

”global information tracker”: you’re in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.

”goddamn idiotic truckload of sh*t”: when it breaks
(/usr/share/doc/git-1.5.3.3/README)

GIT

The stupid content tracker

Florian Westphal

29. November 2007

1024D/F260502D <fw@strlen.de>

1C81 1AD5 EA8F 3047 7555

E8EE 5E2F DA6C F260 502D

- GIT -

Themenübersicht

- Versionskontrolle Allgemein
- Linux Kernel Entwicklungsmodell
- GIT
 - Architektur
 - Grundlagen
 - Features
 - Verteiltes Arbeiten
 - Verschiedenes
- GIT mit anderen SCM nutzen

The Git logo, consisting of three red dashes above the text 'git' in a green monospace font, all contained within a thin black rectangular border.

Versionskontrolle

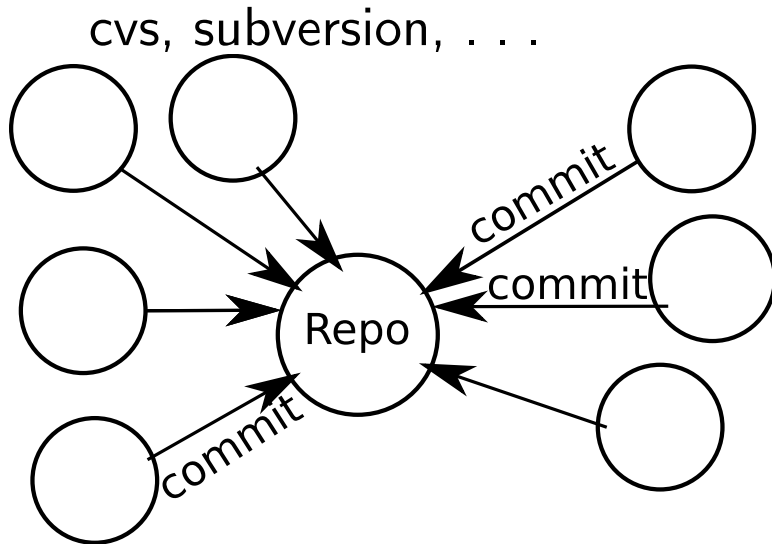
- erfasst den Entwicklungsablauf
- → Reproduzierbarkeit
 - Welchen Zustand hatte Projekt zum Zeitpunkt X?
 - Wer hat welche Änderungen gemacht?
 - Warum wurde Änderung gemacht?
- Zusammenarbeit mehrerer Entwickler
- Und nicht zuletzt: Ermittlung des Schuldigen ;-)

Buzz-Bingo

- Repository: Speichert den Verlauf des Projekts.
- checkout
- commit: hält Änderungen fest; neuer Projektzustand
- branch



Zentrales Repository

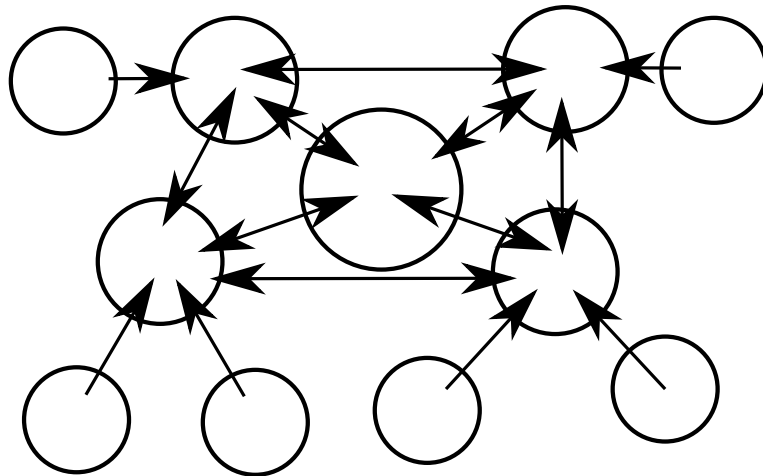


- n Entwickler schreiben in gemeinsam genutztes Repository
 - Koordination: Jeder arbeitet (praktisch gesehen) lokal
 - Bei Check-in ggf. Konfliktbereinigung nötig
 - Politik: Wer erhält Schreibzugang?

