

# Sourcecode Management mit GIT

Clifford Wolf

ROCK Linux - <http://www.rocklinux.org/>

Csync2 - <http://oss.linbit.com/csync2/>

STFL - <http://www.clifford.at/stfl/>

SPL - <http://www.clifford.at/spl/>



## Einleitung

- Inhalt
- Nicht-Inhalt
- Pitfalls
- Benefits
- Kommandos
- Dokumentation
- Arbeitsumgebung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

# Einleitung



# Inhalt

## Einleitung

### ● Inhalt

- Nicht-Inhalt
- Pitfalls
- Benefits
- Kommandos
- Dokumentation
- Arbeitsumgebung

## Repository

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Einleitung
- GIT Datenstrukturen (Repository)
- GIT Workflow (committen, branchen, mergen, etc.)
- Kollaboration (Pull, Push, Patches, etc.)
- GIT Server Administration
- History Manipulation (rebase, filter-branch)
- Subversion Integration
- Programmier-Interfaces



# Nicht-Inhalt

## Einleitung

- Inhalt
- Nicht-Inhalt
- Pitfalls
- Benefits
- Kommandos
- Dokumentation
- Arbeitsumgebung

## Repository

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Kommando-Referenz (es sind über 150 Kommandos!)
- Integration mit CVS / Quilt / Arch
- Step-by-Step Migration zu GIT



# Pitfalls

## Einleitung

- Inhalt
- Nicht-Inhalt
- Pitfalls
- Benefits
- Kommandos
- Dokumentation
- Arbeitsumgebung

## Repository

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Wie jedes umfangreiche Projekt hat auch GIT seine eigene Nomenklatur entwickelt.
- Leider ist diese jedoch nicht eindeutig.  
So wurde der "index" z. Bsp. früher "cache" genannt.
- GIT beinhaltet Low-level-Tools zum Modifizieren der GIT-Datenstrukturen sowie High-level-Wrapper für diese Tools.
- Daher gibt es bei GIT oft verschiedene Wege ein und dasselbe zu tun.
- Bei der Benennung der Tools und der Optionen macht sich eine "gewachsene Struktur" bemerkbar.
- Anders als z. Bsp. bei Subversion ist es bei GIT möglich, einmal committete Daten nachträglich zu verändern oder auch wieder zu löschen.



# Benefits

## Einleitung

- Inhalt
- Nicht-Inhalt
- Pitfalls
- Benefits
- Kommandos
- Dokumentation
- Arbeitsumgebung

## Repository

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Dezentraler Ansatz:
  - ◆ Arbeiten von unterwegs
  - ◆ Lokale Branches zum Rumbprobieren
  - ◆ Lokale Branches für lokale Modifikationen
  - ◆ Keine Commit-Rechte notwendig
- Geschwindigkeit:
  - ◆ GIT ist unglaublich schnell
  - ◆ Dadurch ändert sich auch der persönliche Umgang mit dem Tool
- Transparenz:
  - ◆ Es ist sehr klar was wie gespeichert wird
  - ◆ Es ist sehr klar wie die Tools arbeiten
- Flexibilität:
  - ◆ Mit GIT lässt sich praktisch jeder Workflow abbilden.



# Kommandos

## Einleitung

- Inhalt
- Nicht-Inhalt
- Pitfalls
- Benefits
- Kommandos
- Dokumentation
- Arbeitsumgebung

## Repository

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Alle GIT-Kommandos beginnen mit `git-`.  
Z.Bsp.: `git-init`, `git-checkout` und `git-rev-parse`
- Es gibt auch den Wrapper `git` zum Starten dieser Kommandos.  
Z.Bsp.: `git init`, `git checkout` und `git rev-parse`
- Es wird zwischen *plumbing* (Low-Level) und *porcelain* (High-Level) Kommandos unterschieden.
- Kommandoübersicht: `man git`



# Dokumentation

## Einleitung

- Inhalt
- Nicht-Inhalt
- Pitfalls
- Benefits
- Kommandos
- Dokumentation
- Arbeitsumgebung

## Repository

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Jedes Kommando hat eine Manpage.  
Z.Bsp.: `man git-init`
- Einige dieser Manpages behandeln “nebenbei” auch grundlegende Konzepte von GIT.
- Das “GIT User’s Manual” beinhaltet eine Step-by-step-Einführung in GIT.
- Die Manpages und das Manual sind auch online verfügbar.
- Das Erzeugen der Dokumentation aus den GIT-Sourcen ist nur mit Hilfe einiger exotischerer Tools möglich..





# Arbeitsumgebung

## Einleitung

- Inhalt
- Nicht-Inhalt
- Pitfalls
- Benefits
- Kommandos
- Dokumentation
- **Arbeitsumgebung**

## Repository

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Jede GIT-Anwenderin arbeitet mit einem lokalen GIT-Repository.
- So ein GIT-Repository ist ein Directory, in dem eine Working Copy (die “ausgecheckten Sourcen”) liegt, und das zusätzlich die gesamte Projekt-History plus einigen Metadaten im Unterverzeichnis `.git` beinhaltet.
- Ein neues GIT-Repository wird mit dem Kommando `git-init` erstellt.
- Eine lokale Kopie eines bestehenden Repositorys wird mit dem Kommando `git-clone` erstellt.
- Alle anderen GIT-Kommandos werden in einem bestehenden Repository gestartet.



Einleitung

**Repository**

- Einleitung
- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs
- Config
- Global Config
- Repository-Administration
- Repository Browsing

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

# Repository



# Einleitung

## Einleitung

### Repository

#### ● Einleitung

- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs
- Config
- Global Config
- Repository-Administration
- Repository Browsing

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Ein GIT-Repository beinhaltet die vollständige Geschichte des Projekts.
- Das bedeutet:
  - ◆ Jede Version des Sourcecodes die zum derzeitigen Stand geführt hat.
  - ◆ Die Metainformation welche Version von welcher anderen Version abstammt.
- GIT verwaltet dabei das Projekt als ganzes, und nicht einzelne Dateien.
- Es ist also z. Bsp. nicht direkt möglich zu sehen in welchen Versionen eine bestimmte Datei verändert wurde.
- Branches, die verworfen wurden, werden von GIT nicht weiter beachtet und können auch gelöscht werden.



# Objekte

Einleitung

Repository

- Einleitung
- **Objekte**
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs
- Config
- Global Config
- Repository-Administration
- Repository Browsing

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- GIT verwaltet “Objekte”, die jeweils auf andere Objekte verweisen können.
- Es gibt folgende Objekttypen:
  - ◆ Blobs: Dateiinhalte ohne weitere Metadaten
  - ◆ Trees: entsprechen Verzeichnissen, verweisen auf Blobs für die Dateien und auf weitere Trees für Subdirectorys
  - ◆ Commits: Verweisen auf Vorgänger-Commits sowie auf das entsprechende Tree-Objekt
  - ◆ Tags: Verweisen auf Commits und beinhalten optional eine PGP/GPG Signatur
- Außerhalb dieses Objektbaumes gibt es “Referenzen”, die auf jene Objekte zeigen, die als “Einsprungspunkte” in diesen Graphen dienen.



