

Verteilte Versionskontrolle mit Git von A - Z
Spiel, Spass und Spannung mit Git
Git Goodies

Sebastian „tokkee“ Harl
<sh@teamix.net> / <tokkee@lusc.de>

team(ix) GmbH / LUSC e.V.

LUG-Camp 2011, 2. Juni 2011, Tschierv, CH



- gegründet 2001
- Ursprünge: Open-Source und Netzwerke
 - Debian
 - Nagios
 - Schulungen
 - u.v.m.
- Heute auch:
 - NetApp
 - VMWare
 - Riverbed (WAN-Beschleunigung)
 - Juniper
 - N-IX (Nürnberger Internet-eXchange)

- Wer bezeichnet sich als Programmierer?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?
 - ... mit mehr als 10 Entwicklern?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?
 - ... mit mehr als 10 Entwicklern?
 - ... mit mehr als 100 Entwicklern?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?
 - ... mit mehr als 10 Entwicklern?
 - ... mit mehr als 100 Entwicklern?
 - ... mit mehr als 1000 Entwicklern?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?
 - ... mit mehr als 10 Entwicklern?
 - ... mit mehr als 100 Entwicklern?
 - ... mit mehr als 1000 Entwicklern?
- Wer hat schon ein Versionsverwaltungssystem (VCS) verwendet?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?
 - ... mit mehr als 10 Entwicklern?
 - ... mit mehr als 100 Entwicklern?
 - ... mit mehr als 1000 Entwicklern?
- Wer hat schon ein Versionsverwaltungssystem (VCS) verwendet?
- Wer hat schon ein zentrales VCS (CVS, SVN, ...) verwendet?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?
 - ... mit mehr als 10 Entwicklern?
 - ... mit mehr als 100 Entwicklern?
 - ... mit mehr als 1000 Entwicklern?
- Wer hat schon ein Versionsverwaltungssystem (VCS) verwendet?
- Wer hat schon ein zentrales VCS (CVS, SVN, ...) verwendet?
- Wer hat schon ein dezentrales VCS (Git, bzt, Mercurial, ...) verwendet?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?
 - ... mit mehr als 10 Entwicklern?
 - ... mit mehr als 100 Entwicklern?
 - ... mit mehr als 1000 Entwicklern?
- Wer hat schon ein Versionsverwaltungssystem (VCS) verwendet?
- Wer hat schon ein zentrales VCS (CVS, SVN, ...) verwendet?
- Wer hat schon ein dezentrales VCS (Git, bzzr, Mercurial, ...) verwendet?
- Wer hat schon mit Git gearbeitet?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?
 - ... mit mehr als 10 Entwicklern?
 - ... mit mehr als 100 Entwicklern?
 - ... mit mehr als 1000 Entwicklern?
- Wer hat schon ein Versionsverwaltungssystem (VCS) verwendet?
- Wer hat schon ein zentrales VCS (CVS, SVN, ...) verwendet?
- Wer hat schon ein dezentrales VCS (Git, bzt, Mercurial, ...) verwendet?
- Wer hat schon mit Git gearbeitet?
- Wer hat noch nie was von Git gehört?

- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
 - ... mit mehr als 1 Entwickler?
 - ... mit mehr als 10 Entwicklern?
 - ... mit mehr als 100 Entwicklern?
 - ... mit mehr als 1000 Entwicklern?
- Wer hat schon ein Versionsverwaltungssystem (VCS) verwendet?
- Wer hat schon ein zentrales VCS (CVS, SVN, ...) verwendet?
- Wer hat schon ein dezentrales VCS (Git, bzzr, Mercurial, ...) verwendet?
- Wer hat schon mit Git gearbeitet?
- Wer hat noch nie was von Git gehört? ... **RAUS!** ;-)

Grundlagen: Was ist Versionskontrolle?

Was ist Versionskontrolle?