Linux Kernel Entwicklungsmodell

- ein paar hundert Entwickler
- Austausch von Patchsets über Mailingliste
- "Linus doesn't scale"-Problem
- Lösung: Dezentralisierung

GIT: Dezentral



Jeder Entwickler hat eigenes Repository

- Entwickler publizieren ihre Änderungen
- Andere können diese übernehmen

Linux: Parallele Entwicklung einzelner Subsysteme/Architekturen/. . .

+++ git

Historisches

- Entwicklung ab April 2005 als Neues Linux Kernel SCM
- Heute nutzen auch div. andere Projekte git, z.B.
 - Samba
 - freedesktop.org (cairo, mesa, X.org, . . .)
 - Wine

GIT: Kurzübersicht

- Ausgelegt auf. . .
 - Verteilte Entwicklung, schnelles branching/merging
 - Großes Projekt mit vielen kleinen Änderungen
- Einfache Veröffentlichung von Repositories (http,rsync,ssh,git://)
- Set aus vielen kleinen Programmen, viele Wrapper-Scripts

```
$ git<tab>
```

```
zsh: do you wish to see all 145 possibilities (25 lines)?
```

Die gute Nachricht: zum normalen Arbeiten benötigt man nur eine Handvoll git-Kommandos. Es gibt sogar `git-gui(1)`.



```
--- git
```

plumbing vs. porcelain

GIT Kommandos können in zwei Kategorien geteilt werden:

- High-level "Porcelain":
 - das "normale" User-Interface
- Low-Level "Plumbing":
 - "Porcelain-Unterbau", wird (normalerweise) nicht direkt verwendet
 - etliche "Porcelain"-Kommandos sind Scripts, welche Plumbing-Kommandos nutzen

Git Objekte

- Identifiziert durch SHA-1 digest
- Objekttypen: tree, commit, tag, blob
- `$ git cat-file -t bb7476a`
commit

Tree

Speichert Zustand des Projekts

- Welche Dateien (und wo im Verzeichnisbaum?)
- Welcher Inhalt?

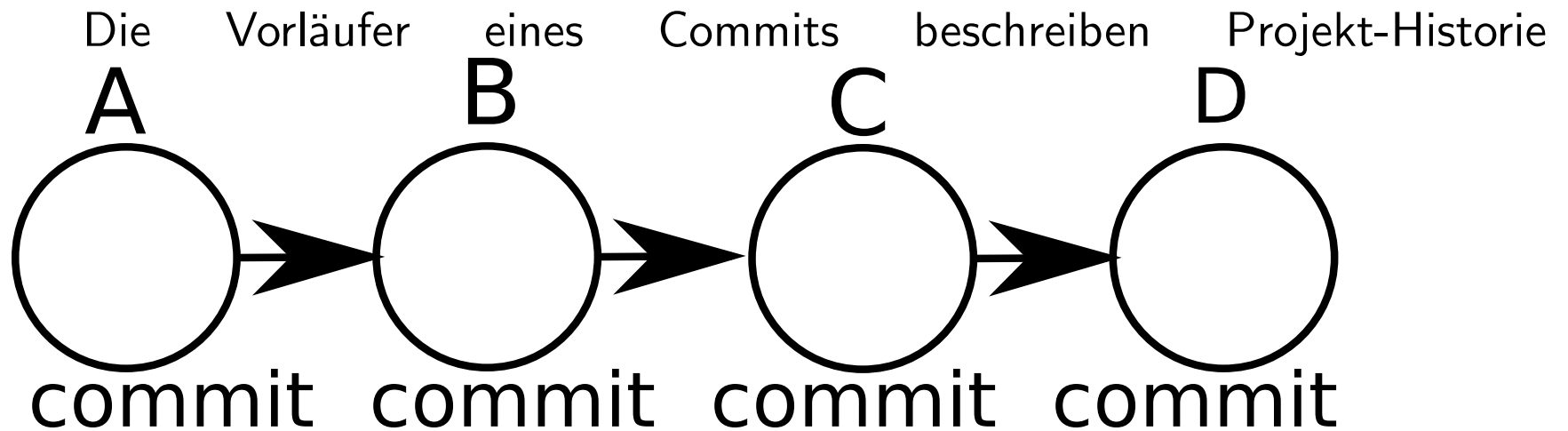


Commit

Welcher Zustand des Projekts?

- es gibt viele Trees
- Commit ist Bindeglied
 - Welcher Tree?
 - Welcher Commit ist Vorgänger?
 - Wer? Weshalb?