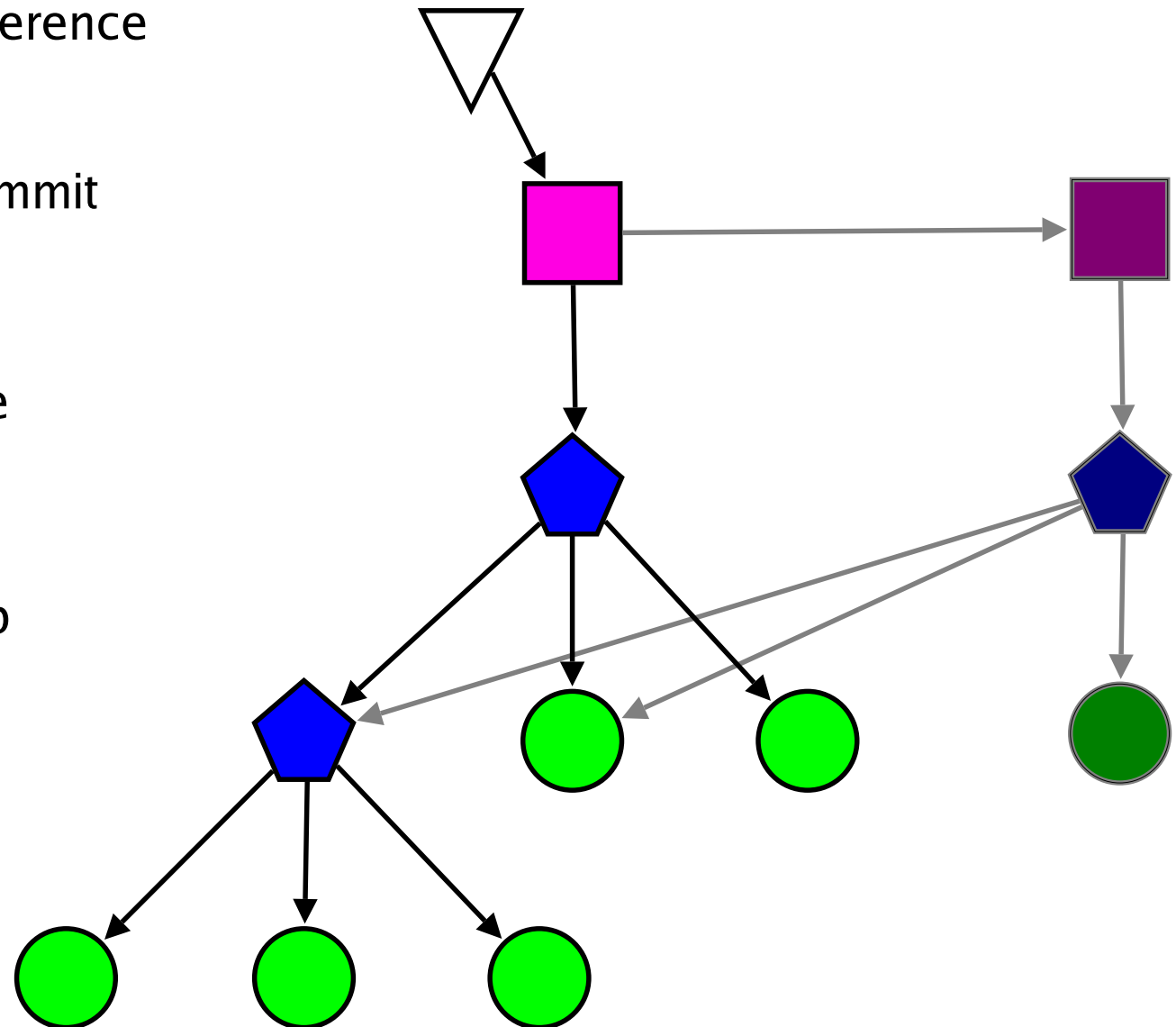
# Object Tree

▽ Reference

■ Commit

◆ Tree

● Blop



Einleitung

Repository

● Einleitung

● Objekte

● Object Tree

● SHA-1 Prüfsummen

● Referenzen (1/2)

● Referenzen (2/2)

● Referenz Lookups

● Reflogs

● Config

● Global Config

● Repository-Administration

● Repository Browsing

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs



# SHA-1 Prüfsummen

## Einleitung

## Repository

- Einleitung
- Objekte
- Object Tree
- **SHA-1 Prüfsummen**
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs
- Config
- Global Config
- Repository-Administration
- Repository Browsing

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Als Bezeichner für GIT-Objekte dienen SHA-1 Prüfsummen über Objekttyp und Inhalt.
- SHA-1 garantiert praktisch Kollisionssicherheit (es gibt  $\approx 1,4615 \cdot 10^{48}$  mögliche SHA-1 Prüfsummen).
- Die Prüfsumme wird bei jedem Laden eines Objekts überprüft. Damit ist ein Objektbaum **eindeutig** und **fälschungssicher** über eine einzelne SHA-1 Prüfsumme ansprechbar.
- Zwei Objekte gleichen Inhalts erhalten automatisch die gleiche Objektbezeichnung.



# Referenzen (1/2)

## Einleitung

## Repository

- Einleitung
- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs
- Config
- Global Config
- Repository-Administration
- Repository Browsing

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- GIT-Objekte können direkt über ihren SHA-1 Key angesprochen werden.
- Die ersten paar Zeichen des SHA-1 Keys reichen ebenfalls aus, wenn sie eindeutig sind.
- Darüber hinaus können beliebige Dateinamen im `.git` Directory verwendet werden, die den entsprechenden Key beinhalten. Insbesondere:

HEAD

ORIG\_HEAD

`refs/heads/branch`

`refs/remote/remote/branch`

`refs/tags/tag`

Der Stand der Working Copy

Der letzte Stand der Working Copy

Die lokalen Branches

Die remote Branches

Die Tags



# Referenzen (2/2)

## Einleitung

## Repository

- Einleitung
- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- **Referenzen (2/2)**
- Referenz Lookups
- Reflogs
- Config
- Global Config
- Repository-Administration
- Repository Browsing

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Wenn der Dateiname relativ zu `.git` nicht existiert werden weitere Directories durchsucht:
  - ◆ `.git/refs/name`
  - ◆ `.git/refs/tags/name`
  - ◆ `.git/refs/heads/name`
  - ◆ `.git/refs/remotes/name`
  - ◆ `.git/refs/remotes/name/HEAD`
- Darüber hinaus gibt es von jeder dieser "Referenz-Dateien" Logfiles, so dass es immer auch möglich ist, alte Versionen wiederherzustellen.
- Alle diese Dateien sind einfache Textfiles.
- Objekte können zusätzlich auch über den "Index" referenziert werden.





