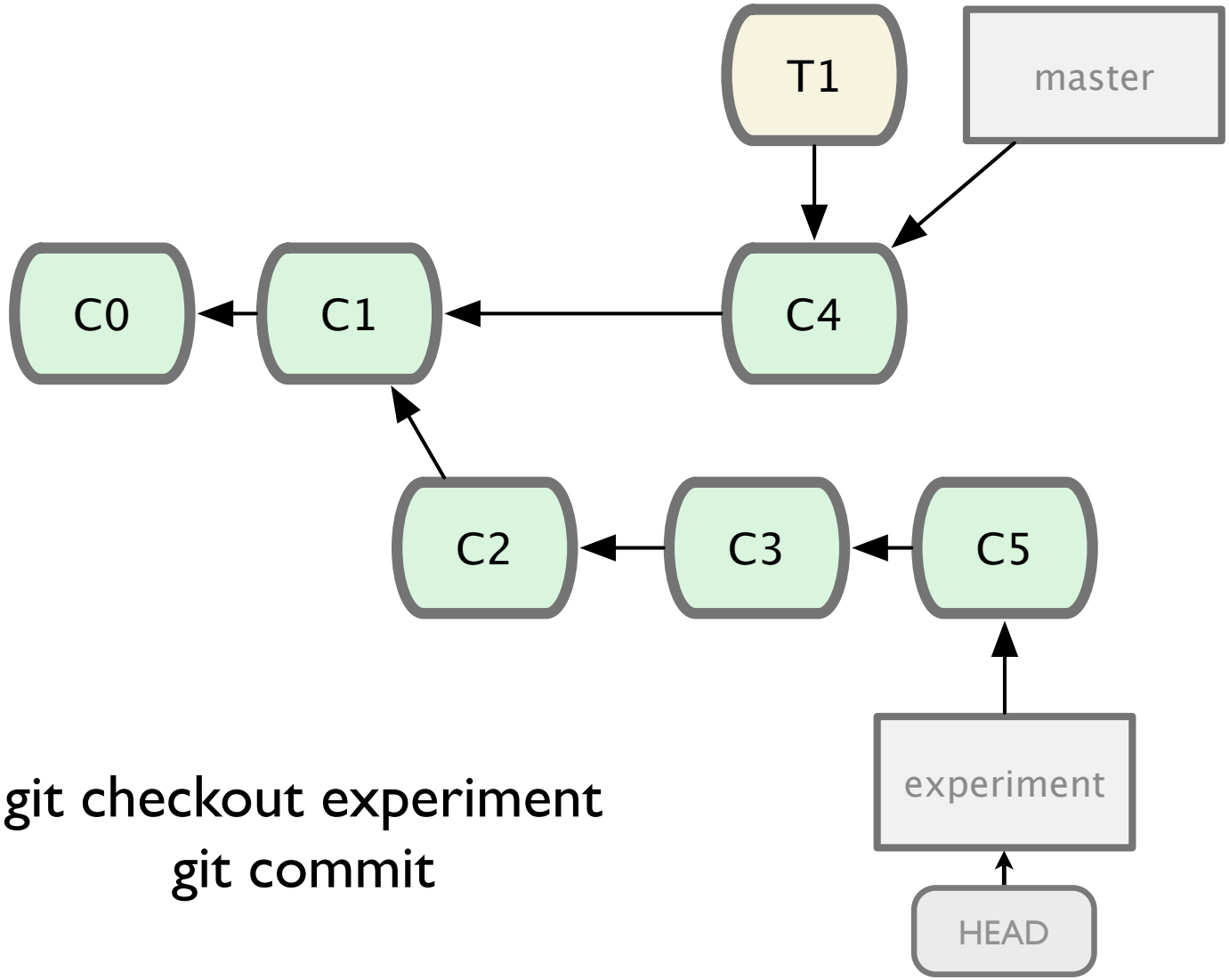


git checkout master

Branching und Merging

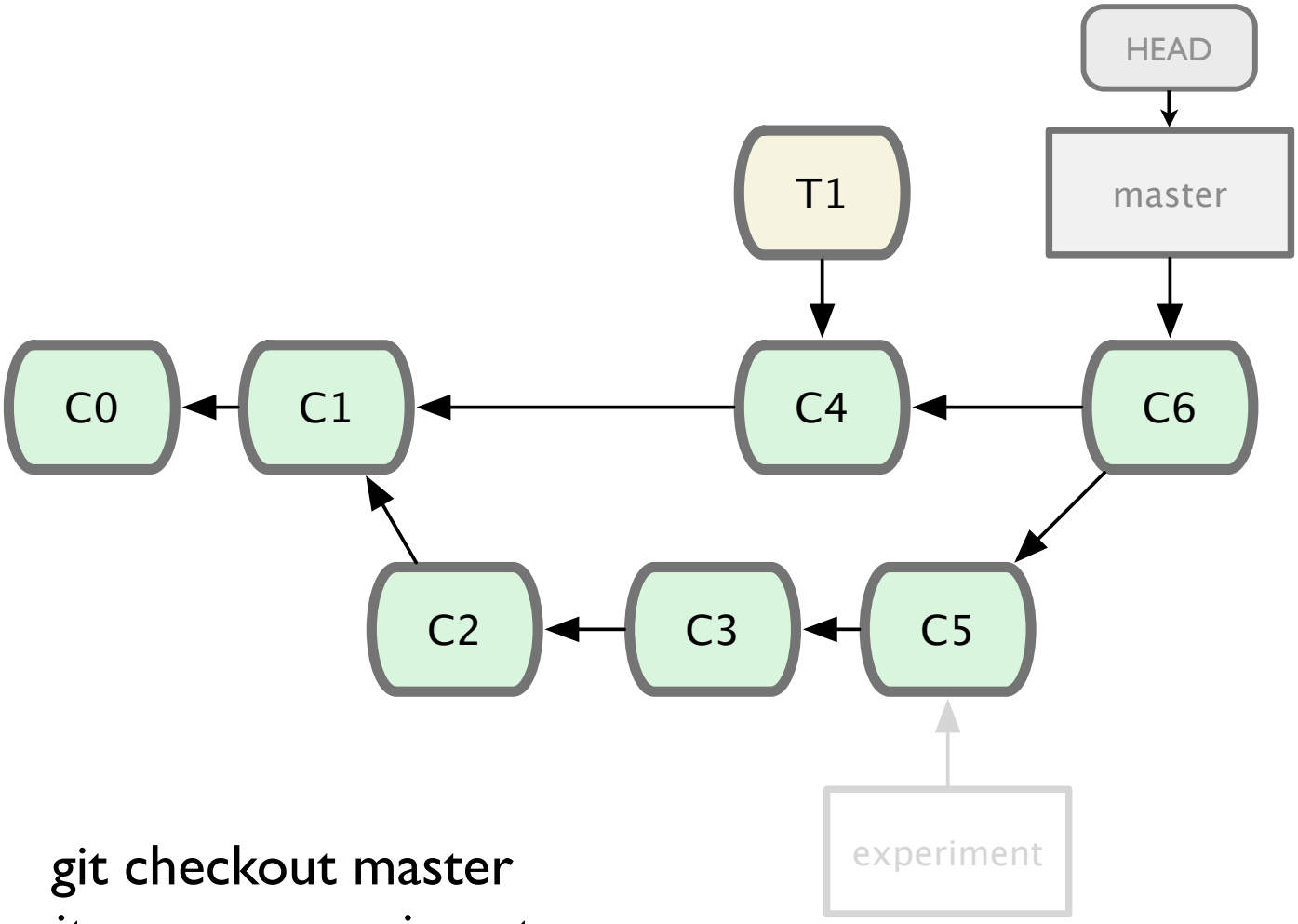


Branching und Merging



git checkout experiment
git commit

Branching und Merging

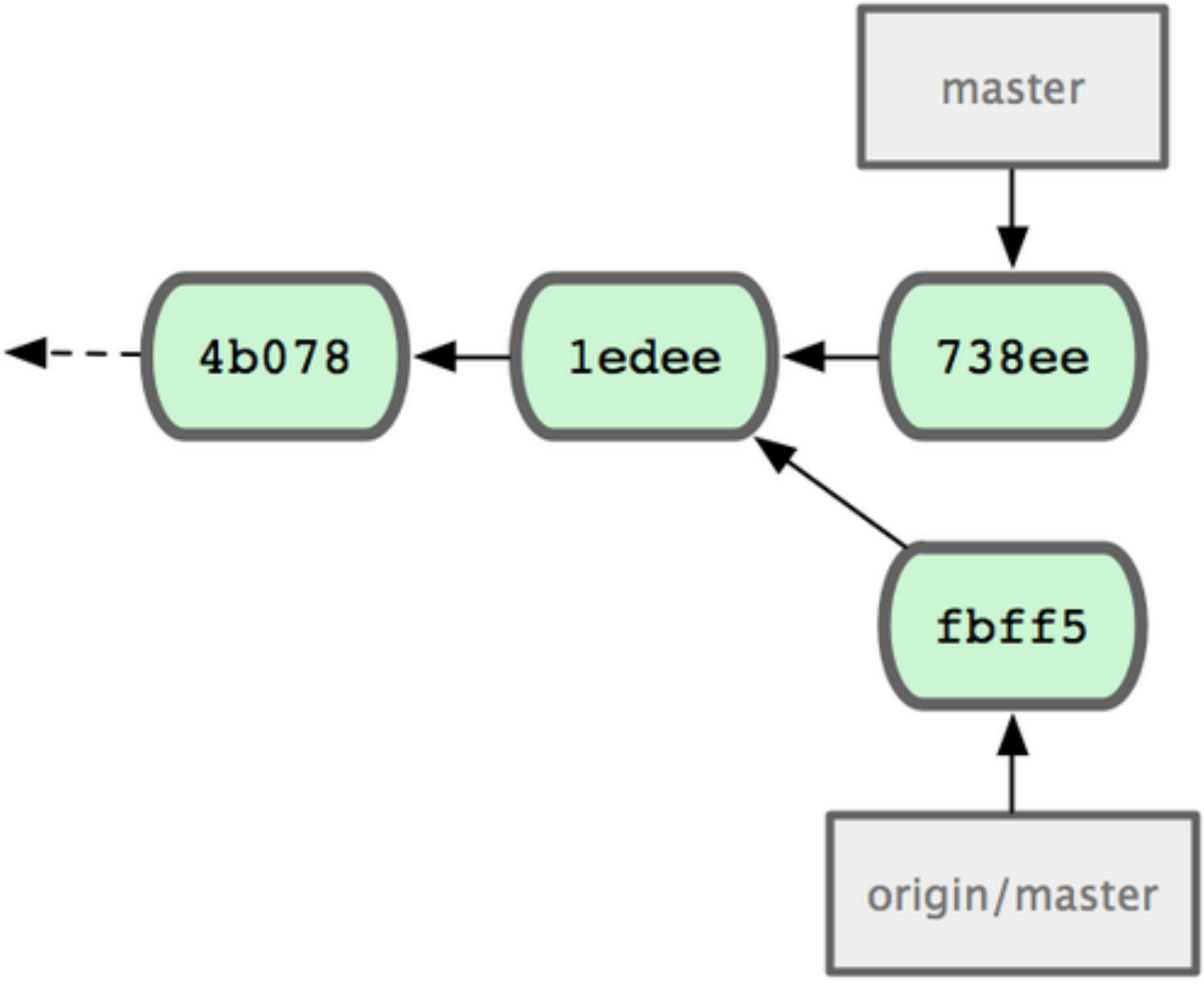


git checkout master
git merge experiment

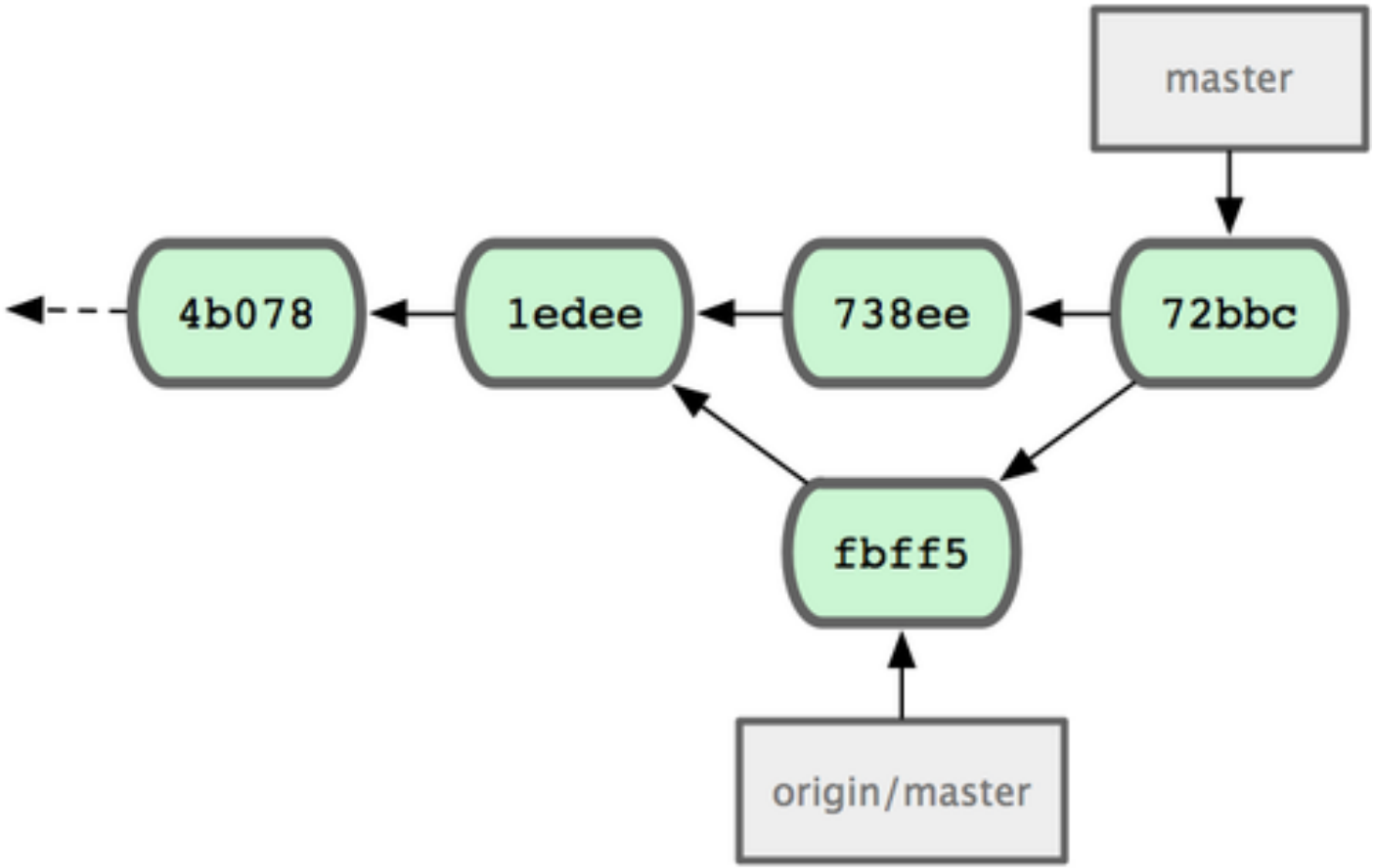
Remotes

- Remotes sind Kürzel um mit anderen Repositories zu arbeiten. Namespaces nach Repository: `jojo/master`
- Remote Branches (Tracking Branches) dienen dazu der Arbeit in anderen Repositories zu folgen
- Entwickelt wird *immer* auf lokalen Branches
- Auf Remote Branches kann nicht committed werden
- Remote Branches werden nur durch fetch / pull geändert