Commit-Graph



Repository

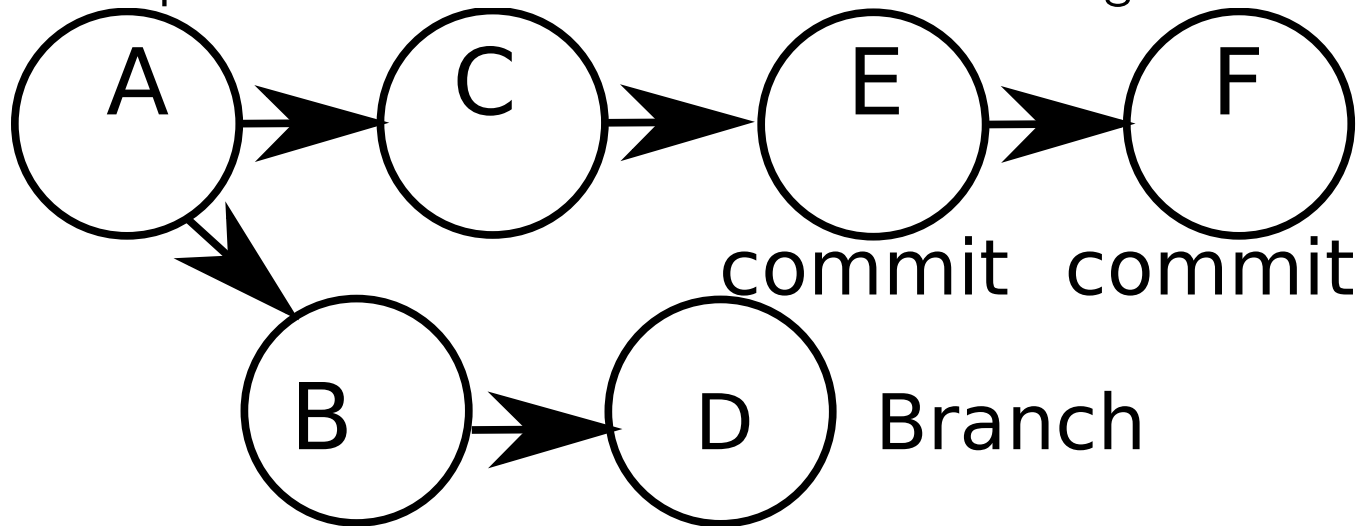
Speichert History bis zum 'jetzigen' Zustand

- alle Commits & zugehörigen Trees
- Commits im Repository bilden einen Graph



Branches

...speichern welcher Commit als letztes erfolgt ist

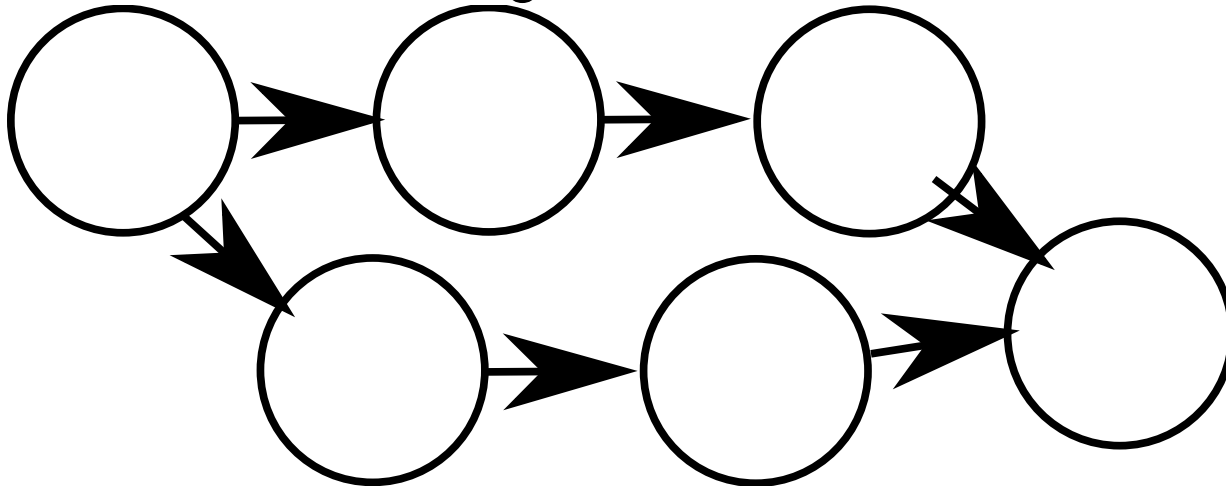


HEAD ref. aktuellen Branch

```
$ cat .git/HEAD
ref: refs/heads/master
$ cat .git/refs/heads/master
145bae8afc3fe39d5c1b58b887d15b303a5a9ebf
```

