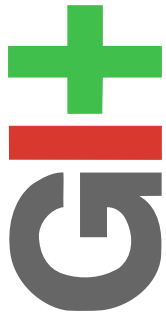
Verteilte Versionskontrolle mit Git

# Spiel, Spaß und Spannung mit Git

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

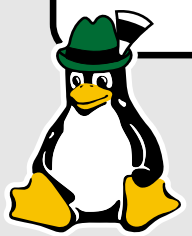
**Der ZeroMQ entfällt wegen Krankheit!**
**Dies ist der Ersatz :-)**

28. April 2012
**Grazer Linuxtage 2012**

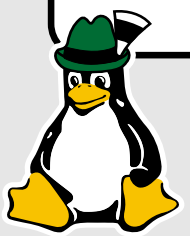
# Statistiken: Wer seid ihr?

- Wer bezeichnet sich als Programmierer?



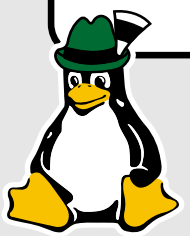
# Statistiken: Wer seid ihr?

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



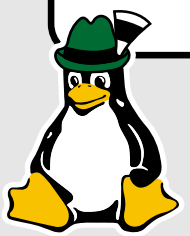
# Statistiken: Wer seid ihr?

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



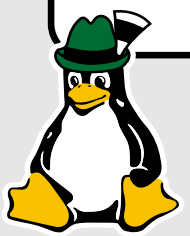
# Statistiken: Wer seid ihr?

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



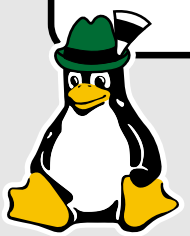
# Statistiken: Wer seid ihr?

- 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?



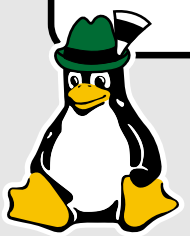
# Statistiken: Wer seid ihr?

- 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?



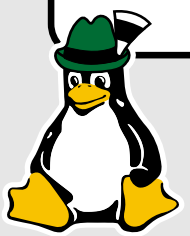
# Statistiken: Wer seid ihr?

- 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?



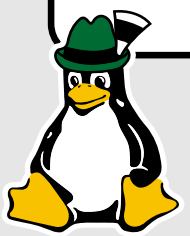
# Statistiken: Wer seid ihr?

- 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?



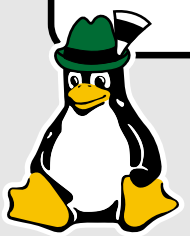
# Statistiken: Wer seid ihr?

- 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, bzr, Mercurial, ...) verwendet?



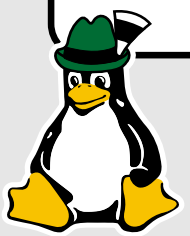
# Statistiken: Wer seid ihr?

- 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, bzr, Mercurial, ...) verwendet?
- Wer hat schon mit Git gearbeitet?



# Statistiken: Wer seid ihr?

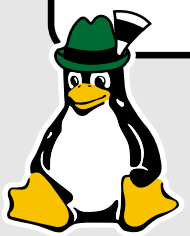
- Wer bezeichnet sich als Programmierer?
- Wer arbeitet an einem OpenSource-Projekt?
  - ... mit mehr als 1 Entwickler?
  - ... mit mehr als 10 Entwicklern?
  - ... mit mehr als 100 Entwicklern?
  - ... mit mehr als 1000 Entwicklern?
- Wer hat schon ein Versionsverwaltungssystem (VCS) verwendet?
- Wer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



## Grundlagen: Was ist Versionskontrolle?

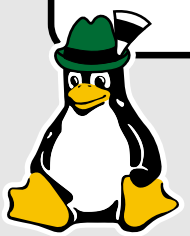
Dezentrale Versionskontrolle

Arbeiten mit Git



# Grundlagen: Was ist Versionskontrolle?

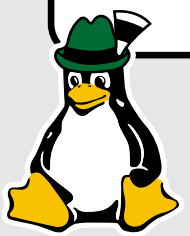
- technisch gesehen: ein Haufen Dateien mit Meta-Informationen und irgendwelchen Beziehungen untereinander ☺



# Grundlagen: Was ist Versionskontrolle?

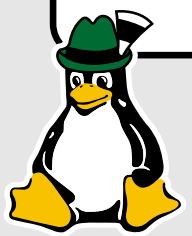
- technisch gesehen: ein Haufen Dateien mit Meta-Informationen und irgendwelchen Beziehungen untereinander ☺

- Protokollieren von Änderungen an (Quell-)text
- Archivierung mit „Rücksetz-Operation“
- koordinierter Zugriff
- parallele Entwicklungszweige (neue Features, alte Releases)



# Grundlagen: Typen von VCSen

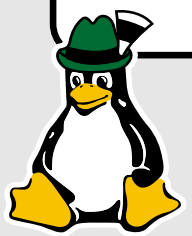
- lock/modify/write
- copy/modify/merge



# Grundlagen: Typen von VCSen

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

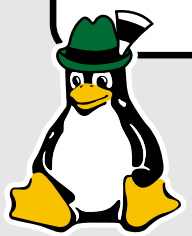
- lokale Versionierung
- zentrale Versionierung
- dezentrale Versionierung