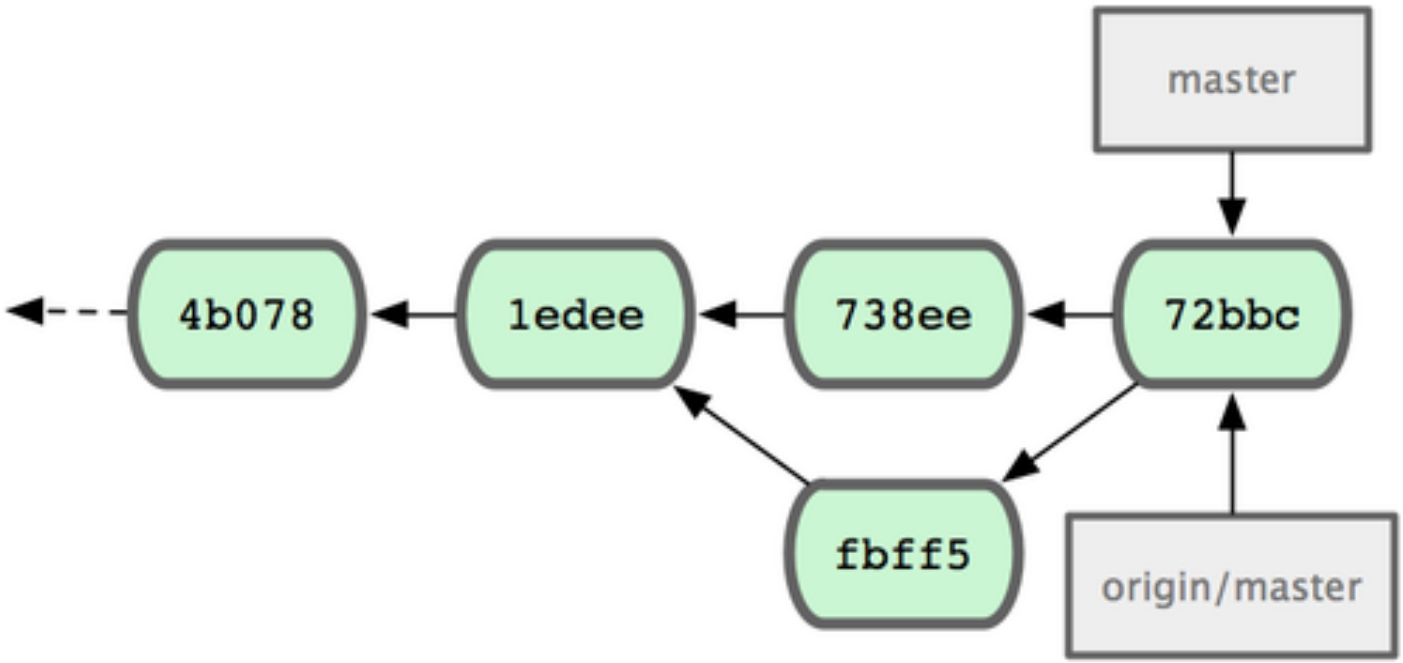
Tracking Branch



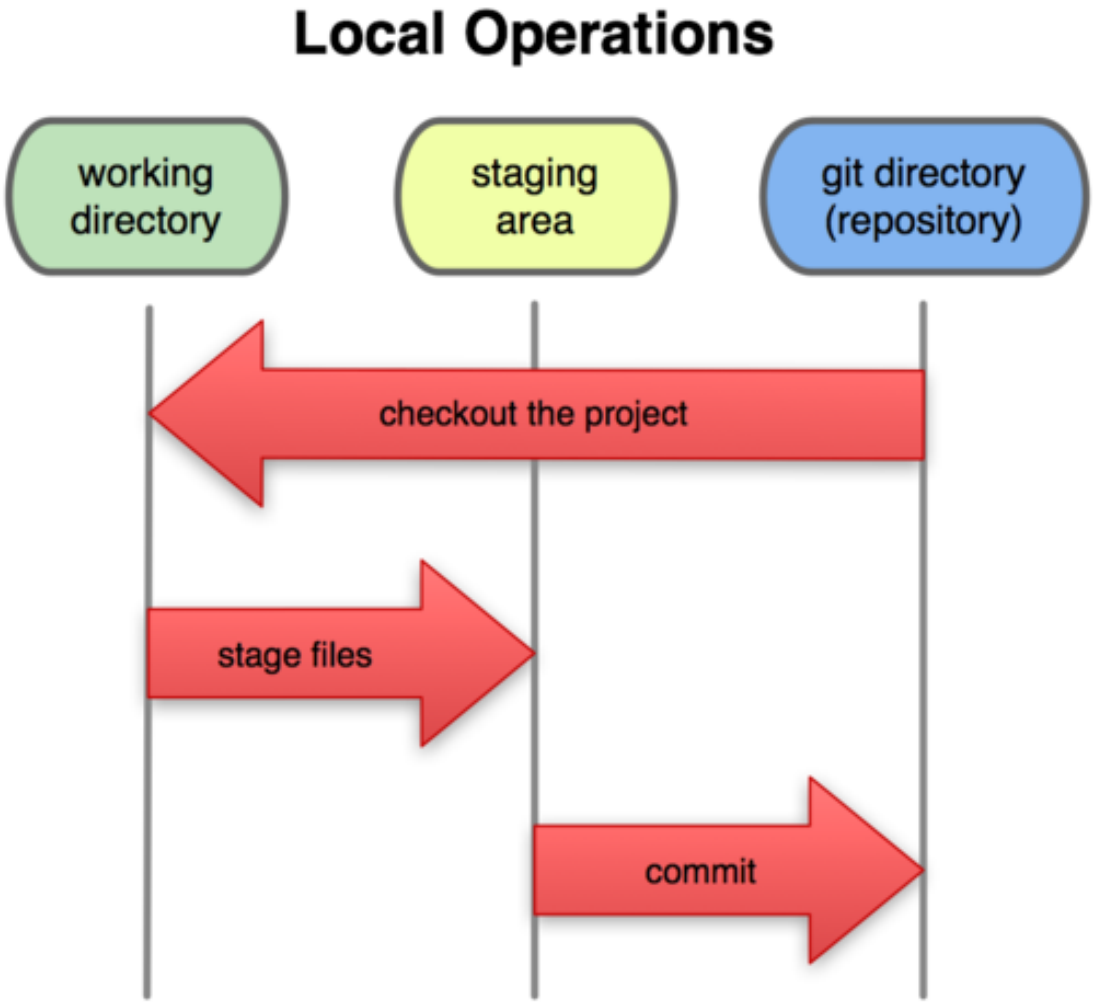
Tracking Branch



Tracking Branch

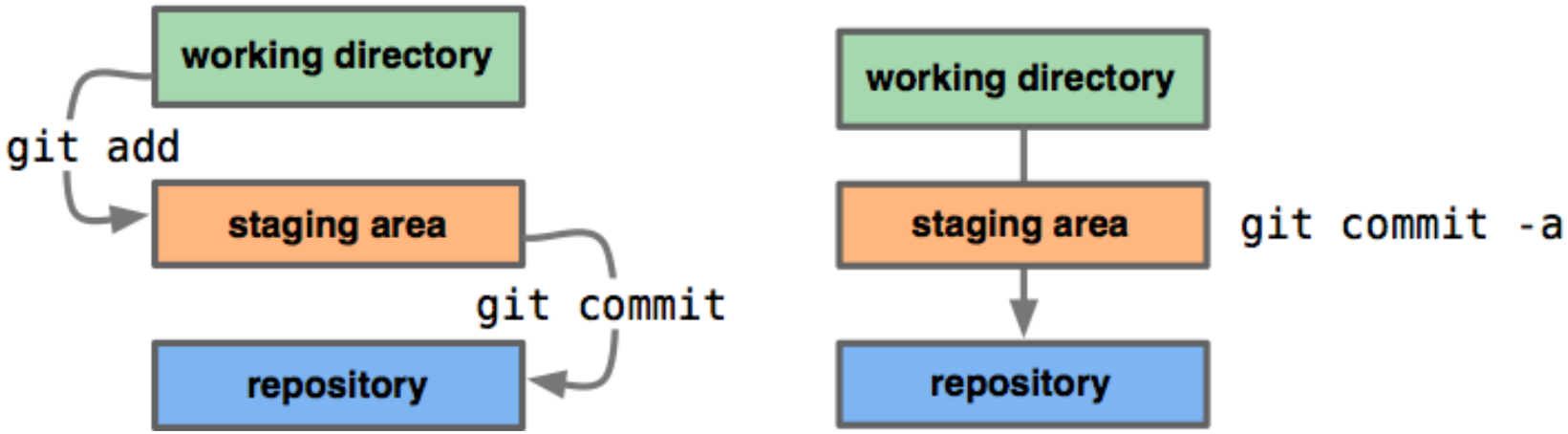


Der Index / Stage



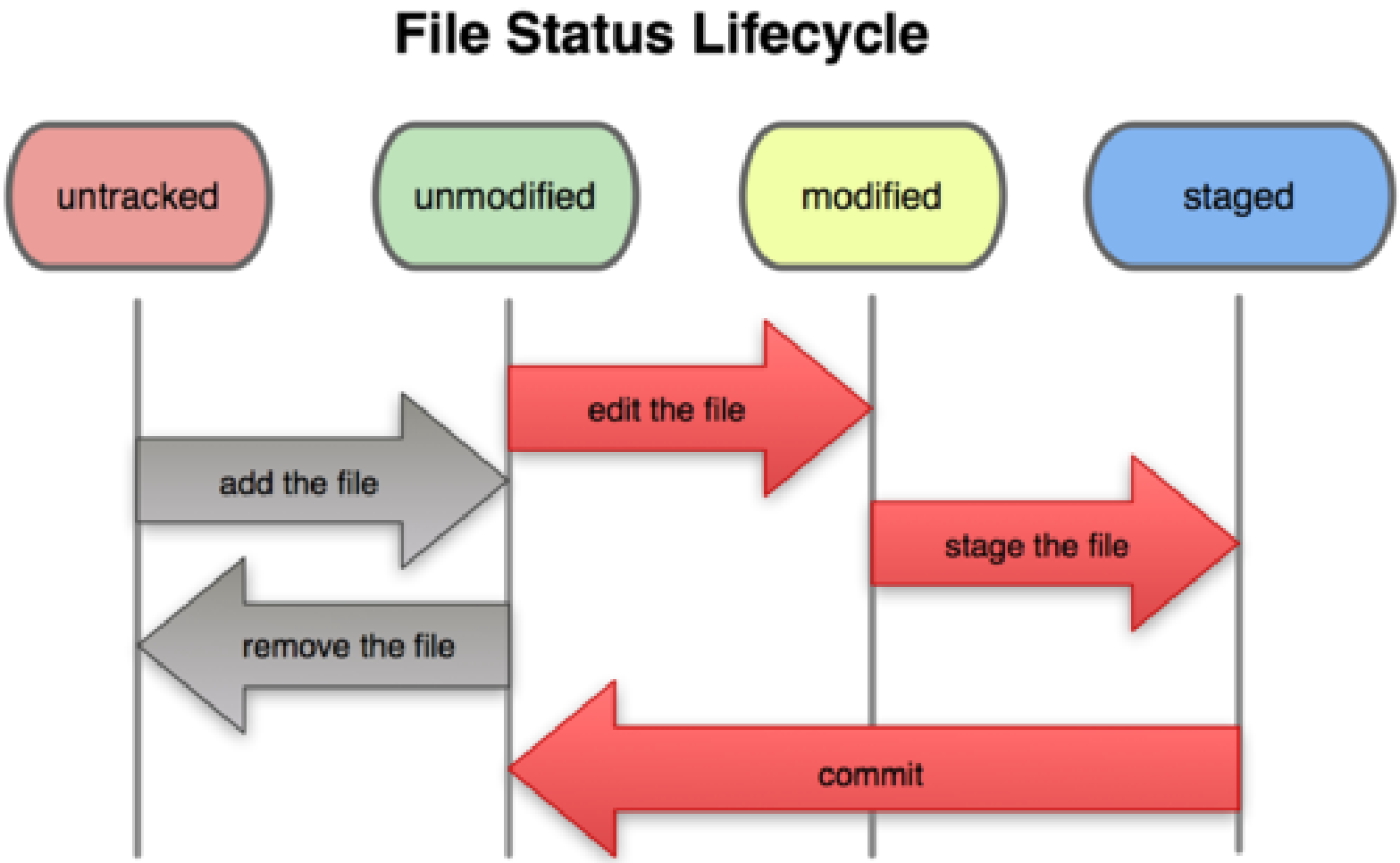
Der Index ist sehr mächtig und Alleinstellungsmerkmal von git

Der Index / Stage



Typische Arbeitsweise mit git (links).
Workflow wie bei SVN (rechts) auch möglich.

Dateizustände



Anfangs evtl. verwirrend, `git status` zeigt immer Hilfe an.

Praxis

Befehle, Dateien, Programme

Wichtige git-Befehle - Arbeiten

- `git help <command>`
  Hilfe zu git-commands, wichtigster Befehl ;)
- `git init`
  Ein git-Repo im aktuellen Verzeichnis beginnen
- `git add`
  Datei dem Index hinzufügen
- `git status`
  Status der Dateien im Arbeitsverzeichnis
- `git diff`
  Aktuelle Änderungen als diff anzeigen
- `git commit`
  Commit erstellen

Wichtige git-Befehle - History

- `git log`
  Log (History) anzeigen
- `git branch`
  Branches anzeigen (aktueller Branch hervorgehoben)
- `git checkout <branch>`
  Branch auschecken
- `git show <object>`
  Objekt (Branch, Commit, HEAD) anzeigen
- `git blame - <file>`
  Anzeigen wer zuletzt welche Zeile geändert hat

Wichtige git-Befehle - Kollaboration

- `git branch -a`
  Alle Branches anzeigen
- `git checkout -b <new_branch> <old_branch>`
  Branch erzeugen und auschecken
- `git pull` bzw. `git fetch`
  Aktualisierung des Repositories vom Server
- `git push`
  Aktualisierung des Repositories auf den Server
- `git merge <branch>`
  Einbinden eines anderen Branches
- `git remote`
  Anzeigen von eingetragenen Servern

git revisions

Für Befehle wie `git log`, `git diff`, `git checkout`, `git show` etc.

- `dae86e1950b1277e545cee180551750029cfe735` oder `dae86e` (falls eindeutig)
  SHA1 des Commits
- `master` oder `origin/master`
  Branch-Namen (References)
- `master~5`
  Der fünfte Vorgänger des letzten Commits auf master
- `master@{yesterday}`
  Zustand von master gestern

Siehe auch `git help rev-parse` für weitere Optionen

git revisions

Für Befehle wie `git log, git diff` etc.