Merge

Verschiedene Zweige zusammenführen



GIT operiert auf Commit-Graph

Merge

```
git merge experimental
```

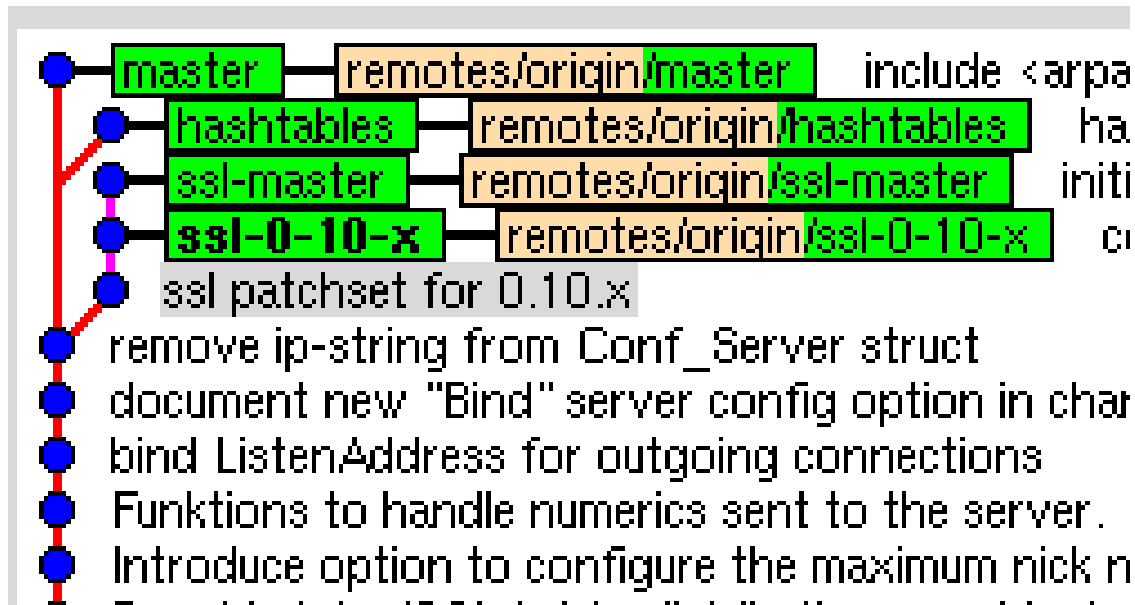
Übernimmt commits aus Zweig "experimental" in den gerade aktiven Zweig.

Nützliche Optionen:

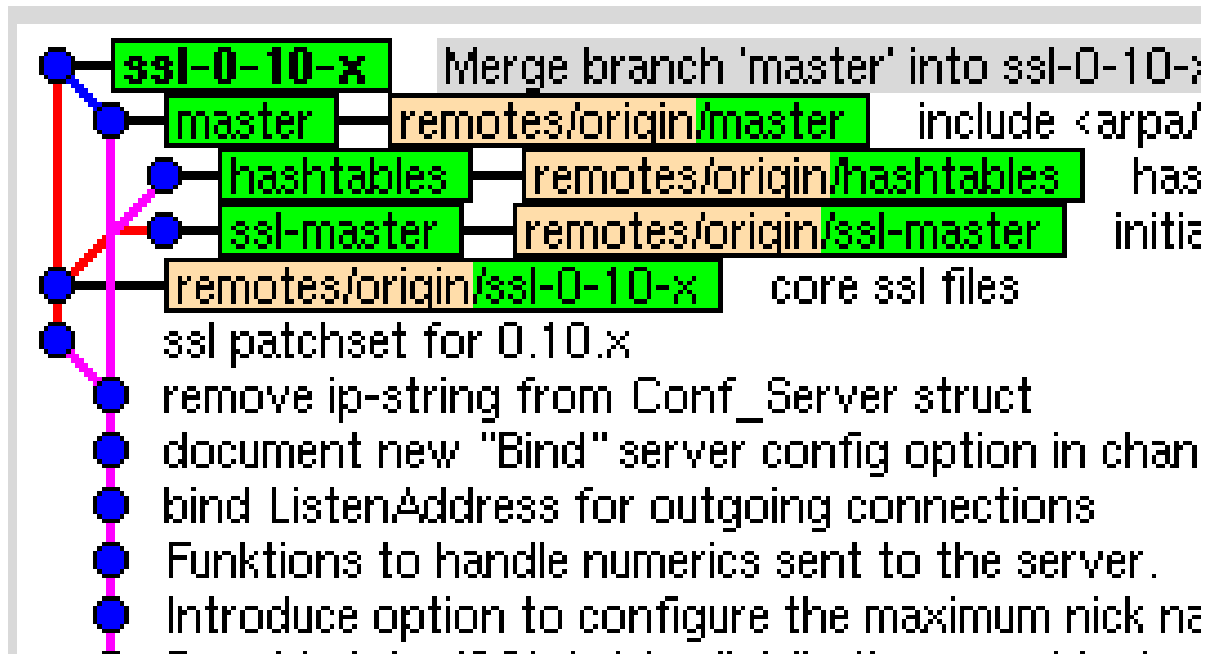
--squash: Übernimmt alle Änderungen als *einzelnes* Gesamtwerk