# Referenz Lookups

Einleitung

Repository

- Einleitung
- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs
- Config
- Global Config
- Repository-Administration
- Repository Browsing

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Referenz-Namen können auch spezielle Zeichen für weitere Lookups beinhalten.
- Beispiele:

$name^$	Der Parent des Commits
$name^n$	Der $n$ te Parent des Commits (bei Merges)
$name^{^^}$	Der Parent des Parents
$name\sim 3$	Der Parent des Parents des Parents
$name@\{1\}$	Der Wert vor der letzten Änderung (Log)
$name@\{2.hours\}$	Der Wert vor zwei Stunden (aus dem Log)
- Die meisten Tools erzeugen automatisch sinnvolle Namen mit  $^$  und  $\sim$  bei der Ausgabe.
- Dokumentation zur vollständigen Syntax:  
`man git-rev-parse`



# Reflogs

## Einleitung

### Repository

- Einleitung
- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- **Reflogs**
- Config
- Global Config
- Repository-Administration
- Repository Browsing

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Jede Änderung einer Referenz wird in einem Logfile gespeichert,
- dazu der Zeitpunkt sowie eine Log-Message.
- Das ermöglicht zum Beispiel ein einfaches Zurücksetzen auf einen alten Wert.
- Die Log-Files liegen im Directory `.git/logs`.
- Es sind einfache Textdateien mit einer Zeile pro Log-Message.
- Tool zum Arbeiten mit Reflogs: `git-reflog`



# Config

## Einleitung

## Repository

- Einleitung
- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs

## ● Config

- Global Config
- Repository-Administration
- Repository Browsing

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Jedes Repository hat ein `.ini`-style Config-File (`.git/config`).
- Globale Einstellungen können in `~/.gitconfig` eingetragen werden.
- Im Config-File kann das Verhalten der GIT-Tools parametrisiert werden, und hier kann auch der Workflow eingestellt werden.
- Zum Beispiel werden im Config-File die “Branch-Pfade” eingestellt, also festgelegt, welcher Branch von welchen anderen Branches merged.
- Dokumentation: `man git-config`



# Global Config

## Einleitung

## Repository

- Einleitung
- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs
- Config

## Global Config

- Repository-Administration
- Repository Browsing

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Globale Einstellungen können in `~/.gitconfig` gespeichert werden.

- Für die globale config hat `git-config` die Option `--global`.

- Beispiele:

```
git-config --global user.email "clifford@clifford."
```

```
git-config --global user.name "Clifford Wolf"
```

```
git-config --global alias.co "checkout"
```

```
git-config --global http.proxy "http://proxy:port"
```



# Repository-Administration

## Einleitung

## Repository

- Einleitung
- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs
- Config
- Global Config
- Repository-Administration
- Repository Browsing

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Das Kommando `git-gc` muss regelmäßig gestartet werden.
- In der Regel geschieht dies automatisch durch andere GIT Kommandos.
- Das Kommando `git-gc` führt folgende Operation aus:
  - ◆ Kompression der Objekt-Datenbank
  - ◆ Expiren alter Reflog Einträge
  - ◆ Löschen alter unreferenzierter Objekte
- Das Kommando `git-prune` löscht alle unreferenzierten Objekte.
- Dokumentation: `man git-gc`, `man git-prune`



# Repository Browsing

## Einleitung

### Repository

- Einleitung
- Objekte
- Object Tree
- SHA-1 Prüfsummen
- Referenzen (1/2)
- Referenzen (2/2)
- Referenz Lookups
- Reflogs
- Config
- Global Config
- Repository-Administration
- Repository Browsing

## Working Copy und Index

## Branchen und Mergen

## Kollaboration

## GIT Server

## Rewriting History

## Subversion Integration

## Eigene Tools

## Weitere Tools

## URLs

- Typ eines Objekts bestimmen: `git-cat-file -t`
- Blop/Commit/Tag anzeigen: `git-cat-file`
- Tree anzeigen: `git-ls-tree`
- High-Level Tool: `git-show`
- Liste aller Referenzen: `git-show-ref`
- Commit-Graph anzeigen: `git-show-branch`
- Grafischer Repository Browser: `gitk`



Einleitung

Repository

**Working Copy und Index**

- Einleitung
- Workcycle
- Weitere Kommandos

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

# Working Copy und Index



# Einleitung

Einleitung

Repository

Working Copy und Index

● Einleitung

● Workcycle

● Weitere Kommandos

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Für ein neues Commit benötigt man ein (neues) Top-level-tree-Objekt.
- Dieses neue Top-level-tree-Objekt wird im “Index” vorbereitet.
- Dieser “Index” ist lediglich eine Liste von (blop) Objekten und den dazugehörigen Dateinamen.
- Ein neuer Tree wird wie folgt im Index aufgebaut:
  - ◆ Der alte Zustand wird mit `git-checkout` in den Index geladen.
  - ◆ Mit `git-add` werden neue blop-Objekte erzeugt und in den Index eingefügt.
  - ◆ Mit `git-rm` können Einträge aus dem Index entfernt werden.
- Schlussendlich werden mit `git-commit` ein neues Tree-Objekt und ein neues Commit-Objekt erzeugt.





# Workcycle

Einleitung

Repository

Working Copy und Index

● Einleitung

● Workcycle

● Weitere Kommandos

Branchen und Mergen

Kollaboration

GIT Server

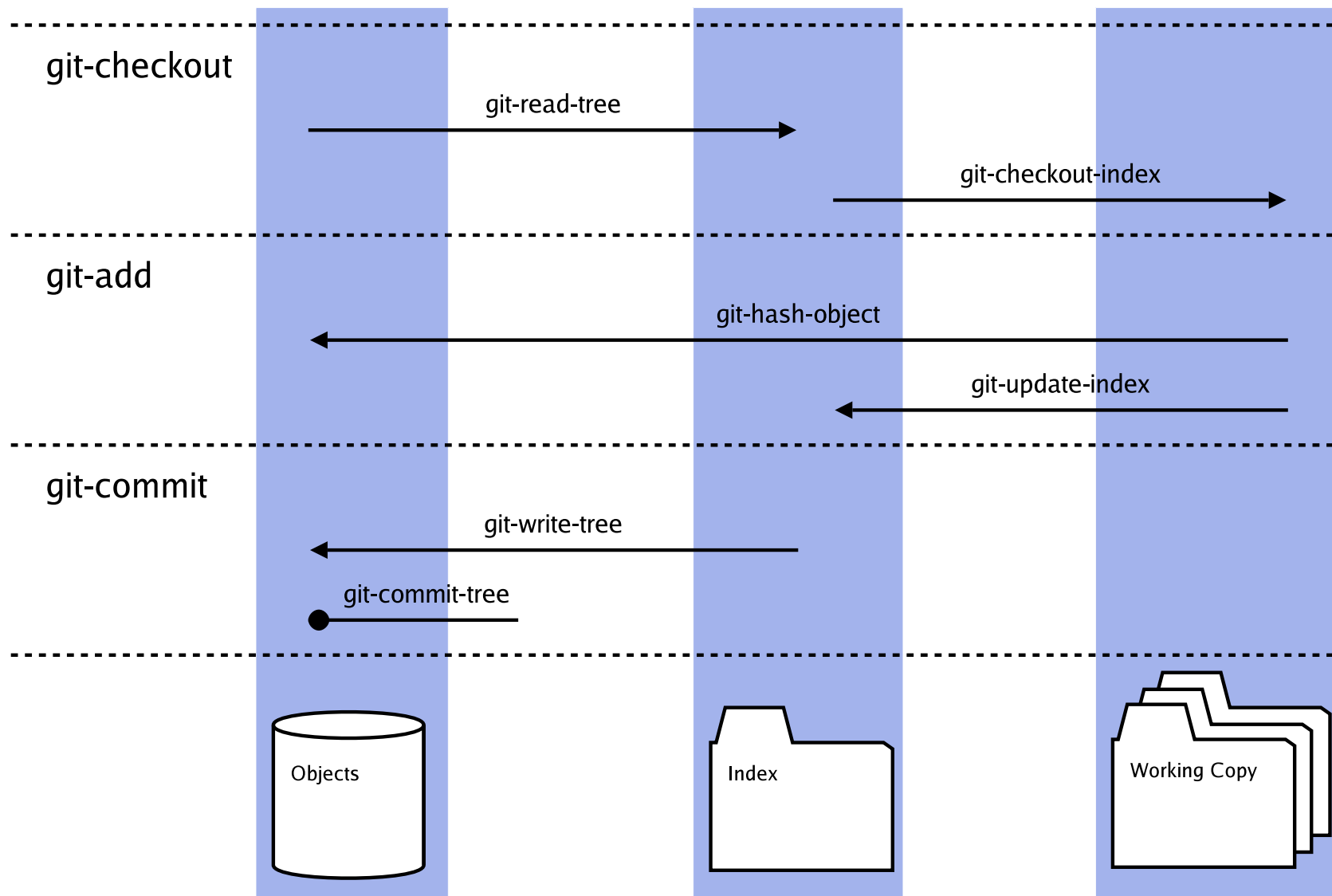
Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs





# Weitere Kommandos

Einleitung

Repository

Working Copy und Index

● Einleitung

● Workcycle

● Weitere Kommandos

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Index anzeigen: `git-ls-files --stage`
- Nur manche Änderungen zum Index: `git-add --patch`
- Low-level Index Updates: `git-update-index`
- Updaten einer Referenz: `git-update-ref`  
(Wird auch automatisch von `git-commit` gemacht.)
- Änderungen zwischen Working Tree, Index und HEAD  
Referenz: `git-status`, `git-diff`



# Branchen und Mergen

Einleitung
Repository
Working Copy und Index
<b>Branchen und Mergen</b>
● Branch (1/2)
● Branch (2/2)
● Merge (1/2)
● Merge (2/2)
● Rebase (1/2)
● Rebase (2/2)
● Konflikte
Kollaboration
GIT Server
Rewriting History
Subversion Integration
Eigene Tools
Weitere Tools
URLs



# Branch (1/2)

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

● Branch (1/2)

● Branch (2/2)

● Merge (1/2)

● Merge (2/2)

● Rebase (1/2)

● Rebase (2/2)

● Konflikte

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

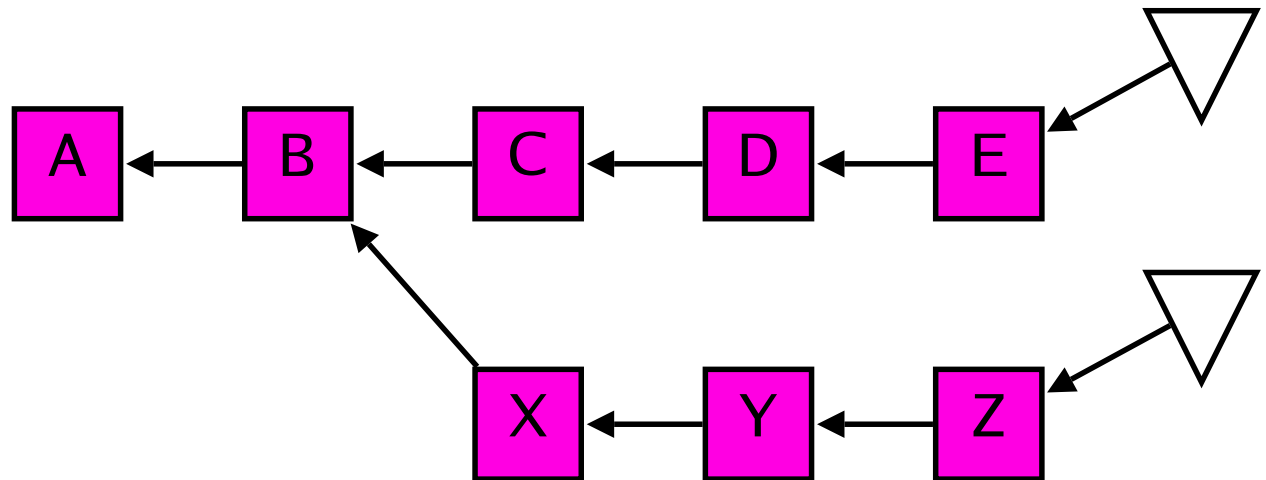
- Ein Commit kann mehrere Nachkommen haben.
- Diese nennt man dann *Branches*.
- Mit `git-commit` wird auch die aktuelle Referenz geupdatet.
- Für mehrere Branches werden mehrere Referenzen benötigt.
- Branches können angelegt werden mit:  
`git-branch Neuer-Branch [Alter-Branch]` bzw.  
`git-checkout -b Neuer-Branch [Alter-Branch]`
- Zwischen Branches kann gewechselt werden (Working Copy und Index) mittels:  
`git-checkout Branch-Name`
- Alle Branches auflisten: `git-branch`



# Branch (2/2)

▽ Reference

■ Commit



- Einleitung
- Repository
- Working Copy und Index
- Branchen und Mergen**
  - Branch (1/2)
  - **Branch (2/2)**
  - Merge (1/2)
  - Merge (2/2)
  - Rebase (1/2)
  - Rebase (2/2)
  - Konflikte
- Kollaboration
- GIT Server
- Rewriting History
- Subversion Integration
- Eigene Tools
- Weitere Tools
- URLs



# Merge (1/2)

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

● Branch (1/2)

● Branch (2/2)

● Merge (1/2)

● Merge (2/2)

● Rebase (1/2)

● Rebase (2/2)