- `master..testing`
  Änderungen in testing aber nicht in master
- `master..`
  Änderungen im aktuellen Branch die nicht in master sind
- `..master`
  Änderungen die in master aber nicht im aktuellen Branch sind
- `master...testing`
  Symmetrische Differenz

Siehe auch `git help rev-parse` für weitere Optionen

Wichtige git-Dateien

- `$HOME/.gitconfig`
  git Benutzerkonfiguration
- `.gitignore`
  Zum Ignorieren bestimmter Dateien bzw. Dateitypen, Verzeichnisweise und rekursiv
- `.git`
  git Repository, nur *einmal*, im obersten Verzeichnis
- `.git/config`
  Konfiguration auf Repository-Ebene, z.B. Remotes
- `.git/objects`
  Objektdatenbank (Dateien, Commits, Tags)
- `.git/refs`
  Branches (References)

Wichtige git-Dateien - Branches

Namespaces für refs (”references”) in `.git`:

- `.git/refs/heads`
  Lokale Branches
- `.git/refs/remotes`
  Remote Branches
- `.git/refs/tags`
  Tags (Versionsnummern)

Gearbeitet wird *immer* auf lokalen Branches, Remote Branches werden nur von `git pull` bzw. `git fetch` geändert.
Commits ohne Branch gehen früher oder später verloren!

git Tutorial

Initiale Konfiguration:

```
git config --global user.name "Johannes Gilger"
git config --global user.email "gilger@rz.de"
```

Eventuell noch:

```
git config --global color.ui "auto"
git config --global color.status.changed "yellow"
```

Das erste Repository

```
[0]jojo@macbook:~$ mkdir repo
[0]jojo@macbook:~$ cd repo
[0]jojo@macbook:~/repo$ git init
Initialized empty Git repository in /Users/jojo/repo/.git/
[0]jojo@macbook:~/repo$ echo "Hello World" >> README
[0]jojo@macbook:~/repo$ git add README
[0]jojo@macbook:~/repo$ git commit -m 'Mein erster Commit'
[master (root-commit) c36d472] Mein erster Commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
[0]jojo@macbook:~/repo$ git log
 commit c36d472617681e8377a07426f69328e7e742b2b4
 Author: Johannes Gilger <heipei@hackvalue.de>
 Date:   Fri Apr 2 11:17:43 2010 +0200

     Mein erster Commit
[0]jojo@macbook:~/repo$
```

Das erste Repository

```
[0]jojo@macbook:~/repo$ echo "Datei Nummer 2" >> datei
[0]jojo@macbook:~/repo$ echo "Changes" >> README
[0]jojo@macbook:~/repo$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   README
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       datei
no changes added to commit (use "git add" and/or "git commit -a")
[0]jojo@macbook:~/repo$ git add README
[0]jojo@macbook:~/repo$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       datei
[0]jojo@macbook:~/repo$
```

Das erste Repository

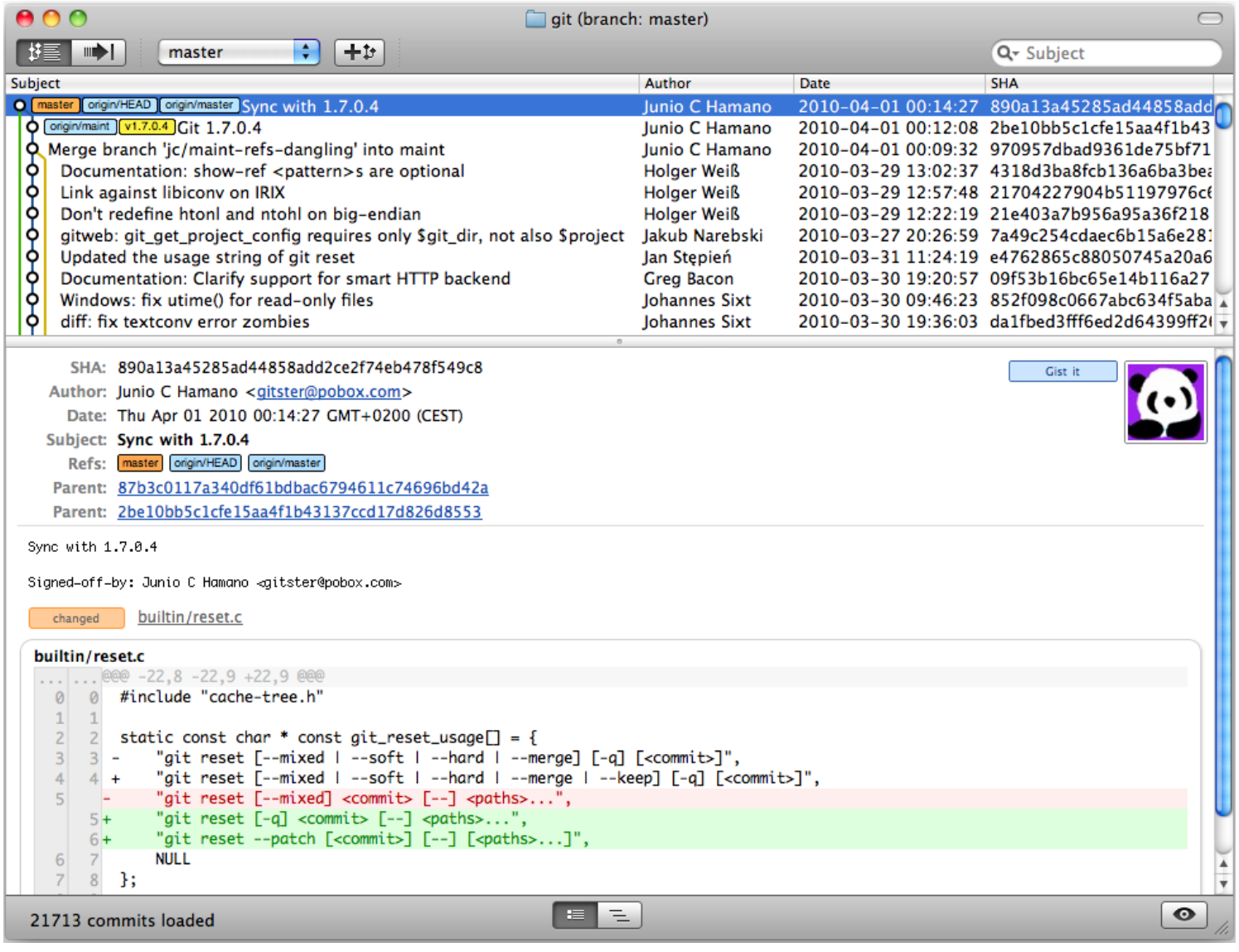
```
[0]jojo@macbook:~/repo$ git add datei
[0]jojo@macbook:~/repo$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README
#       new file:   datei
#
[0]jojo@macbook:~/repo$ git commit -m 'Zweiter Commit'
[master e6f9fa5] Zweiter Commit
 2 files changed, 2 insertions(+), 0 deletions(-)
 create mode 100644 datei
[0]jojo@macbook:~/repo$ git log
 commit e6f9fa5188351f82b60a51b1d69cef06a2026878
 Author: Johannes Gilger <heipei@hackvalue.de>
 Date:   Fri Apr 2 11:31:33 2010 +0200

     Zweiter Commit

 commit c36d472617681e8377a07426f69328e7e742b2b4
 Author: Johannes Gilger <heipei@hackvalue.de>
 Date:   Fri Apr 2 11:17:43 2010 +0200

     Mein erster Commit
[0]jojo@macbook:~/repo$
```