--no-commit: Wie-der-Name-schon-sagt

Pre-Merge



Post-Merge



Merging: Konfliktbereinigung

```
<<<<<<< HEAD:doc/sample-ngircd.conf
# SSL Trusted CA Certificates File (for verifying peer cert
;SSLCAFile = /etc/ssl/CA/cacert.pem

# Certificate Revocation File (for marking otherwise vaild
;SSLCRLFile = /etc/ssl/CA/crl.pem
=====
# SSL Server Key Certificate
;SSLCertFile = /etc/ngircd/ssl/server-cert.pem
>>>>>>> ngircd-ssl-0.1.X:doc/sample-ngircd.conf
```

Unterhalb HEAD bis =====: aktuelle Version

Unterhalb ===== bis ngircd-ssl-0.1.X (branch name): branch

Tags

Tag = "commit mit name"

```
git-tag release-0-1 e451fff
```

```
git-tag -s
```

erzeugt gnupg-signiertes Tag-Objekt

Minimale Konfiguration

```
$ git version
git version 1.5.3.4
$ git config --global user.name "Florian Westphal"
$ git config --global user.email fw@strlen.de
$ git config --global user.name
Florian Westphal
```

→ Eindeutiger Autor

Repo Erzeugen

- Neues Repo anlegen

```
$ mkdir projekt
```

```
$ cd projekt
```

```
$ git init
```

```
Initialized empty Git repository in .git/
```

- Projekt importieren:

```
git init && git add . && git commit -m initial\ import
```

Dateien hinzufügen

```
echo neue datei > blubber  
git add blubber  
git commit -m "bla blubb"
```

Änderungen einspielen

```
$ echo mehr inhalt >> blubber
$ git commit -m "tolle erweiterungen"
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   blubber
#
no changes added to commit (use "git add" and/or "git commit -a")
zsh: exit 1      git commit -m "tolle erweiterungen"
```

→ Auch Änderungen müssen erst als "bitte hinzufügen" markiert werden

git add und git commit

```
$ git add blubber
$ echo noch mehr inhalt >> blubber
$ git commit -m "inhalt und noch mehr inhalt"
$ Created commit bb7476a: inhalt und noch mehr inhalt
  1 files changed, 1 insertions(+), 0 deletions(-)
$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   blubber [..]
```

Es wird jeweils nur bis zum letzten "git add" comittet

Log, History

```
$ git log
commit e451fff70219397c330bfa9456cbe8fc2acf1f6c
Author: Florian Westphal <fw@strlen.de>
Date:   Fri Oct 26 02:18:58 2007 +0200
```

tolle erweiterungen

Wenn Changeset erwünscht ist (anstatt nur Beschreibung): -p

Log, History

Commit anzeigen:

```
$ git show e451fff70
[..]
diff --git a/blubber b/blubber
[..]
```

Weitere Kommandos zum "graben":

```
git show HEAD
git show HEAD~5
git show e451:blubber
git log branchA..branchB
git log -p -Ssuchwort
```

Verteilte Entwicklung

- existierendes Repo kopieren: `git clone`

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

- Eigene Änderungen machen
 - ggf. in extra-Branch:
 - `git checkout -b meinfeature`