● Konflikte

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

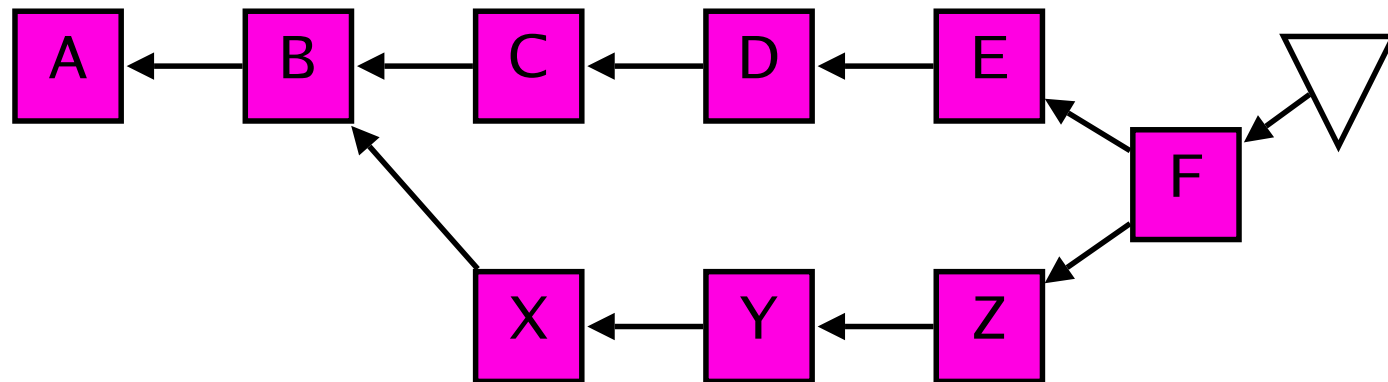
- Ein Commit kann mehrere Vorfahren haben.
- Diese nennt man dann *Merges*.
- Die Änderungen aus einem Branch in den aktuellen Branch mergen:  
`git-merge Anderer-Branch`
- Manchmal möchte man auch mehrere Branches gleichzeitig mergen:  
`git-merge Anderer-Branch1 Anderer-Branch2 ...`
- Häufig möchte man anschließend den gemergten Branch löschen:  
`git-branch -d Anderer-Branch`



# Merge (2/2)

▽ Reference

■ Commit



Einleitung

Repository

Working Copy und Index

Branchen und Mergen

● Branch (1/2)

● Branch (2/2)

● Merge (1/2)

● Merge (2/2)

● Rebase (1/2)

● Rebase (2/2)

● Konflikte

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs



# Rebase (1/2)

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

● Branch (1/2)

● Branch (2/2)

● Merge (1/2)

● Merge (2/2)

● Rebase (1/2)

● Rebase (2/2)

● Konflikte

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Manchmal möchte man einen Branch auch mit einem neuen Ursprungspunkt neu erzeugen.
- Dieser Prozess wird *Rebase* genannt.
- Dazu wird im betroffenen Branch ausgeführt:  
`git-rebase Upstream-Branch`
- Das ist speziell dann interessant, wenn man Branches für Upstream-Repositorys warten oder aufbereiten möchte.
- Der ursprüngliche Branch wird dadurch dereferenziert,
- Aber über das Reflog kann er noch gefunden werden.





# Rebase (2/2)

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

- Branch (1/2)
- Branch (2/2)
- Merge (1/2)
- Merge (2/2)
- Rebase (1/2)
- Rebase (2/2)
- Konflikte

Kollaboration

GIT Server

Rewriting History

Subversion Integration

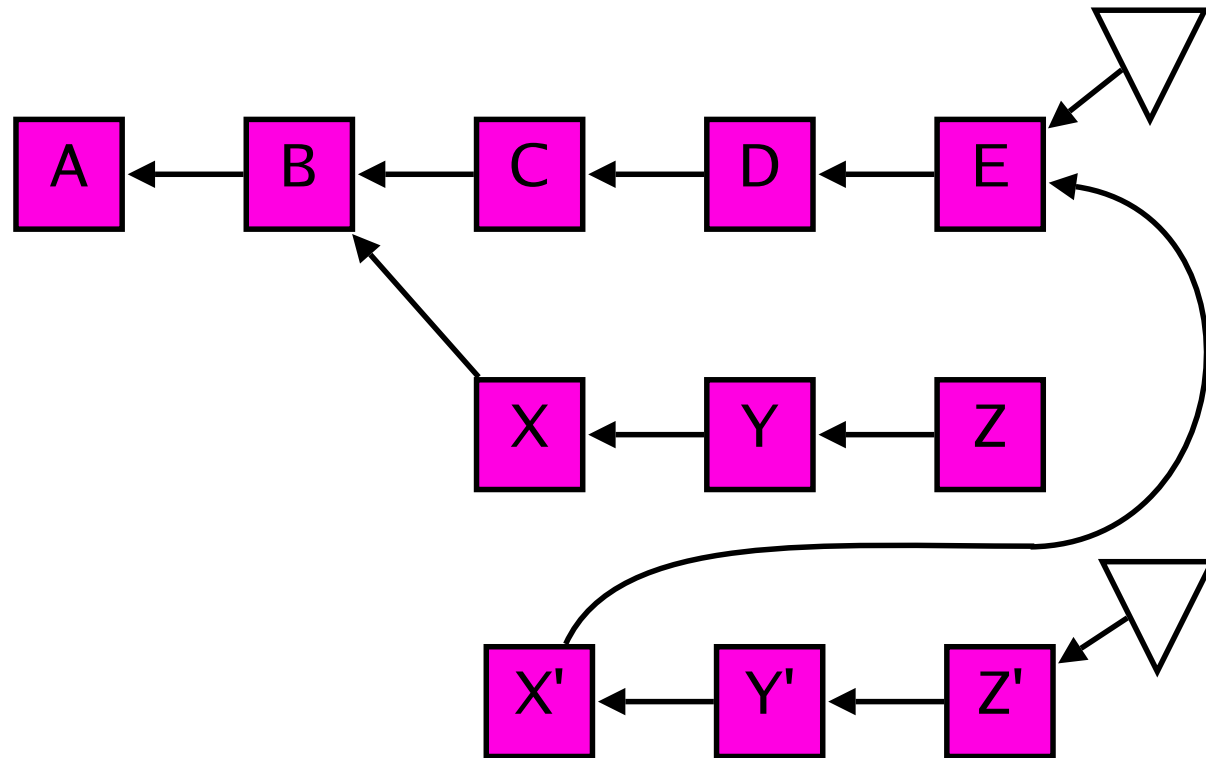
Eigene Tools

Weitere Tools

URLs

▽ Reference

■ Commit





# Konflikte

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

● Branch (1/2)

● Branch (2/2)

● Merge (1/2)

● Merge (2/2)

● Rebase (1/2)

● Rebase (2/2)

● Konflikte

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Beim Mergen und Rebasen können Konflikte auftreten.
- Dann wird das Kommando mit einem Fehler abgebrochen und die am Konflikt beteiligten Dateien werden im Index referenziert.
- Ein solcher Index kann nicht zu einem Tree umgewandelt werden.
- Dateien, für die der Konflikt aufgelöst wurde, werden mit `git-add` markiert.
- Nachdem alle Konflikte aufgelöst wurden:
  - `git-commit` (bei einem Merge)
  - `git-rebase --continue` (bei einem Rebase)



Einleitung

Repository

Working Copy und Index

Branchen und Mergen

**Kollaboration**

- Einleitung
- Protokolle
- Fetch (1/2)
- Fetch (2/2)
- Pull
- Push
- Example .git/config
- Clonen
- Patches
- Linear Development
- Bisect
- Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

# Kollaboration



# Einleitung

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

● Einleitung

● Protokolle

● Fetch (1/2)

● Fetch (2/2)

● Pull

● Push

● Example .git/config

● Clonen

● Patches

● Linear Development

● Bisect

● Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Grundlage für kollaboratives Arbeiten bildet die Möglichkeit zu branchen und zu mergen (bzw. rebasen).
- Dann braucht nur noch jede Entwicklerin ihre Branches - und die Änderungen können via Merge von einem Branch in einen anderen übernommen werden.
- GIT erlaubt es, Branches zwischen Repositorys zu spiegeln.
- Das bedeutet, jede Entwicklerin kann ihr eigenes Repository für ihre eigenen Branches betreiben.