Typen von Versionskontrollsystemen

Dezentrale Versionskontrolle

Arbeiten mit Git

Git Goodies

- technisch gesehen: ein Haufen Dateien mit Meta-Informationen und irgendwelchen Beziehungen untereinander 😊

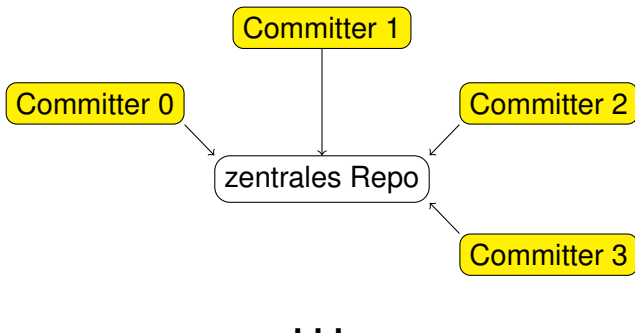
- technisch gesehen: ein Haufen Dateien mit Meta-Informationen und irgendwelchen Beziehungen untereinander 😊
- Protokollieren von Änderungen an (Quell-)text
- Archivierung mit „Rücksetz-Operation“
- koordinierter Zugriff
- parallele Entwicklungszweige (neue Features, alte Releases)

- lock/modify/write
- copy/modify/merge

- lock/modify/write
- copy/modify/merge
- **lokale Versionierung**
- zentrale Versionierung
- dezentrale Versionierung

- lock/modify/write
- copy/modify/merge

- lokale Versionierung
- **zentrale Versionierung**
- dezentrale Versionierung



- lock/modify/write
- copy/modify/merge

- lokale Versionierung
- zentrale Versionierung
- **dezentrale Versionierung**

Grundlagen: Was ist Versionskontrolle?

Dezentrale Versionskontrolle

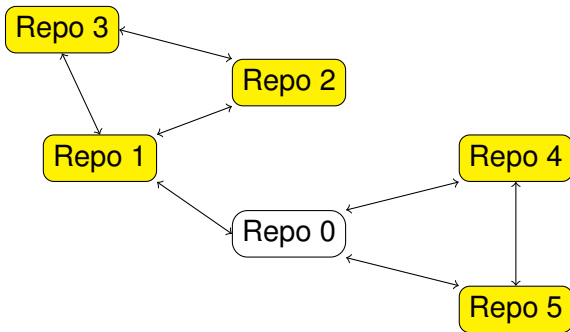
Grundlagen von dezentralen Versionskontrollsystemen
Arbeitsweise von dezentralen Versionskontrollsystemen

Arbeiten mit Git

Git Goodies

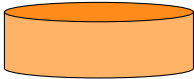
... woher kommt Git eigentlich?

- üblicherweise ein/wenige Hauptentwickler/Projektleiter
- viele Mitwirkende (versch. Umfang/Arbeitsgebiet)
- ggf. Subsystem-Verantwortliche;
Entwickler mit mehreren Arbeitsrechnern
- ein „zentraler“/„offizielles“ Repository
- temporäre und Feature-Branches

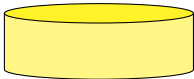


...

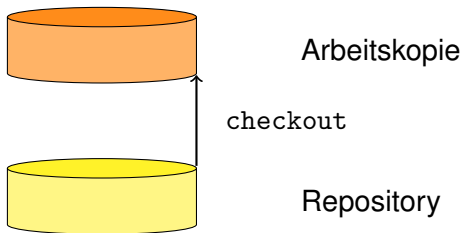
- „Peer-to-Peer“ Ansatz
- jede Arbeitskopie bringt ein komplettes Repository mit (Klon)
- gearbeitet wird auf lokalem Repository
 - ⇒ kein Netzwerk-Zugriff nötig
 - ⇒ Operationen schnell
 - ⇒ Offline-Arbeit möglich
- automatisches „Backup“ durch Repository-Klons
- Zusammenführen meist auf Basis eines „Web-of-Trust“