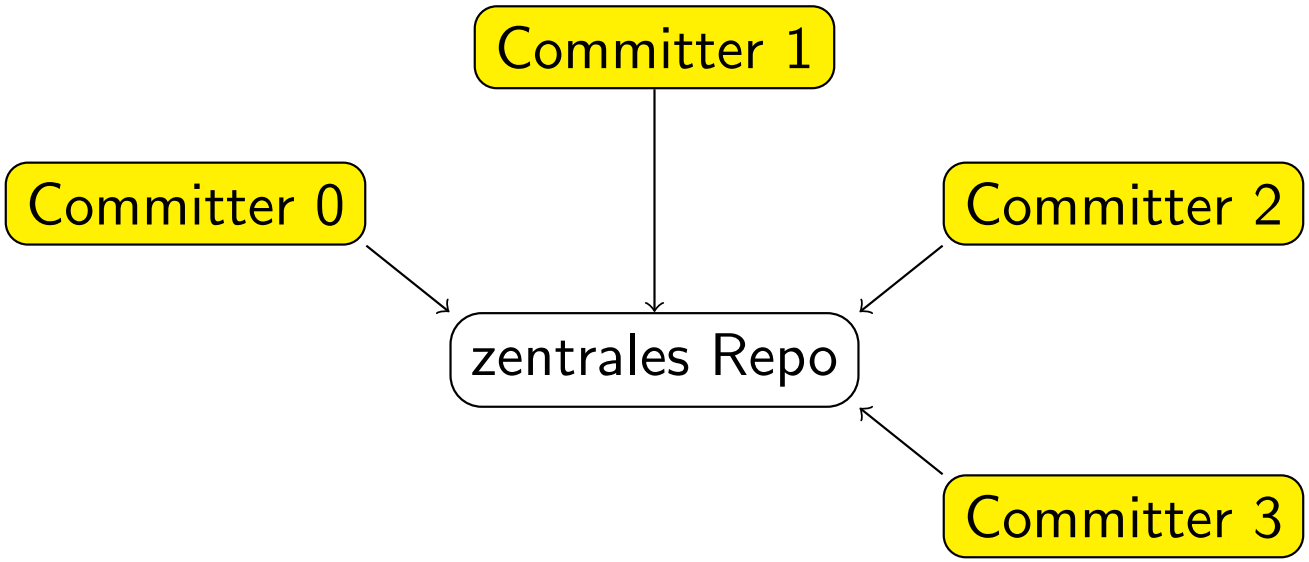
# Grundlagen: Typen von VCSen

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

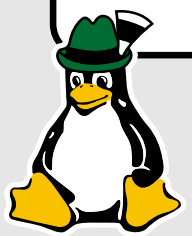
- lokale Versionierung
- zentrale Versionierung
- dezentrale Versionierung



# Grundlagen: Typen von VCSen



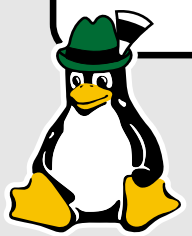

. . .



# Grundlagen: Typen von VCSen

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

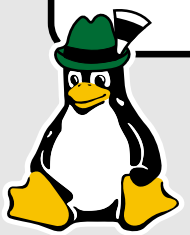
- lokale Versionierung
- zentrale Versionierung
- dezentrale Versionierung



Grundlagen: Was ist Versionskontrolle?

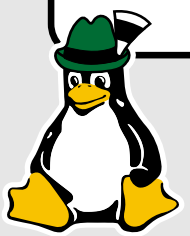
Dezentrale Versionskontrolle

Arbeiten mit Git

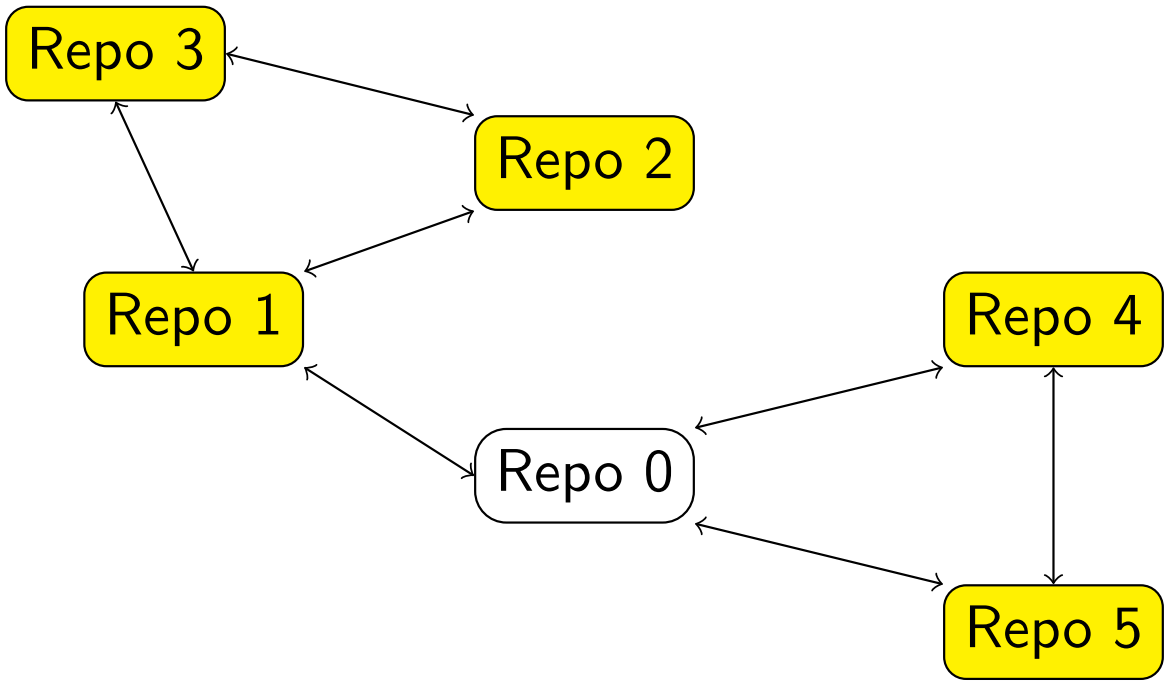


# Workflow in OpenSource Projekten

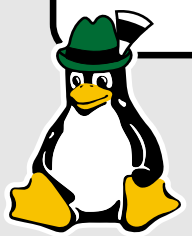
- üblicherweise ein/wenige Hauptentwickler/Projektleiter
- viele Mitwirkende (versch. Umfang/Arbeitsgebiet)
- ggf. Subsystem-Verantwortliche;
  Entwickler mit mehreren Arbeitsrechnern
- ein „zentraler“/„offizielles“ Repository
- temporäre und Feature-Branches



# Workflow in OpenSource Projekten

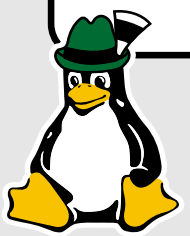



. . .