# Protokolle

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

● Einleitung

● Protokolle

● Fetch (1/2)

● Fetch (2/2)

● Pull

● Push

● Example .git/config

● Clonen

● Patches

● Linear Development

● Bisect

● Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

## ■ Der Zugriff auf andere Repositorys ist über mehrere Protokolle möglich:

- ◆ `rsync://host.xz/path/to/repo.git/`
- ◆ `http://host.xz/path/to/repo.git/`
- ◆ `https://host.xz/path/to/repo.git/`
- ◆ `git://host.xz/path/to/repo.git/`
- ◆ `git://host.xz/~user/path/to/repo.git/`
- ◆ `ssh://[user@]host.xz[:port]/path/to/repo.git/`
- ◆ `ssh://[user@]host.xz/path/to/repo.git/`
- ◆ `ssh://[user@]host.xz/~user/path/to/repo.git/`
- ◆ `ssh://[user@]host.xz/~path/to/repo.git`
- ◆ `file:///path/to/repo.git/`

## ■ Einige kürzere Formen sind möglich. Zum Beispiel:

- ◆ `[user@]host.xz/path/to/repo.git/` (ssh)
- ◆ `/path/to/repo.git/` (lokale files)



# Fetch (1/2)

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

● Einleitung

● Protokolle

● Fetch (1/2)

● Fetch (2/2)

● Pull

● Push

● Example .git/config

● Clonen

● Patches

● Linear Development

● Bisect

● Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Mit dem Kommando `git-fetch` können Branches (Referenzen) aus einem anderen Repository importiert werden.

- Alle von der Referenz erreichbaren Objekte werden gegebenenfalls ebenfalls importiert.

- Beispiel:

```
git-fetch git://git.kernel.org/pub/scm/...  
...linux/kernel/git/torvalds/linux-2.6.git  
' +refs/heads/*:refs/remotes/linus/* '
```

- Solche *remote branches* benutzen in der Regel Referenz-Namen nach dem Muster `refs/remotes/short-rep-name/branch-name`.



# Fetch (2/2)

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

- Einleitung
- Protokolle
- Fetch (1/2)
- **Fetch (2/2)**
- Pull
- Push
- Example .git/config
- Clonen
- Patches
- Linear Development
- Bisect
- Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Per Default macht `git-fetch` nur *fast-forward* Updates.
- Das bedeutet, die alte Referenz muss von der neuen aus erreichbar sein.
- Die Option `-f` bzw. ein führendes `+` in der *refspec* überschreiben diesen Check.
- `git-fetch` ist ein *plumbing* (low-level) Tool.
- Normalerweise wird das High-level-Tool `git-pull` verwendet.



# Pull

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

● Einleitung

● Protokolle

● Fetch (1/2)

● Fetch (2/2)

● Pull

● Push

● Example .git/config

● Clonen

● Patches

● Linear Development

● Bisect

● Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Das (porcelain) Kommando `git-pull` ist eine Kombination aus `git-fetch` und `git-merge`.
- In der Regel wird die zu verfolgende Referenz in die Datei `.git/config` geschrieben.
- `git-pull` kann auch so konfiguriert werden, dass es `git-rebase` statt `git-merge` verwendet.
- `git-pull` wird wie `git-merge` in der Working Copy des zu aktualisierenden Branches gestartet.
- `git-pull` entspricht von der typischen Verwendung her dem `svn up` bzw. `cvs up` aus Subversion bzw. CVS.





# Push

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

- Einleitung
- Protokolle
- Fetch (1/2)
- Fetch (2/2)
- Pull
- **Push**
- Example .git/config
- Clonen
- Patches
- Linear Development
- Bisect
- Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- `git-push` wird zum Updaten von Remote Repositorys verwendet.
- Per Default macht `git-push` nur *fast-forward* Updates.
- In der Regel wird in `.git/config` eingestellt, welche lokalen Branches in welche Remote-Banches gepushed werden sollen.
- Als Parameter erwartet `git-push` den Namen des Remote Repositorys.
- Pushes über das Netzwerk werden in der Praxis fast ausschließlich per SSH Protokoll erlaubt.



# Example .git/config

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

- Einleitung
- Protokolle
- Fetch (1/2)
- Fetch (2/2)
- Pull
- Push
- **Example .git/config**
- Clonen
- Patches
- Linear Development
- Bisect
- Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

```
[remote "origin"]
url = git://git.kernel.org/pub/scm/linux/kernel/git/
fetch = +refs/heads/*:refs/remotes/origin/*
```

```
[remote "gitorious"]
url = git@gitorious.org:linux-cw/mainline.git
push = +refs/heads/*:refs/heads/*
```

```
[branch "master"]
remote = origin
merge = refs/heads/master
```

```
[branch "feature-proc-rlim"]
remote = origin
merge = refs/heads/master
rebase = true
```



# Clonen

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

- Einleitung
- Protokolle
- Fetch (1/2)
- Fetch (2/2)
- Pull
- Push
- Example .git/config
- Clonen
- Patches
- Linear Development
- Bisect
- Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Die meisten GIT-Repositorys beinhalten lokale Branches zu bereits vorhandenen Netzwerk-Repositories.
- Zum Erstellen solcher Repositorys gibt es das Porcelain-Tool `git-clone`.
- Beispiel:  

```
git-clone git://git.kernel.org/pub/scm/linux/...  
...kernel/git/torvalds/linux-2.6.git linux-26
```
- `git-clone` erzeugt ein lokales Repository mit einer passenden `.git/config` Datei, führt das initiale `git-fetch` durch und erzeugt einen lokalen Branch `master` vom gleichnamigen Branch im Remote-Repository.



# Patches

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

- Einleitung
- Protokolle
- Fetch (1/2)
- Fetch (2/2)
- Pull
- Push
- Example .git/config
- Clonen
- Patches
- Linear Development
- Bisect
- Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Ungeachtet der Möglichkeit Branches zu pullen möchte man oft den “traditionellen” Weg über Patches und Emails gehen.
- Dafür gibt es einige spezielle Porcelain-Tools.
- `git-format-patch` erzeugt fertige E-Mail Texte mit Patches.
- `git-send-email` versendet diese Patches per E-Mail.
- `git-am` applied Patches aus einem Mailbox-File.