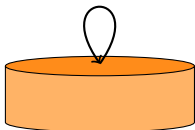
Arbeitskopie



Repository



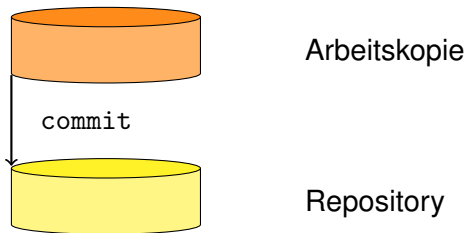
modify



Arbeitskopie

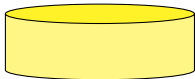


Repository





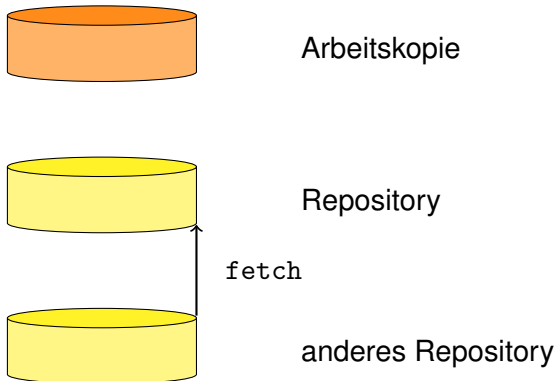
Arbeitskopie

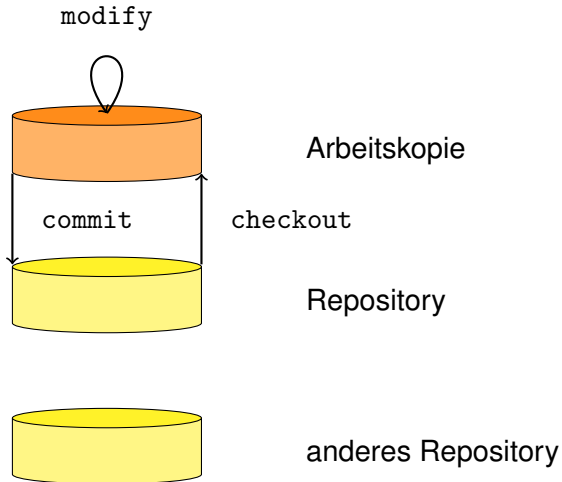


Repository



anderes Repository



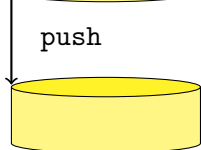




Arbeitskopie



Repository



anderes Repository

Grundlagen: Was ist Versionskontrolle?

Dezentrale Versionskontrolle

Arbeiten mit Git

- Datenhaltung in Git und grundlegende Konzepte
- Git konfigurieren
- Änderungen vornehmen und protokollieren
- Änderungen anzeigen
- Branching und Merging
- Frontends

Git Goodies

- <http://www.git.or.cz/>
- VCS (Version Control System)
- 2005 von Linus Torvalds initiiert
(aktueller Maintainer: Junio C. Hamano)
- dezentral
- schnell und effizient
- kryptographisch gesichert
- „Toolkit design“
- OpenSource (GPLv2)
- weit verbreitet im Einsatz (z.B. Linux Kernel, Ruby on Rails, Perl, WINE, X.org, GNOME, Qt, Debian, ...)

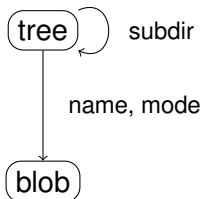
- ca. 150 einzelne Befehle
- „Porcelains“ und „Plumbing“
- Dokumentation als Manpages — `git(7)`
- `git help`, `git <command> -h`
- Benutzer Handbuch: <http://www.kernel.org/pub/software/scm/git/docs/user-manual.html>
- „Git Community Book“: <http://book.git-scm.com/>
- Buch „Pro Git“: <http://progit.org/book/>

- **Repository:** „Datenbank“/„Storage“ für Dateien und deren Historie mit Meta-Informationen
- **Arbeitskopie:** lokale Kopie der Daten aus einem bestimmten Snapshot (Revision)
- **Commit:** Snapshot der Daten zu einem bestimmten Zeitpunkt
- **Branch:** aufeinanderfolgende Commits; Entwicklungsstrang
- **Tag:** Name für einen bestimmten Commit, ggf. mit weiteren Meta-Informationen

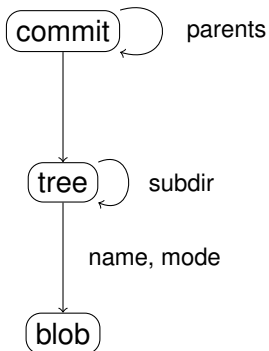
- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe

blob

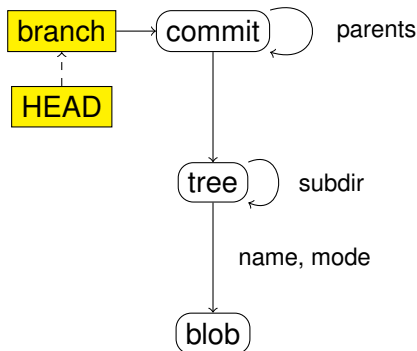
- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe



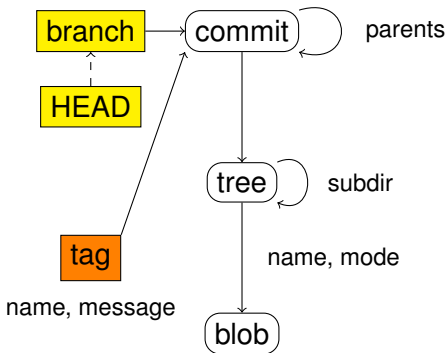
- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe



- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe



- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe



Sich Git vorstellen

- `git config --global user.name <Dein Name>`
- `git config --global user.email <du@deine-domain.tld>`
- → Benutzerinformationen für Commit-Metadaten

Bunt und in Farbe

- `git config --global color.ui auto`
- → farbige branch, diff, grep, status Ausgaben

Bunt und in Farbe

- `git config --global color.ui auto`
- → farbige branch, diff, grep, status Ausgaben

Nützliche Aliase

- `git config --global merge.tool vimdiff`
- `git config --global push.default current`
- `git config --global alias.wdiff 'diff --color-words'`
- ...

Nützliche Aliase

- `stat 'status -s'`
- `unadd 'rm --cached'`
- `dlog 'log --decorate'`
- `graph 'log --graph
--pretty=format:"%Cred%h%Creset
-%C(yellow)%d%Creset %s
%Cgreen(%cr)
%C(bold blue)<%an>%Creset"
--abbrev-commit --date=relative'`

Neues, leeres Repository

```
$ mkdir project  
$ cd project  
$ git init  
Initialized empty Git  
repository in .../.git/
```


Neues, leeres Repository

```
$ mkdir project  
$ cd project  
$ git init  
Initialized empty Git  
repository in .../.git/
```

Bestehendes Repository „klonen“

```
$ git clone <rep>  
...
```

Ändern

```
$ vim foo bar
```

```
$ git add foo bar
```

- add, rm, mv

Ändern

```
$ vim foo bar
```

```
$ git add foo bar
```

- add, rm, mv

Geschichte fortführen/ändern

```
$ git commit
```

```
$ git reset --hard HEAD^
```

- reset, revert, checkout

- Einzeilige, kurze (< 80, optimal < 50 Zeichen)
Zusammenfassung
- Leerzeile
- Detaillierte Beschreibung/Erklärung
- nicht vorgeschrieben, aber „common practice“ und von vielen Tools erwartet

git-remote: do not use user input in a printf format string

'git remote show' substituted the remote name into a string that was later used as a printf format string. If a remote name contains a printf format specifier like this:

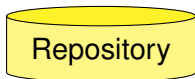
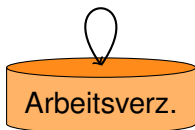
```
$ git remote add foo%sbar .
```