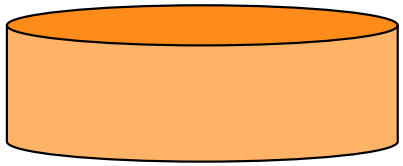


# Grundlagen von dezentralen VCSen

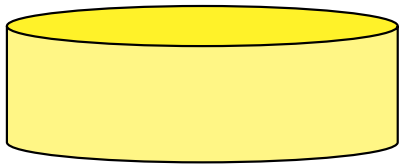
- „Peer-to-Peer“ Ansatz
- jede Arbeitskopie bringt ein komplettes Repository mit (Klon)
- gearbeitet wird auf lokalem Repository
  ⇒ kein Netzwerk-Zugriff nötig
  ⇒ Operationen schnell
  ⇒ Offline-Arbeit möglich
- automatisches „Backup“ durch Repository-Klons
- Zusammenführen meist auf Basis eines „Web-of-Trust“



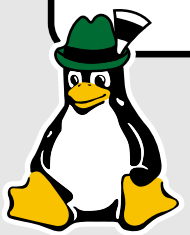
# Arbeitsweise



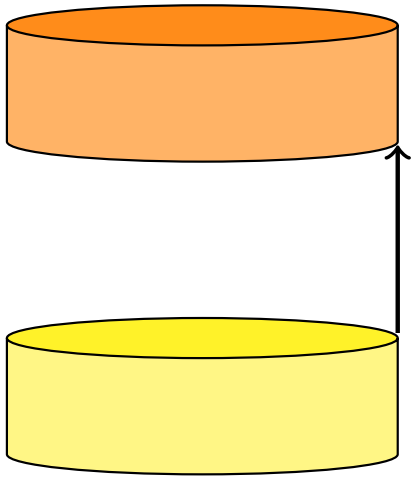
Arbeitskopie

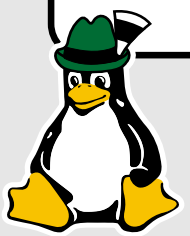


Repository

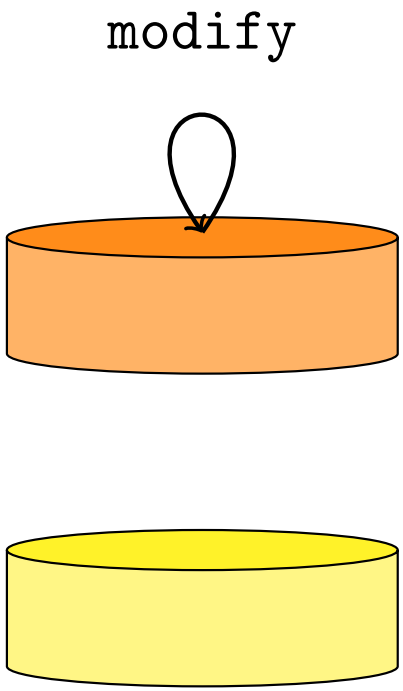


# Arbeitsweise



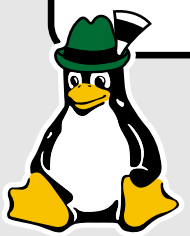



# Arbeitsweise

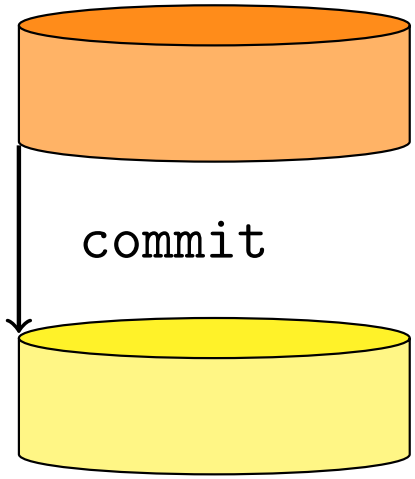



Arbeitskopie

Repository

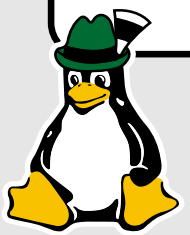


# Arbeitsweise

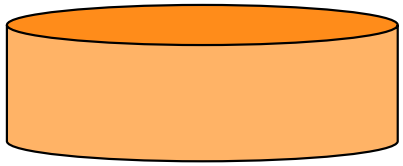



Arbeitskopie

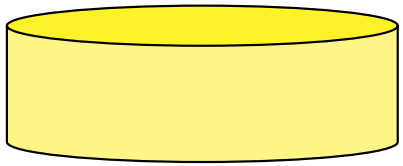
Repository



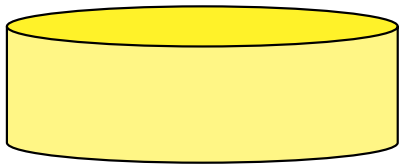
# Arbeitsweise



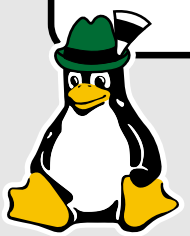
Arbeitskopie



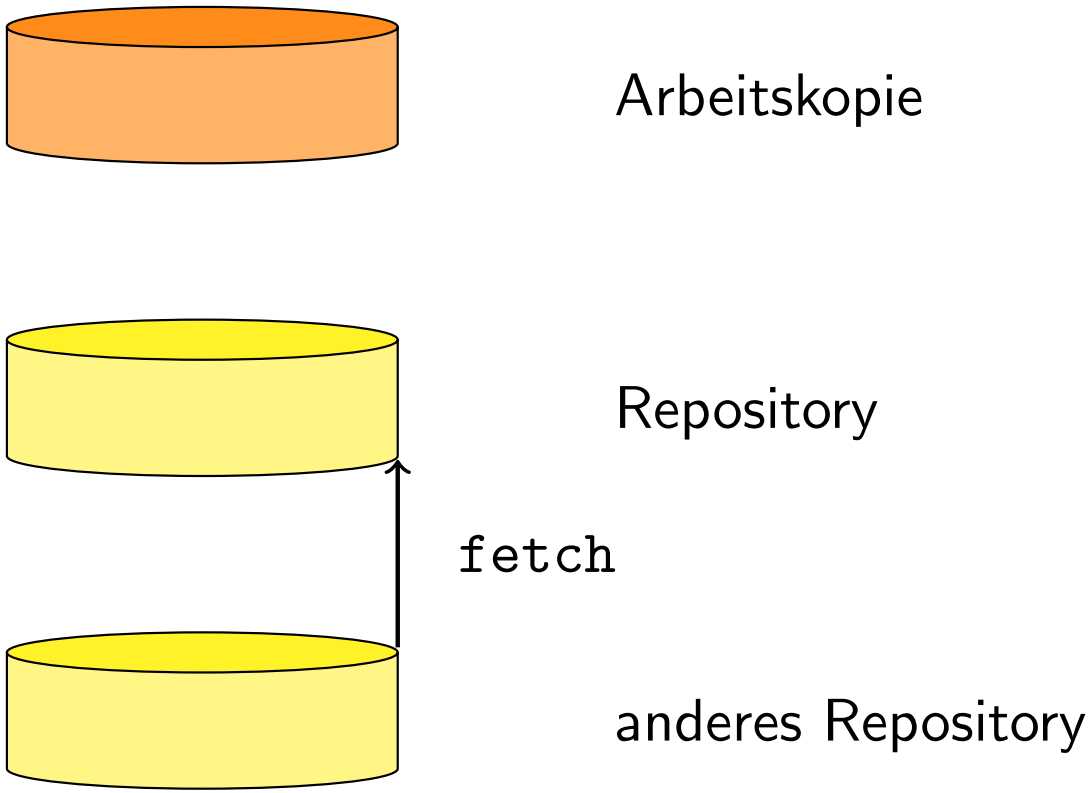
Repository