# Linear Development

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

- Einleitung
- Protokolle
- Fetch (1/2)
- Fetch (2/2)
- Pull
- Push
- Example .git/config
- Clonen
- Patches

● Linear Development

- Bisect
- Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- GIT wird in der Regel mit *Non-Linear Development* in Verbindung gebracht.
- Aber mit GIT ist ebenso ein linearer Workflow umsetzbar:
- Alle Entwicklerinnen pushen in ein gemeinsames Upstream-Repository.
- Nur Fast-forward-Updates sind erlaubt.  
(Das kann mit einem `.git/hooks/update` Script erzwungen werden.)
- Das heißt, es kann nur pushen wer zuerst den aktuellen HEAD gepulled hat.



# Bisect

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

- Einleitung
- Protokolle
- Fetch (1/2)
- Fetch (2/2)
- Pull
- Push
- Example .git/config
- Clonen
- Patches
- Linear Development
- Bisect
- Bare Repositorys

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Fehlersuche in großen Projekten kann aufwendig sein.
- Das Tool `git-bisect` implementiert eine binäre Suche nach jenem Commit, der den Fehler eingeführt hat.
- Workflow:
  - ◆ `git bisect start [<bad> [<good>]]`
  - ◆ `git bisect { bad | good | skip } [<rev>]`
  - ◆ `git bisect { log | visualize }`
  - ◆ `git bisect reset`
- Automatisiertes Bisect mit Script:  
`git bisect run my_script`



# Bare Repositories

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

- Einleitung
- Protokolle
- Fetch (1/2)
- Fetch (2/2)
- Pull
- Push
- Example .git/config
- Clonen
- Patches
- Linear Development
- Bisect
- Bare Repositories

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Bei Netzwerk Repositories wird keine Working Copy benötigt.
- Daher gibt es auch *Bare Repositories*.
- Solche *Bare Repositories* sind Directories mit der Dateierweiterung `.git`, die nur die Dateien aus dem `.git` Directory beinhalten.
- Daher enden die GIT-URLs auch meistens mit `.git`.



[Einleitung](#)

[Repository](#)

[Working Copy und Index](#)

[Branchen und Mergen](#)

[Kollaboration](#)

**[GIT Server](#)**

- GIT via SSH
- git-daemon

[Rewriting History](#)

[Subversion Integration](#)

[Eigene Tools](#)

[Weitere Tools](#)

[URLs](#)

# GIT Server





# GIT via SSH

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

● GIT via SSH

● git-daemon

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Um per SSH auf ein Repository zugreifen zu können ist kein eigener Serverprozess notwendig.
- Für Pushes wird am Server das Programm `git-receive-pack` aufgerufen.
- Für Pulls wird am Server `git-upload-pack` aufgerufen.
- Restricted SSH login Shell für GIT: `git-shell`
- Beispiel:  

```
git-clone ssh://clifford@clifford.at/~/.  
...gitdata/linux-2.6 linux-2.6
```



# git-daemon

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

● GIT via SSH

● git-daemon

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Als Server für world-readable GIT Repositorys gibt es `git-daemon`.
- `git-daemon` kann als Stand-alone-Daemon oder unter `inetd` laufen.
- `git-daemon` exportiert per Default nur Repositorys mit der Datei `git-daemon-export-ok`.
- Dokumentation: `man git-daemon`



[Einleitung](#)

[Repository](#)

[Working Copy und Index](#)

[Branchen und Mergen](#)

[Kollaboration](#)

[GIT Server](#)

**[Rewriting History](#)**

- Interactive Rebase (1/2)
- Interactive Rebase (1/2)
- git-reset
- git-filter-branch

[Subversion Integration](#)

[Eigene Tools](#)

[Weitere Tools](#)

[URLs](#)

# Rewriting History



# Interactive Rebase (1/2)

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

● Interactive Rebase (1/2)

● Interactive Rebase (1/2)

● git-reset

● git-filter-branch

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- `git-rebase` erlaubt das Ersetzen einer Commit-Reihe durch eine andere.
- Beim *interactive rebase* kann nicht nur die Base-Revision sondern auch die Commit-Reihe selbst verändert werden:

```
git-rebase -i base-revision
```

- Beim interactive Rebase öffnet `git-rebase` einen Text-Editor mit den Commits in der betroffenen Commit-Reihe.
- Durch Editieren dieses Textes kann die Commit-Reihe verändert werden:
  - ◆ Entfernen von Commits.
  - ◆ Ändern der Reihenfolge.
  - ◆ Verändern von Commits.
  - ◆ Zusammenfügen von Commits.



# Interactive Rebase (1/2)

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

● Interactive Rebase (1/2)

● Interactive Rebase (1/2)

● git-reset

● git-filter-branch

Subversion Integration

Eigene Tools

Weitere Tools

URLs

```
pick b87fba1 Erster Patch im Rebase
pick 054f700 Zweiter Patch im Rebase
pick 4683128 Dritter Patch im Rebase
```

```
# Rebase b6c4c9a..4683128 onto b6c4c9a
```

```
#
```

```
# Commands:
```

```
# pick = use commit
```

```
# edit = use commit, but stop for amending
```

```
# squash = use commit, but meld into previous commit
```

```
#
```

```
# If you remove a line here THAT COMMIT WILL BE LOST
```

```
# However, if you remove everything, the rebase will
```

```
# be aborted.
```

```
#
```

Mehr Informationen: `man git-rebase`



# git-reset

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

● Interactive Rebase (1/2)

● Interactive Rebase (1/2)

● **git-reset**

● git-filter-branch

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Mit `git-reset` kann eine Referenz auf einen (alten) Commit (zurück-)gesetzt werden.

- Nur die HEAD Referenz setzen:

```
git-reset --soft commit
```

- Die HEAD Referenz und den Index setzen:

```
git-reset --mixed commit (default)
```

- Die HEAD Referenz, Index und Working-Copy setzen:

```
git-reset --hard commit
```

- Beispiel:

```
git branch topic/wip  
git reset --hard HEAD~3  
git checkout topic/wip
```



# git-filter-branch

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

● Interactive Rebase (1/2)

● Interactive Rebase (1/2)

● git-reset

● git-filter-branch

Subversion Integration

Eigene Tools

Weitere Tools

URLs

- Mit `git-filter-branch` kann man automatisiert ganze Branches verändern.
- Natürlich wird es dadurch unter Umständen sehr erschwert, Änderungen einfach zu pullen, zu mergen und zu pushen.
- Beispiel (Script in der Working Copy):  

```
git-filter-branch --tree-filter 'rm ...  
... filename' HEAD
```
- Beispiel (Script direkt mit dem Index):  

```
git-filter-branch --index-filter 'git-...  
...update-index --remove filename' HEAD
```



# Subversion Integration

- [Einleitung](#)
- [Repository](#)
- [Working Copy und Index](#)
- [Branchen und Mergen](#)
- [Kollaboration](#)
- [GIT Server](#)
- [Rewriting History](#)
- [Subversion Integration](#)**
  - [git-svn](#)
  - [Example](#)
  - [git-svnimport](#)
- [Eigene Tools](#)
- [Weitere Tools](#)
- [URLs](#)