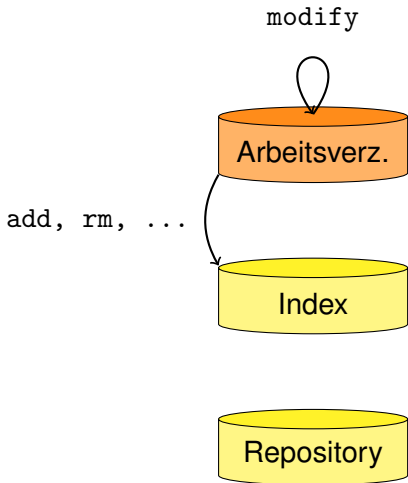
then the command

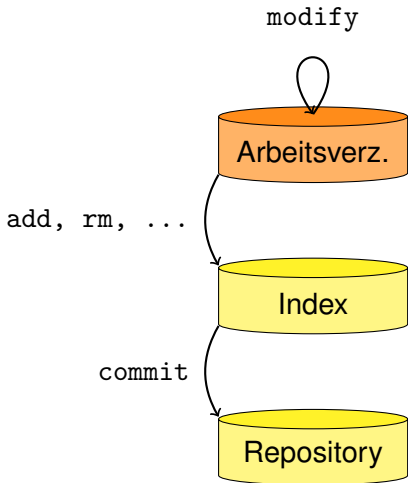
```
$ git remote show foo%sbar
```

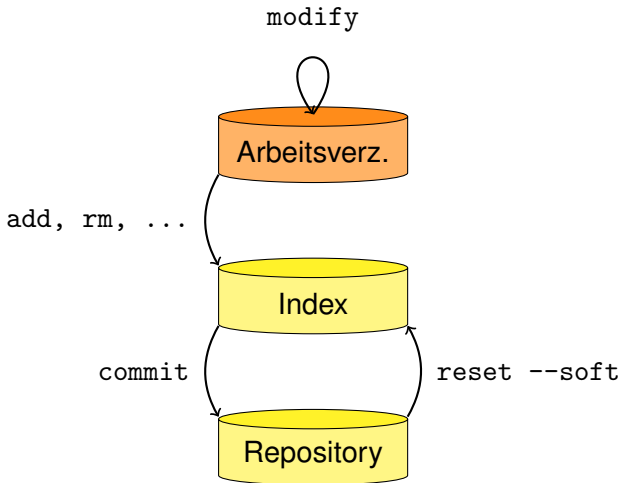
would print garbage (if you are lucky) or crash. This fixes it.

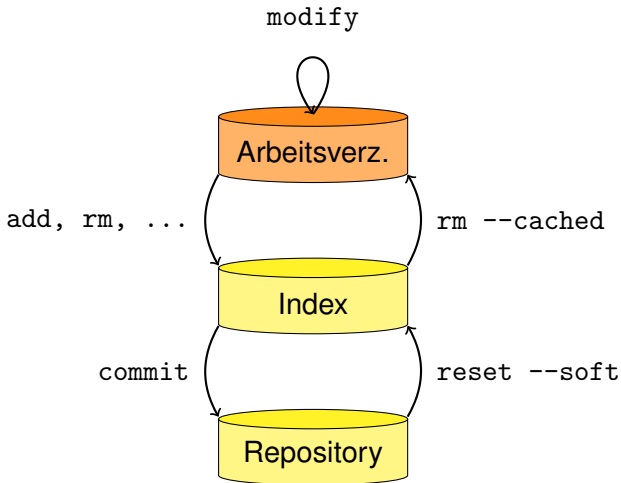
modify

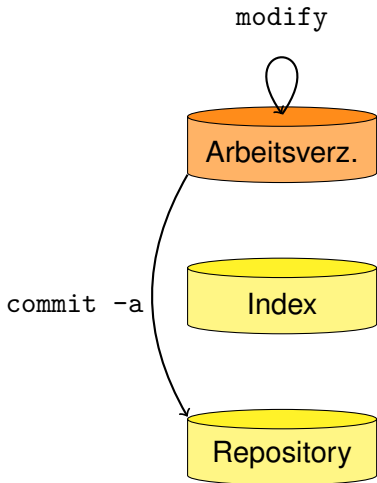


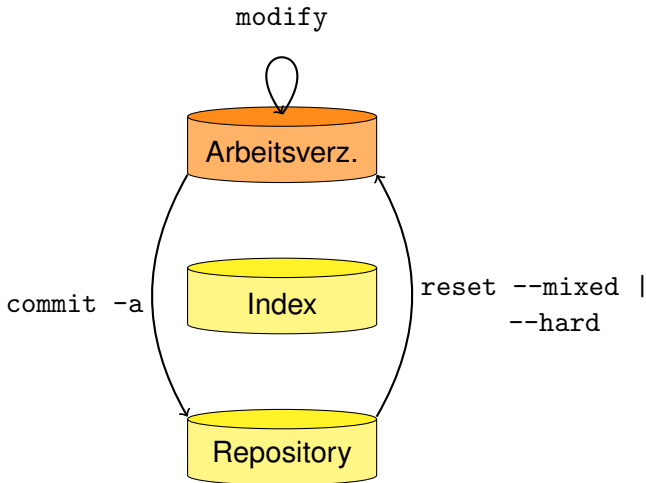












`git reset --hard | --soft?`

oder

`git revert?`

oder

`git checkout?`

Status der Arbeitskopie

```
$ git status
```

```
$ git diff
```

Status der Arbeitskopie

```
$ git status
```

```
$ git diff
```

Historie betrachten

```
$ git log
```

```
$ tig
```

Status der Arbeitskopie

```
$ git status  
$ git diff
```

Historie betrachten

```
$ git log  
$ tig
```

Objekte betrachten

```
$ git show  
$ git show HEAD:foo
```

(siehe gitrevisions(7))

- Commits, Trees, Blobs, Tags


```
$ git tag -m '<Beschreibung>' <Name> <Commit>  
$ git tag -l
```