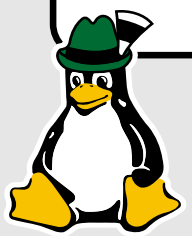


anderes Repository

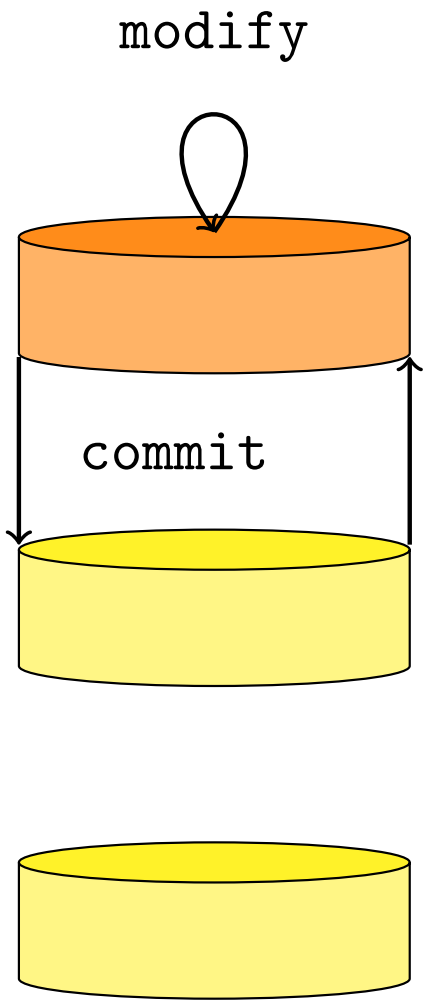


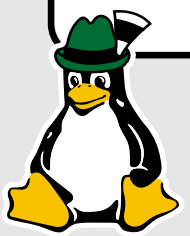
# Arbeitsweise



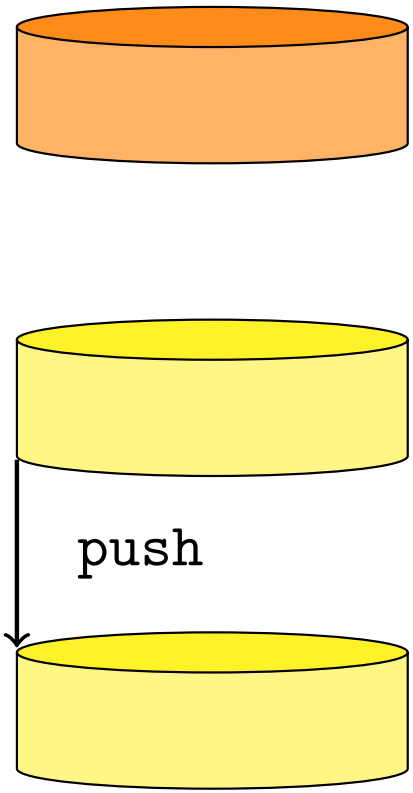



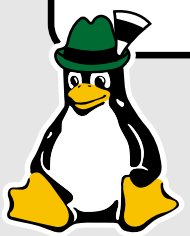
# Arbeitsweise



# Arbeitsweise

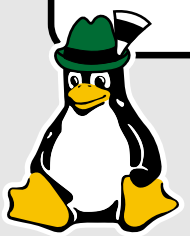


Grundlagen: Was ist Versionskontrolle?

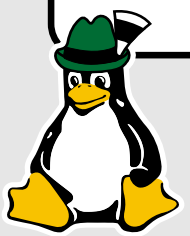
Dezentrale Versionskontrolle

Arbeiten mit Git



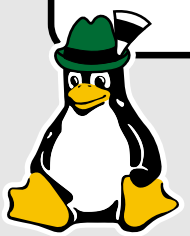
# Git: Übersicht

- `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, . . . )



# Arbeiten mit Git: Grundlagen

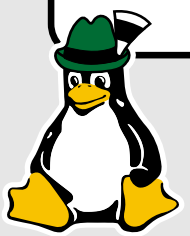
- ca. 150 einzelne Befehle
- „Porcelains“ und „Plumbing“
- Dokumentation als Manpages — `git(7)`
- `git help`, `git <command> -h`
- Benutzer Handbuch: `http://www.kernel.org/pub/software/scm/git/docs/user-manual.html`
- „Git Community Book“: `http://book.git-scm.com/`
- Buch „Pro Git“: `http://progit.org/book/`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



# Datenhaltung: Git Objektdatenbank

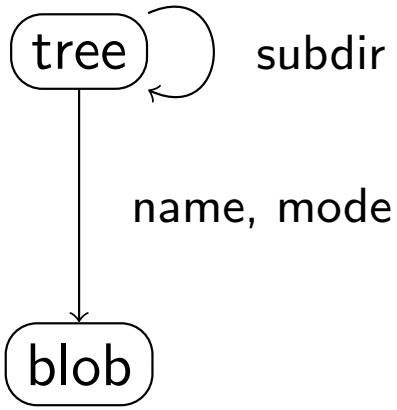
- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe

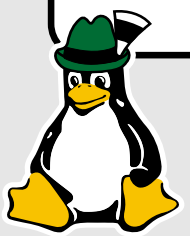
blob



# Datenhaltung: Git Objektdatenbank

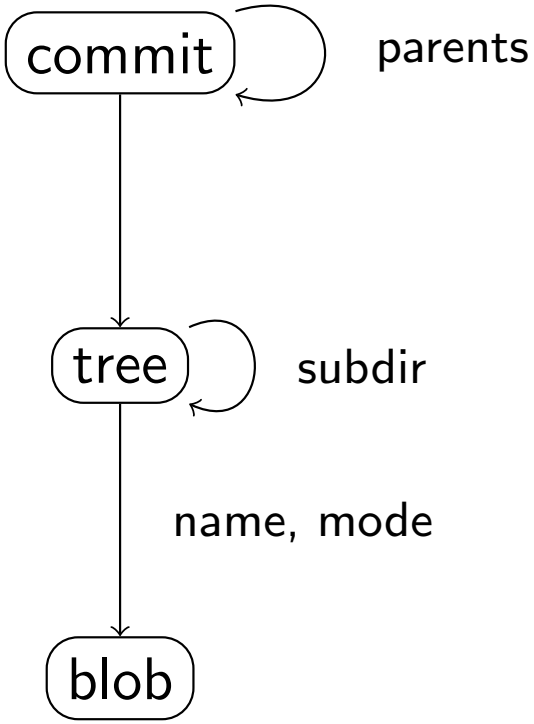
- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe

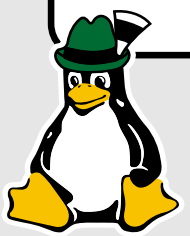


# Datenhaltung: Git Objektdatenbank

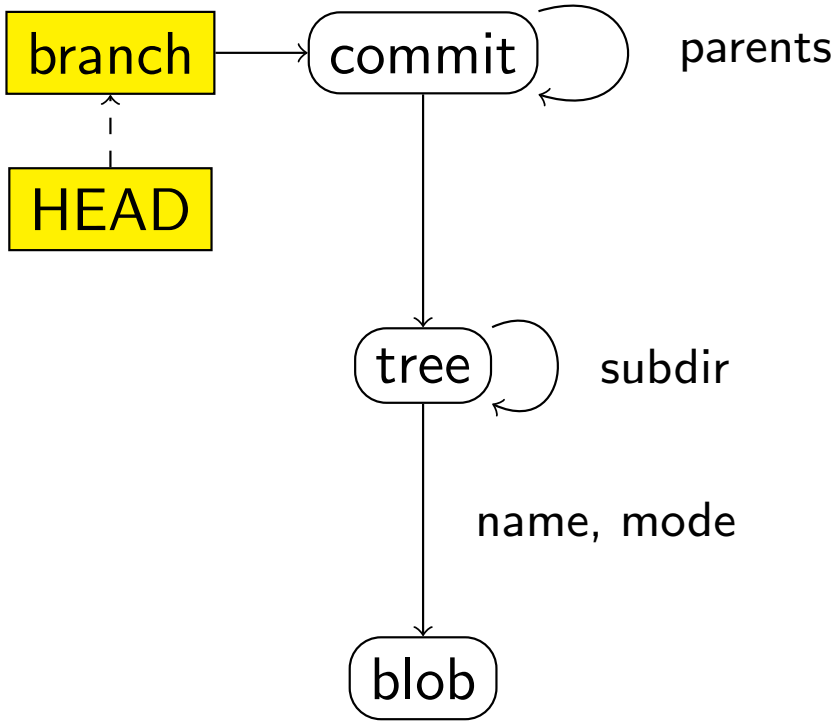
- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe

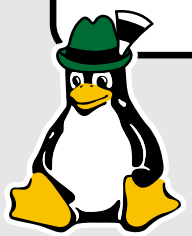


# Datenhaltung: Git Objektdatenbank

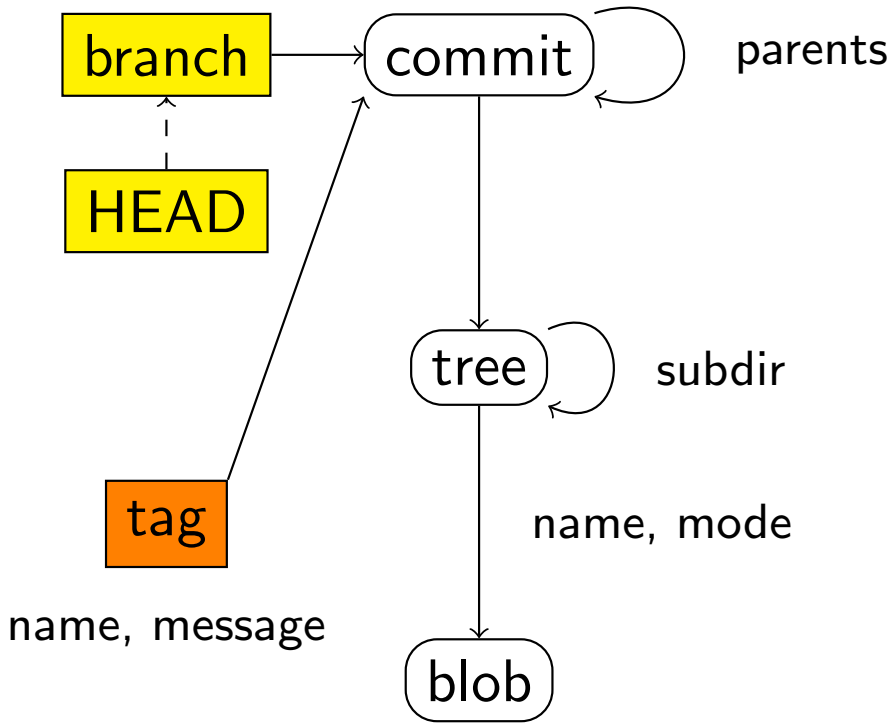
- DAG (directed acyclic graph)
- Objekte identifiziert durch SHA-1 Summe

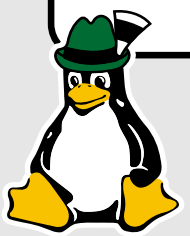


# Datenhaltung: Git Objektdatenbank

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

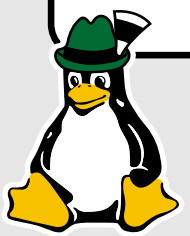


# Git konfigurieren

## Sich Git vorstellen

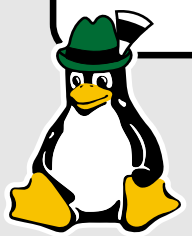
- `git config --global user.name <Dein Name>`
- `git config --global user.email <du@deine-domain.tld>`
- → Benutzerinformationen für Commit-Metadaten