git Installation & Interfaces

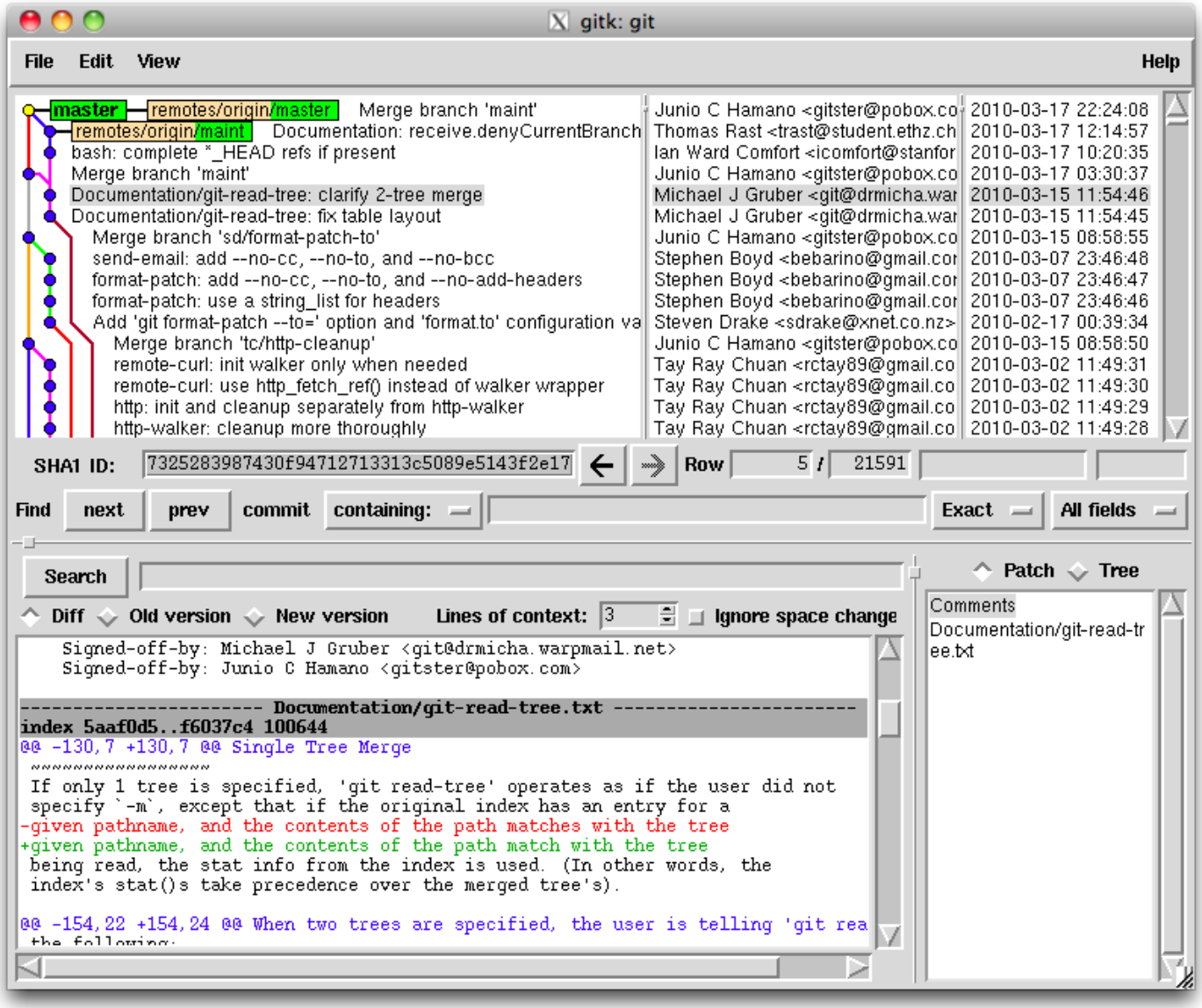
- **Linux:** Installation mittels Distribution
  Grafische Programme: `gitk` und `git gui`
- **Mac OS X:** Installation mittels fink / MacPorts oder git-osx-installer
  Grafische Programme: `gitk` oder GitX
- **Windows:** Installation per msysgit
  Grafische Programme: tortoisegit
- **Web:** GitWeb (privat), GitHub (Hosted)

$\Rightarrow$ Wie immer ist die git-Webseite der beste Anlaufpunkt
Aber: Eigentlich kein GUI nötig um mit git zu arbeiten