- „Zeiger“ auf einen Commit
optional mit Metadaten (“annotated tag“)
- Kennzeichnung von bestimmten Entwicklungsständen
(insb. Releases)
- „annotated tag“: Autor, Datum, Beschreibung, optional
GnuPG Signatur

- Branch: „automatischer“ Zeiger auf eine Reihe von Commits
- HEAD: Zeiger auf den aktuellen Branch
- master: „Standard“-Branch
- Merge: Zusammenführen von zwei (oder mehr) Entwicklungssträngen

Branch erzeugen

```
$ git checkout -b <Name>
```

Branch erzeugen

```
$ git checkout -b <Name>
```

```
$ git branch  
master  
* <Name>
```

Branch erzeugen

```
$ git checkout -b <Name>
```

```
$ git branch  
master  
* <Name>
```

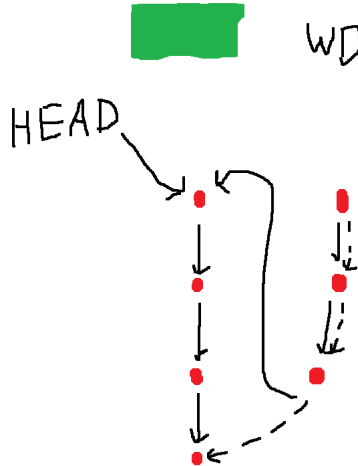
Branches zusammenführen

```
$ git merge master  
$ git rebase master # nur in privaten Branches!
```

Was war eigentlich gestern?

Git von A - Z

Folie 29



Konflikte auflösen

- Konflikte entstehen, wenn die gleiche Stelle unterschiedlich geändert wurde ⇒ manuelles Eingreifen nötig
- Commit-Erzeugung wird unterbrochen
- Konfliktanzeiger in den betroffenen Dateien
- manuelle Entscheidung, wie beide Änderungen zusammengeführt werden
- `git mergetool`

Repository klonen

```
$ git clone <rep>
```

Austauschen von Änderungen

```
$ git pull
```

```
$ git push
```

Alternativ:

- `git format-patch`, `git send-mail`

- „remote“: Repository, dessen Änderungen verfolgt werden
- „remote branch“: Branch, welcher der Zustand in einem anderen Repository widerspiegelt
- technisch: Branch in einem anderen Namensraum mit anderer Semantik

Arbeiten mit „remotes“

```
$ git remote add <Name> URL
```

```
$ git remote update <Name>
```

```
$ git push <Name> # ggf. zusätzlich Branch angeben
```

- Standard-Remote
- wird automatisch bei `clone` erstellt, inkl. lokaler Kopie des „master“ Branch
- Empfehlung: `pushurl` verwenden

```
[remote "origin"]
  fetch = +refs/heads/*:refs/remotes/origin/*
  url = git://git.domain.tld/project.git
  pushurl = ssh://git@git.domain.tld:project.git
[branch "master"]
  remote = origin
  merge = refs/heads/master
```

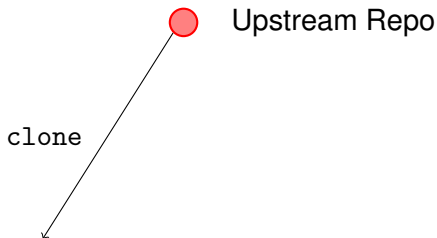
- lokal: `/path/to/repository/`
- http: `http://domain.tld/repository.git`
- git: `git://domain.tld/repository.git`
- ssh: `domain.tld:path/to/repository/`

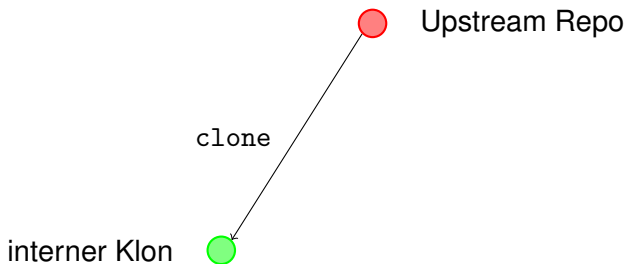
gitolite¹ erleichtert die Verwaltung von Git-Servern.