# Git konfigurieren

## Bunt und in Farbe

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



# Git konfigurieren

## Bunt und in Farbe

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

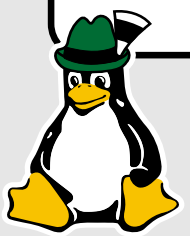
## Weitere nützliche Optionen

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

# Repositories erstellen

## Neues, leeres Repository

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



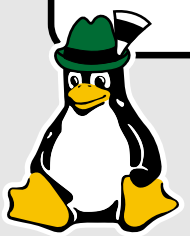
# Repositories erstellen

## Neues, leeres Repository

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

## Bestehendes Repository „klonen“

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

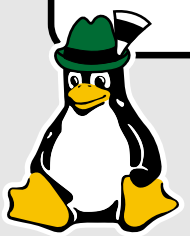


# Änderungen vornehmen

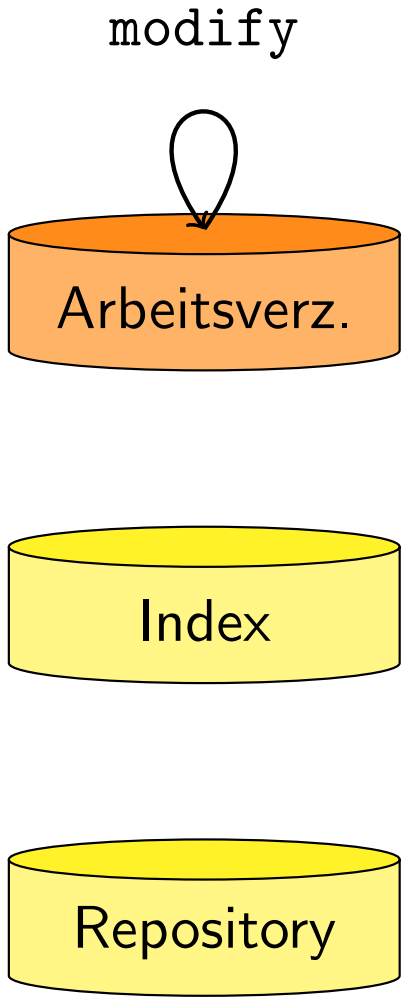
## Ändern

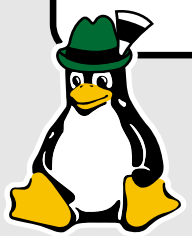
```
$ vim foo bar
$ git add foo bar
```

- `add`, `rm`, `mv`

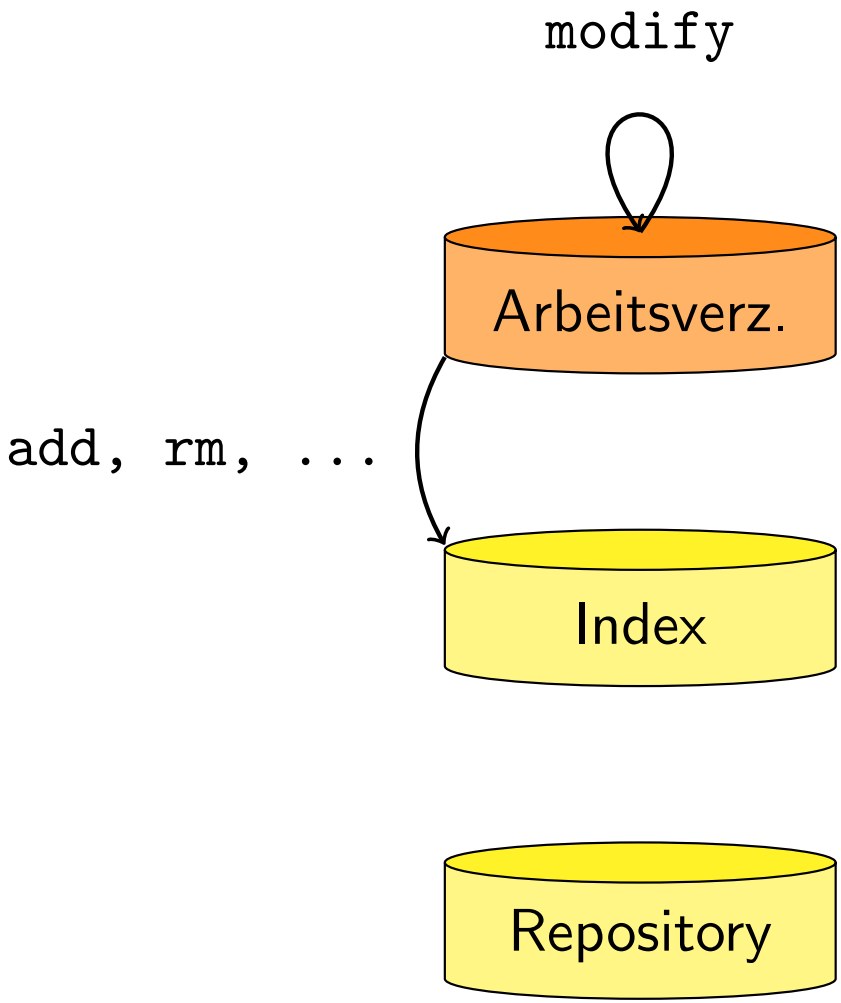


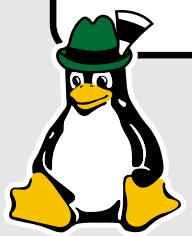