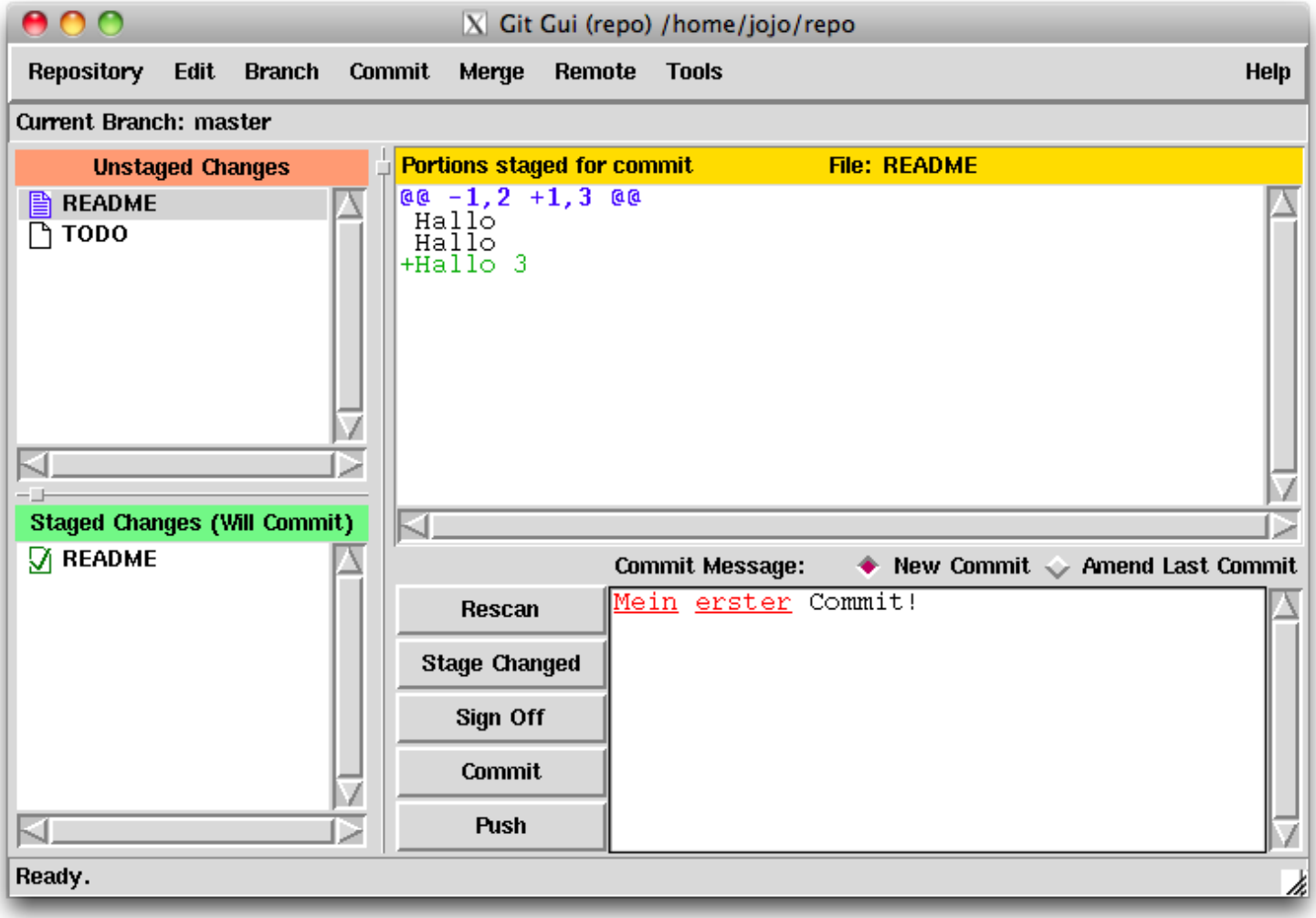
git GUIs - GitX



git GUIs - gitk

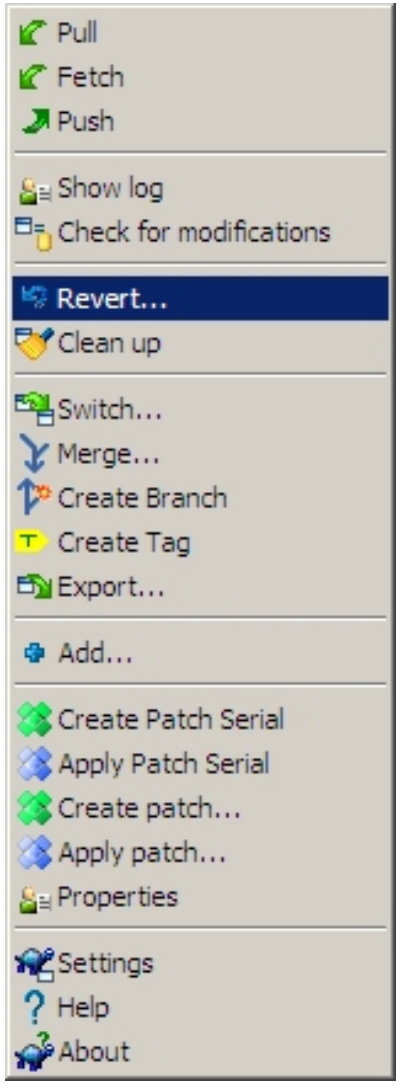


git GUIs - git gui

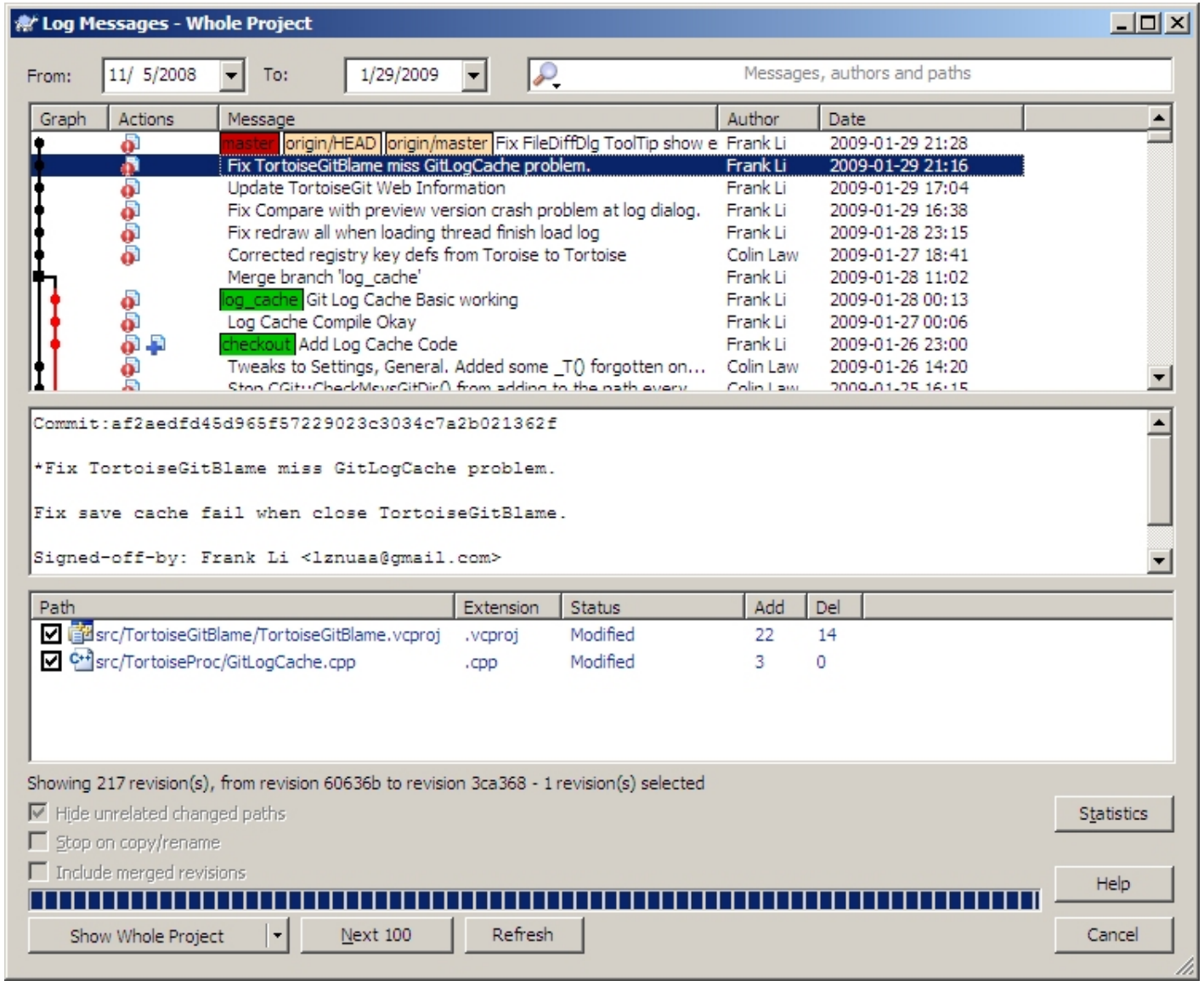


Dateien können abschnittsweise committed werden

git GUIs - tortoisegit



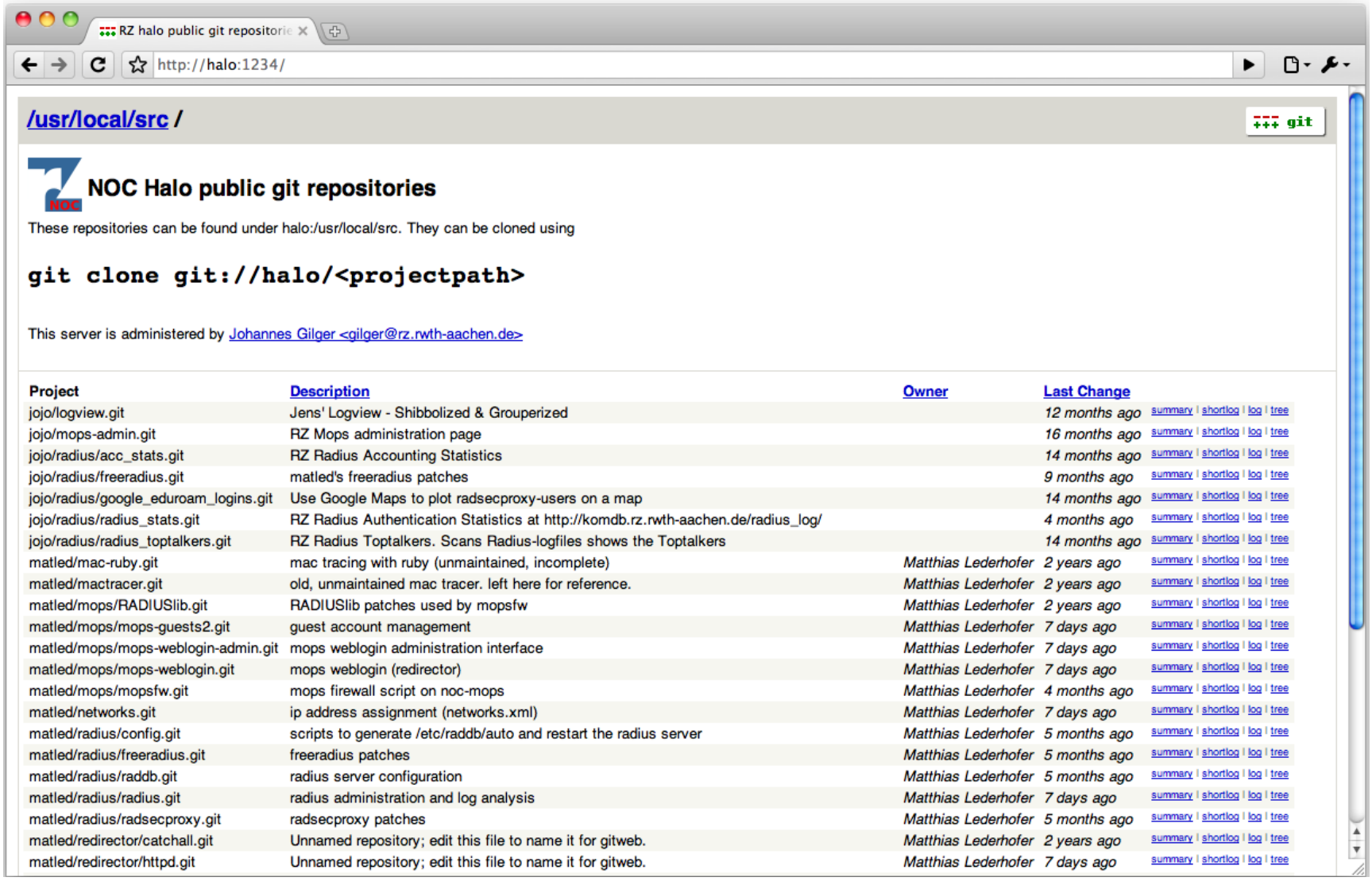
git GUIs - tortoisegit



Wie wir im NOC git benutzen

- **Szenario 1:** Entwicklung (kleinerer) Webanwendungen
  Meistens an Ort und Stelle (.git), sehr schnell und einfach
- **Szenario 2:** Änderung an externer Software festhalten
  (z.B. wenn Software von uns ”shibbolized” wird)
- **Szenario 3:** Größere Projekte mit mehreren Mitarbeitern.
  Ausgezeichnete Versionen (v0.5) direkt unterstützt

Wie wir im NOC git benutzen



Project	Description	Owner	Last Change	
jojo/logview.git	Jens' Logview - Shibbolized & Grouperized		12 months ago	summary | shortlog | log | tree
jojo/mops-admin.git	RZ Mops administration page		16 months ago	summary | shortlog | log | tree
jojo/radius/acc_stats.git	RZ Radius Accounting Statistics		14 months ago	summary | shortlog | log | tree
jojo/radius/freeradius.git	matled's freeradius patches		9 months ago	summary | shortlog | log | tree
jojo/radius/google_eduroam_logins.git	Use Google Maps to plot radsecproxy-users on a map		14 months ago	summary | shortlog | log | tree
jojo/radius/radius_stats.git	RZ Radius Authentication Statistics at http://komdb.rz.rwth-aachen.de/radius_log/		4 months ago	summary | shortlog | log | tree
jojo/radius/radius_toptalkers.git	RZ Radius Toptalkers. Scans Radius-logfiles shows the Toptalkers		14 months ago	summary | shortlog | log | tree
matled/mac-ruby.git	mac tracing with ruby (unmaintained, incomplete)	Matthias Lederhofer	2 years ago	summary | shortlog | log | tree
matled/mactracer.git	old, unmaintained mac tracer. left here for reference.	Matthias Lederhofer	2 years ago	summary | shortlog | log | tree
matled/mops/RADIUSlib.git	RADIUSlib patches used by mopsfw	Matthias Lederhofer	2 years ago	summary | shortlog | log | tree
matled/mops/mops-guests2.git	guest account management	Matthias Lederhofer	7 days ago	summary | shortlog | log | tree
matled/mops/mops-weblogin-admin.git	mops weblogin administration interface	Matthias Lederhofer	7 days ago	summary | shortlog | log | tree
matled/mops/mops-weblogin.git	mops weblogin (redirector)	Matthias Lederhofer	7 days ago	summary | shortlog | log | tree
matled/mops/mopsfw.git	mops firewall script on noc-mops	Matthias Lederhofer	4 months ago	summary | shortlog | log | tree
matled/networks.git	ip address assignment (networks.xml)	Matthias Lederhofer	7 days ago	summary | shortlog | log | tree
matled/radius/config.git	scripts to generate /etc/raddb/auto and restart the radius server	Matthias Lederhofer	5 months ago	summary | shortlog | log | tree
matled/radius/freeradius.git	freeradius patches	Matthias Lederhofer	5 months ago	summary | shortlog | log | tree
matled/radius/raddb.git	radius server configuration	Matthias Lederhofer	5 months ago	summary | shortlog | log | tree
matled/radius/radius.git	radius administration and log analysis	Matthias Lederhofer	7 days ago	summary | shortlog | log | tree
matled/radius/radsecproxy.git	radsecproxy patches	Matthias Lederhofer	5 months ago	summary | shortlog | log | tree