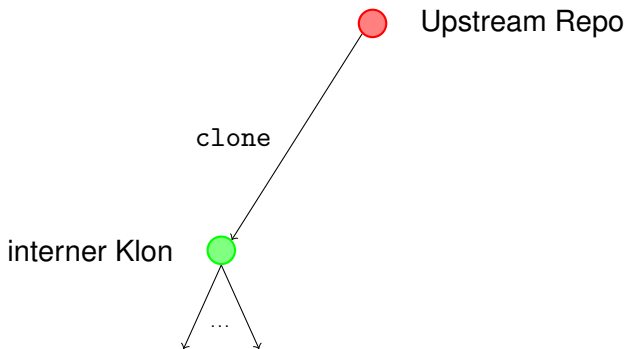
- Zugriff über einzelnen Benutzer und SSH Pubkeys
- Zugriffsberechtigung pro Branch oder Pfad
- einfache Konfiguration
- Konfiguration von Gitweb und git-daemon

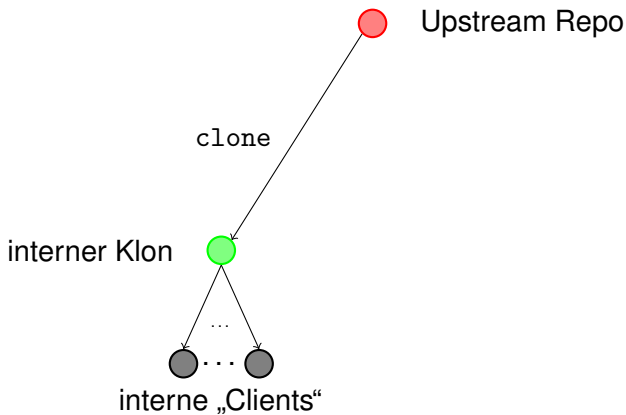
¹<http://github.com/sitaramc/gitolite>

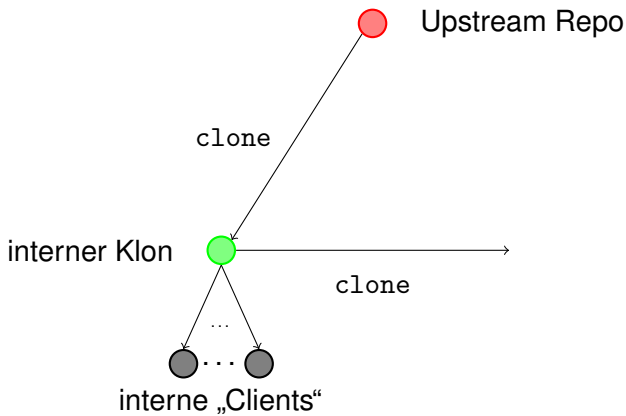
- Upstream Repo

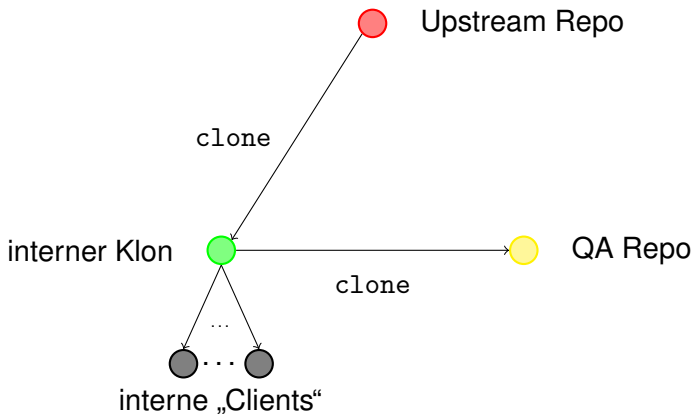


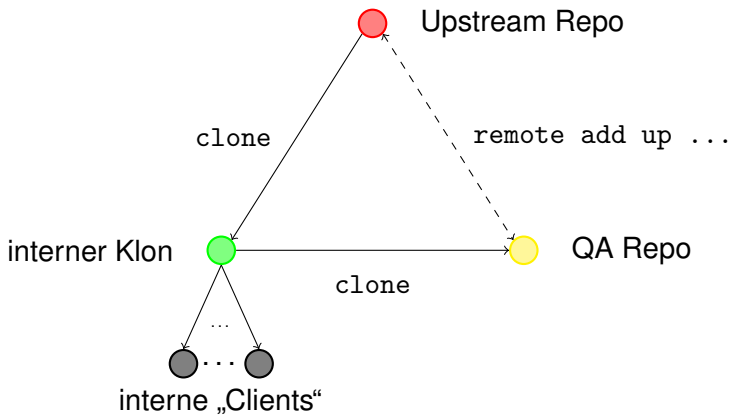


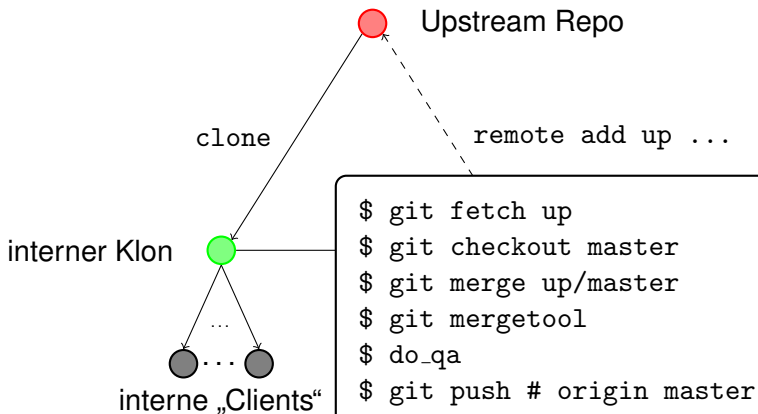












- `tig` (ncurses)
- `gitk` (Tk, read-only)
- `qgit` (Qt)
- `magit` (emacs)
- `egit` (Eclipse)

Grundlagen: Was ist Versionskontrolle?

Dezentrale Versionskontrolle

Arbeiten mit Git

Git Goodies

Protokollierung, Änderungen betrachten

Die Historie ändern

Änderungen verwalten

Interaktion mit der Umwelt

```
git diff --color-words
```



```
git diff --color-words
```

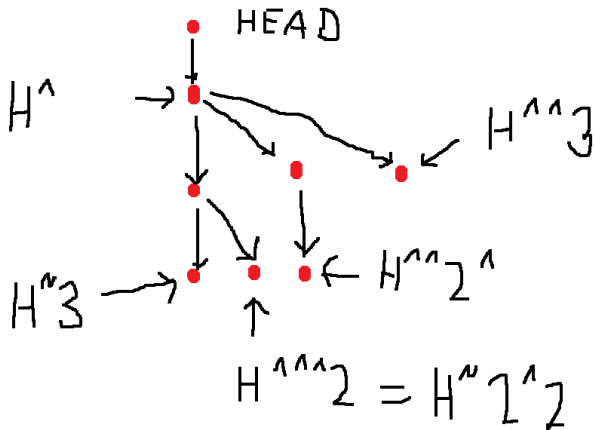
```
-brown fox jumps over the lazy  
-dog.  
+brown fox jumps over the lousy  
+and lazy dog.
```

```
brown fox jumps over the lousy  
and lazy dog.
```

```
git reflog
```

```
master@{two weeks ago}  
git log --since 'last monday'
```

→ git rev-parse



```
git log -p -U0
```

```
git log -S<string>
```

```