# git-svn

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

● git-svn

● Example

● git-svnimport

Eigene Tools

Weitere Tools

URLs

- Mit `git-svn` können SVN Repositorys in lokale GIT Repositorys gecloned werden.
- Lokale Commits können auch ins SVN Repository committet werden.
- Neue Revisions im Subversion können auch ins lokale Repository importiert werden.
- Dokumentation: `man git-svn`



# Example

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

● git-svn

● Example

● git-svnimport

Eigene Tools

Weitere Tools

URLs

```
$ git-svn clone http://svn.clifford.at/stfl/trunk stfl
$ cd stfl
```

```
# do some work
$ git-commit ...
```

```
# instead of 'svn up':
$ git-svn rebase
```

```
# pushing changes to SVN repository:
$ git-svn dcommit
```

```
# import svn:ignore properties:
$ git-svn show-ignore >> .git/info/exclude
```



# git-svnimport

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

● git-svn

● Example

● git-svnimport

Eigene Tools

Weitere Tools

URLs

- Das alte Tool `git-svnimport` importierte ganze SVN Repositorys.
- Ein Zurückpropagieren zu SVN war nicht möglich.
- Die bei Subversion übliche Verzeichnisstruktur (`/trunk`, `/branches`, `/tags`, etc.) wurde vorausgesetzt.
- `git-svnimport` ist seit Oktober 2007 deprecated.
- Das Script kann aber noch unter `/contrib/examples/` in den GIT Sourcen gefunden werden.



[Einleitung](#)

[Repository](#)

[Working Copy und Index](#)

[Branchen und Mergen](#)

[Kollaboration](#)

[GIT Server](#)

[Rewriting History](#)

[Subversion Integration](#)

**[Eigene Tools](#)**

- [Shell Scripts](#)
- [Tipps](#)
- [git-sh-setup](#)

[Weitere Tools](#)

[URLs](#)

# Eigene Tools



# Shell Scripts

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

● Shell Scripts

● Tipps

● git-sh-setup

Weitere Tools

URLs

- Viele der GIT (Porcelain-)Tools sind als Shell-Skripte implementiert.
- Diese Tools benutzen andere GIT-Kommandos für Low-Level-Aufgaben.
- Es bietet sich daher an, bei Bedarf eigene (Porcelain-)Tools als Shell-Skripts zu implementieren.
- Beispiele für GIT Tools, die als Shell-Skripte implementiert sind:  
`git-bisect`, `git-clone`, `git-merge`, `git-pull`,  
`git-rebase`, `git-stash`, ...



# Tipps

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

● Shell Scripts

● Tipps

● git-sh-setup

Weitere Tools

URLs

- Files im `.git` Directory nur direkt angreifen, wenn es kein entsprechendes Tool gibt.  
Siehe zum Beispiel: `git-update-ref`, `git-rev-parse`, `git-config`, ...
- GIT Kommandos sollten von Scripts immer mit `'git kommando'` statt `'git-kommando'` gestartet werden.
- Parsen von Argumenten und Optionen:  
`git-rev-parse --parseopt`
- Initialisieren von Scripten: `git-sh-setup`



# git-sh-setup

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

● Shell Scripts

● Tipps

● git-sh-setup

Weitere Tools

URLs

- Mit `' . git-sh-setup'` kann ein GIT Shell-Script bequem initialisiert werden.
- `git-sh-setup` definiert einige praktische Funktionen.
- Davor sind einige Shell Variablen zu setzen.
- Dokumentation: `man git-sh-setup`

## Usage Example:

```
USAGE='[  | save | list | clear | create ]'
SUBDIRECTORY_OK=Yes
OPTIONS_SPEC=
. git-sh-setup
require_work_tree
cd_to_toplevel
```



[Einleitung](#)

[Repository](#)

[Working Copy und Index](#)

[Branchen und Mergen](#)

[Kollaboration](#)

[GIT Server](#)

[Rewriting History](#)

[Subversion Integration](#)

[Eigene Tools](#)

**Weitere Tools**

- [git-stash](#)
- [git-clean](#)
- [git-tag](#)
- [git-blame](#)

[URLs](#)

# Weitere Tools





# git-stash

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

● git-stash

● git-clean

● git-tag

● git-blame

URLs

- `git-stash` comittet lokale Changes nach `refs/stash` und resettet den Working-Tree.
- `git-stash apply` spielt diese Änderungen wieder im Working-Tree ein.
- Das ist vor allem nützlich um 'mal schnell' die aktuelle Arbeit beiseite stellen zu können um z. Bsp. einen Hotfix zu schreiben.

## Usage Example:

```
$ git-stash
<edit emergency fix>
$ git-commit -a -m "Fix in a hurry"
$ git-stash apply
```



# git-clean

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

● git-stash

● **git-clean**

● git-tag

● git-blame

URLs

- `git-clean` entfernt Dateien aus dem Working-Tree die nicht unter der Kontrolle von git stehen.
- Damit erfüllt `git-clean` eine ähnliche Funktion wie das `make clean`, das die meisten Makefiles implementieren.
- Lokale Änderungen können mit `git-checkout` rückgängig gemacht werden.
- Lokale Änderungen, die dem Index hinzugefügt wurden, können mit `git-reset` und `git-checkout -f` rückgängig gemacht werden.
- Dokumentation: `man git-clean`, `man git-checkout`, `man git-reset`



# git-tag

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

● git-stash

● git-clean

● git-tag

● git-blame

URLs

- Mit `git-tag` können Tags erzeugt und abgefragt werden.
- Tags sind Referenzen und können z. Bsp. ganz normal mit `git-checkout` ausgecheckt werden.
- Bei Bedarf wird ein eigenes Tag-Objekt erzeugt, das eine Message sowie eine GPG-Signatur beinhalten kann.
- Dokumentation: `man git-tag`



# git-blame

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

● git-stash

● git-clean

● git-tag

● git-blame

URLs

- `git-blame` gibt eine Datei zeilenweise aus und fügt jeder Zeile Informationen zur letzten Änderung hinzu.
- Erleichtert es die “Schuldige” für einen Bug zu finden.
- Oder einfach nur den größeren Kontext einer Änderung zu ermitteln.
- Für die Weiterverarbeitung gibt es ein eigenes “porcelain” Ausgabeformat (Option `-p`).
- Dokumentation: `man git-blame`



Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

**URLs**

- GIT
- Diese Präsentation

# URLs



# GIT

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

● GIT

● Diese Präsentation

## ■ GIT Homepage

<http://git.or.cz/>

## ■ GIT Manpages

<http://www.kernel.org/pub/software/scm/git/docs/>

## ■ GIT User's Manual

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



# Diese Präsentation

Einleitung

Repository

Working Copy und Index

Branchen und Mergen

Kollaboration

GIT Server

Rewriting History

Subversion Integration

Eigene Tools

Weitere Tools

URLs

● GIT

● Diese Präsentation

## ■ Cliffords Tools

<http://svn.clifford.at/tools/trunk/>

## ■ Clifford Wolf

<http://www.clifford.at/>

## ■ Weitere Präsentationen

<http://www.clifford.at/papers/>

## ■ Diese Präsentation

<http://www.clifford.at/papers/2008/git/>