matled/redirector/catchall.git	Unnamed repository; edit this file to name it for gitweb.	Matthias Lederhofer	2 years ago	summary | shortlog | log | tree
matled/redirector/httpd.git	Unnamed repository; edit this file to name it for gitweb.	Matthias Lederhofer	7 days ago	summary | shortlog | log | tree

Wie wir im NOC git benutzen

/usr/local/src / matled/ticket.git / commitdiff

summary | shortlog | log | commit | commitdiff | tree
raw (parent: fa9bc16)

fix error for unauthorized users master

Matthias Lederhofer [Thu, 25 Mar 2010 13:08:46 +0000 (14:08 +0100)]

app/controllers/application_controller.rb	patch | blob | blame | history
app/views/layouts/application.rhtml	patch | blob | blame | history

```
diff --git a/app/controllers/application_controller.rb b/app/controllers/application_controller.rb
index 3af2415..5f02b42 100644 (file)
--- a/app/controllers/application_controller.rb
+++ b/app/controllers/application_controller.rb
@@ -27,7 +27,7 @@ class ApplicationController < ActionController::Base
     def load_login
         @login = User.login(@shibboleth.username, session[:admin] && is_admin?)

-        # XXX: create admin users automatically on first login
+        # create admin users automatically on first login
         if !@login && is_admin?
           User.new(:username => @shibboleth.username, :name => @shibboleth.name).save!
           @login = User.login(@shibboleth.username, session[:admin] && is_admin?)
diff --git a/app/views/layouts/application.rhtml b/app/views/layouts/application.rhtml
index aeea145..726e803 100644 (file)
--- a/app/views/layouts/application.rhtml
+++ b/app/views/layouts/application.rhtml
@@ -26,6 +26,7 @@
 <% if @is_admin %>
 <li><%= link_to 'toggle admin', :controller => '/login', :action => 'toggleadmin', :url => url_for %></li>
 <% end %>
+<% if @login %>
 <% if @login.superuser? %>
 <li><%= link_to 'Status', :controller => '/admin/statuses' %></li>
 <li><%= link_to 'Priority', :controller => '/admin/priorities' %></li>
```

Benutzung von git lernen

1. **git installieren**
   Linux: Package-Manager
2. **Kopie von bisherigem Code in neues Verzeichnis**
   Dann ein `git init` und weiterentwickeln
3. **Regelmäßig Doku / manpages lesen**
4. **Alle Befehle ausprobieren**
   Insbesondere destruktive Aktionen und Recovery
5. **Im Zweifelsfall vorher das ganze Verzeichnis kopieren**
   Als Backup wenn man sich noch nicht ganz sicher ist
6. **Dateien in `.git` anschauen**
   Was sind Objekte? Was sind Branches?

Literatur zu git

- **Pro Git** - http://progit.org/book/
  Sehr ausführliches Online-Buch das alle Themen behandelt
- **”Getting Git”** -
  http://www.gitcasts.com/git-talk
  Guter Vortrag (mit Folien) zur Benutzung und Datenstrukturen
- **git ref** - http://gitref.org/
  Die wichtigsten git-Befehle kurz erklärt
- **git - SVN crash course** -
  http://git-scm.com/course/svn.html
  Einführung in git mit Vergleichen zu svn
- **Git - kurz & gut** - O’Reilly, ISBN: 978-3-89721-914-4
  Gute und umfassende Einführung in Deutsch (€9,90)

The End

Fragerunde!

Folgefragen können gerne an gilger@rz.rwth-aachen.de und lederhofer@rz.rwth-aachen.de gerichtet werden.
Oder einfach vorbei schauen ;)