git log -S<string> -p --pickaxe-all
```

```
git bisect
```

git cherry / git-wtf²

²<http://git-wt-commit.rubyforge.org/>

git cherry / git-wtf²

```
Local branch: sh/check_dbi
[ ] NOT in sync with remote (you should push)
  - Merge branch 'master' of $remote_url into sh/check_dbi [cc6024e]
  - check_dbi: Added simple regex example to help output. [8123a9d]
  - check_dbi: Added SERVER_VERSION metric. [6c85913]
  - check_dbi: Added -r and -R options. [7852606]
  - check_dbi: Added -e option. [357c8e4]
  ... and 8 more (use -A to see all).
Remote branch: origin/master ($remote_url)
[X] in sync with local
```

²<http://git-wt-commit.rubyforge.org/>

```
git rebase -i
```

```
git commit --amend
```

```
git filter-branch
```

```
git add -p
```

```
git stash
```

```
git cherry-pick
```

git svn


```
git fast-export
```

```
  bzt-fast-export
```

```
  darcs-fast-export
```

```
  hg-fast-export
```

```
  svn-dump-fast-export
```

```
git fast-import
```

```
  bzt-fast-import
```

```
  hg-fastimport
```

Vielen Dank für die Aufmerksamkeit!

Gibt es Fragen?

Kontakt:

team(ix) GmbH / LUSC e.V.

Sebastian „tokkee“ Harl

<sh@teamix.net> / <tokkee@lusc.de>

<http://tokkee.org/events.html>