Änderungen Weitergeben

Methode 1: `git-push`

- Überspielt Commits in das angegebene Repo (z.B. origin)
- meist via ssh. (`git://` ist readonly)
- Damit kann auch CVS-Ähnliches (d.h. zentrales) Modell umgesetzt werden
- man `git-receive-pack` enthält Commithook Beispiele
- `git-shell` für GIT-User welche ins Repo schreiben sollen ohne eShell-Zugriff zu erhalten

Änderungen Weitergeben

Methode 2: Patches versenden

```
git-format-patch <last-common-ancestorid>
```

erzeugt Patches der Form

```
0001-erste-zeile-aus-dem-passenden-commit-log.patch
```

Patch ist bereits mit "From:" und "Subject" versehen.

Wenn viele (zusammengehörige) Patches verschickt werden sollen, ist ein Blick auf `git-send-email` empfehlenswert.

Änderungen Weitergeben

Methode 3: `git-pull`

1. Eigenes Repo veröffentlichen, z.B. via `git-daemon` (`git://..`)
2. Andere können dann Änderungen in diesem Repo in eigenes Repo übernehmen
3. `git-request-pull` generiert Email-geeignete Changeset-Zusammenfassung

`git-pull == git-fetch + git-merge + git-commit`

Konfliktbereinigung

- Falls Entwicklung zu sehr divergiert: Merge-Konflikte, Patches "passen" nicht
- Lösung: Regelmässig pull von 'Upstream'
 1. Merge funktioniert: alles ok, weiterarbeiten (am besten Merge rückgängig machen; hält History "sauber")
 2. Merge funktioniert nicht
 - (a) Nun entweder Merge händisch fertigstellen (wie gehabt) und weiterarbeiten *ODER*
 - (b) git-rebase

git-rebase

```
git-rebase <upstream> <meinbranch>
```

1. sichert alle Änderungen im lokalen Zweig
2. setzt lokalen Zweig zurück (`git-reset --hard <upstream>`)
3. spielt vorher gesicherte Änderungen zurück
4. Falls Merge-Konflikt: Manuell auflösen und:
 - `git rebase --continue` *ODER*:
 - `git rebase --skip` ("dieser commit ist eh schrott") *ODER*:
 - `git rebase --abort` ("alles sch...")

git-rebase

```
git-rebase <upstream> <meinbranch>
```

1. sichert alle Änderungen im lokalen Zweig
2. setzt lokalen Zweig zurück (`git-reset --hard <upstream>`)
3. spielt vorher gesicherte Änderungen zurück
4. Falls Merge-Konflikt: Manuell auflösen und:
 - `git rebase --continue` *ODER*:
 - `git rebase --skip` ("dieser commit ist eh schrott") *ODER*:
 - `git rebase --abort` ("alles sch...")

Achtung: `git-rebase` verändert Commit-History



Git stash

`git-stash`

- sichert den aktuellen Zustand (incl. nicht-comitteter Änderungen)
- danach kann dann z.B. ein anderer Branch ausgecheckt werden ('kannst du dir mal schnell xyz angucken?')
- anschliessend kann alter Zustand mit `git stash apply` wieder hergestellt werden
- Wie viele andere git-Kommandos ist auch `git-stash` "nur" ein Script

Bisect

`git-bisect`

aka "Feature X funktioniert nicht mehr (und gestern liefs noch!)"

- Ermittle den "schuldigen" commit im Commit-Graph
- Funktioniert auch bei nicht-linearem Graph
- Stellt Projektzustand zwischem funktionierendem und nicht-funktionierendem Tree her
- Testen: Funktionierte es bei dieser Version?
- Commit-Graph wird immer weiter eingeschränkt bis der verantwortliche Commit gefunden ist

Interaktion mit anderen SCM

- CVS

- `git-cvsmimport -d<CVSRROOT> cvsmodule`
- `git → CVS: git-cvsexportcommit`
`cd path/to/cvs; export GIT_DIR=path/to/git/.git`
`git-cvsexportcommit <id>`

- subversion

- `git-svn clone svn://....`
- `git → svn git commit; git-svn dcommit`

Weiterführende Lektüre

- <http://www.kernel.org/pub/software/scm/git/docs/user-manual.html>
- <http://www.kernel.org/pub/software/scm/git/docs/everyday.html>