# Das Mysterium „Index“



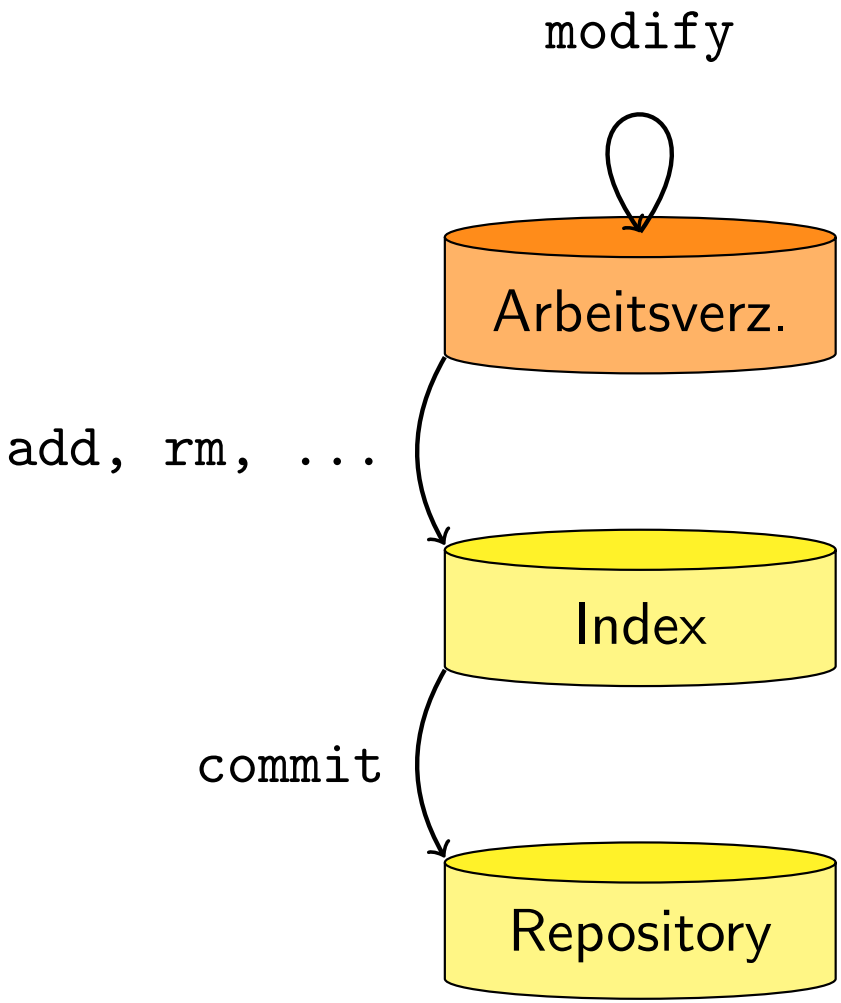



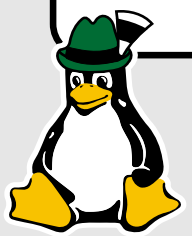
# Das Mysterium „Index“



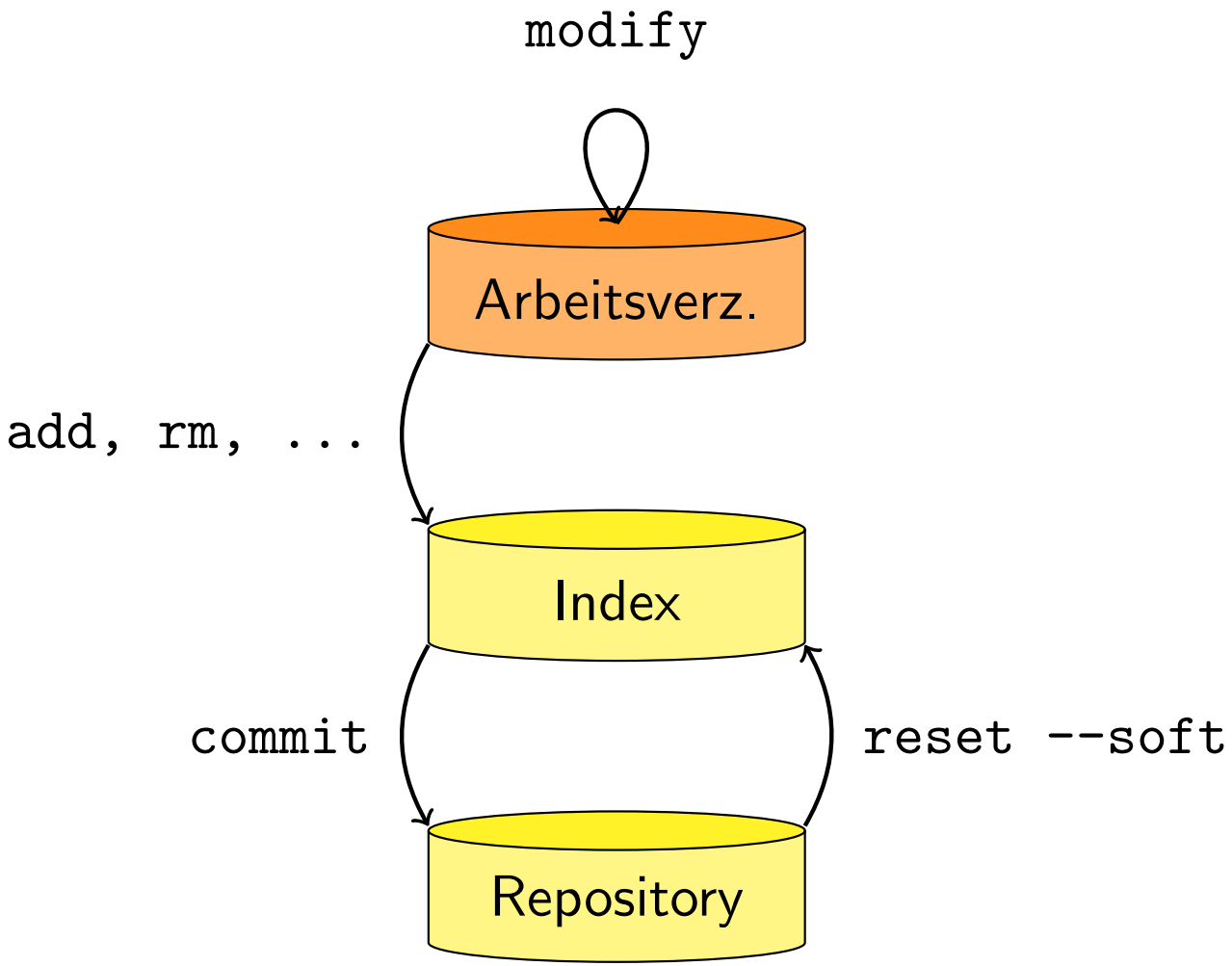



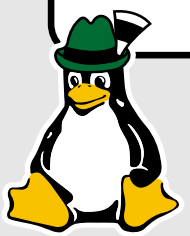
# Das Mysterium „Index“



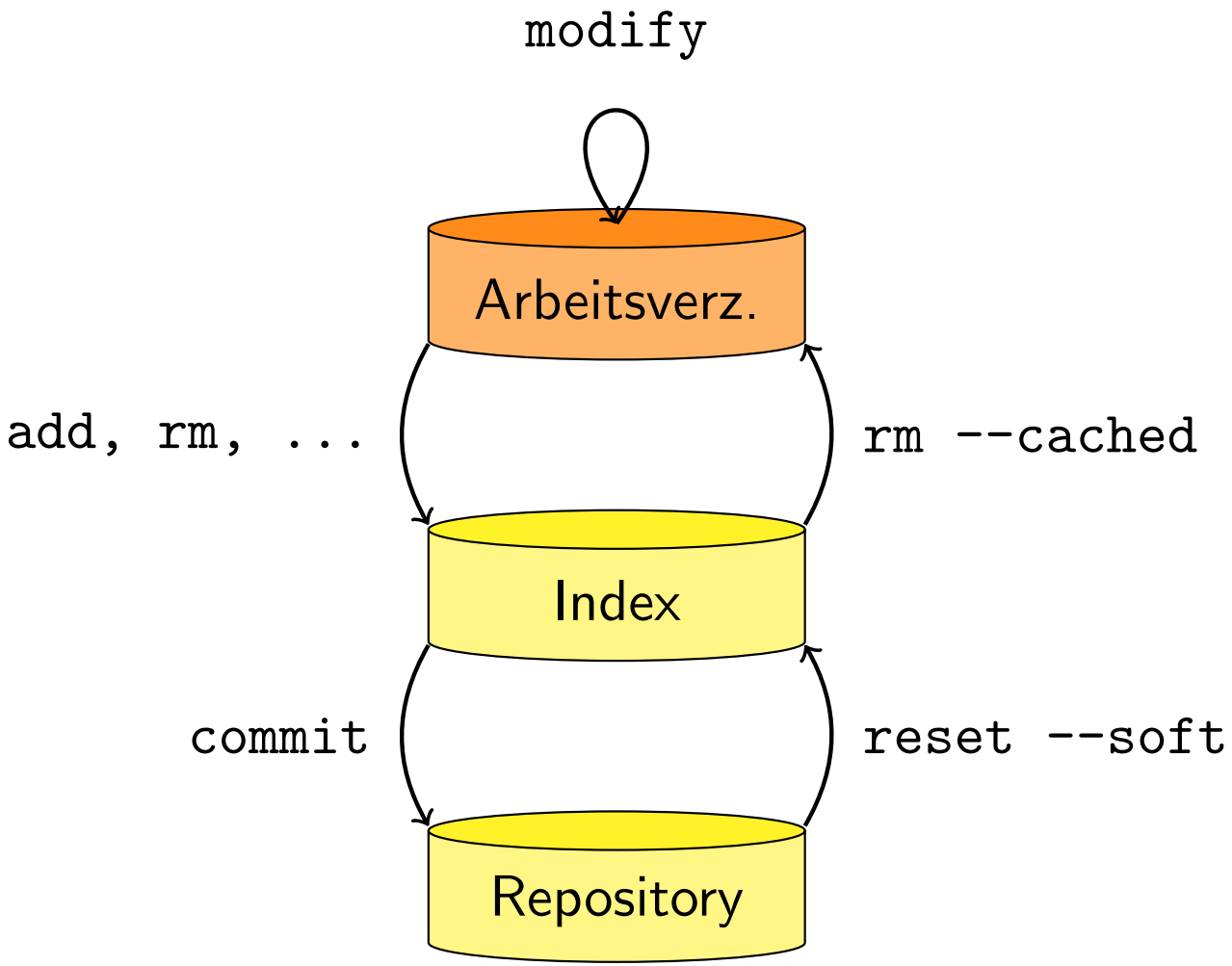



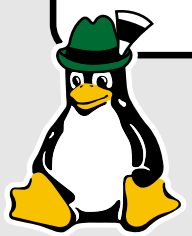
# Das Mysterium „Index“



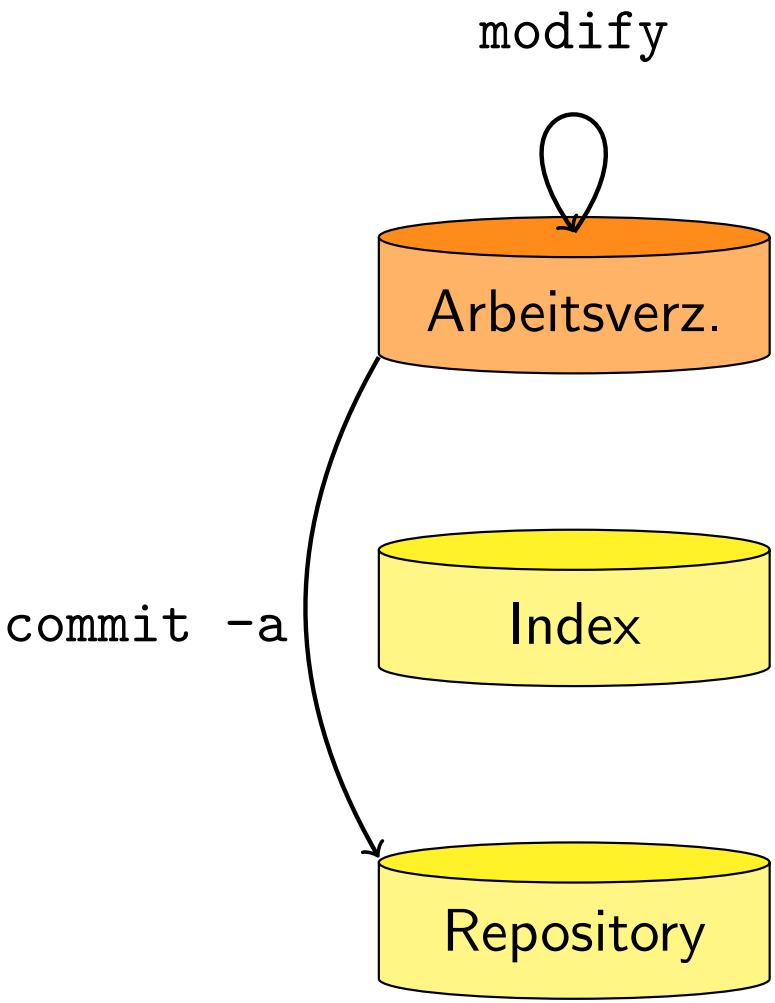



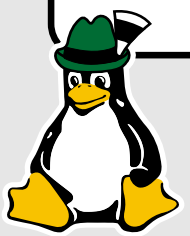
# Das Mysterium „Index“



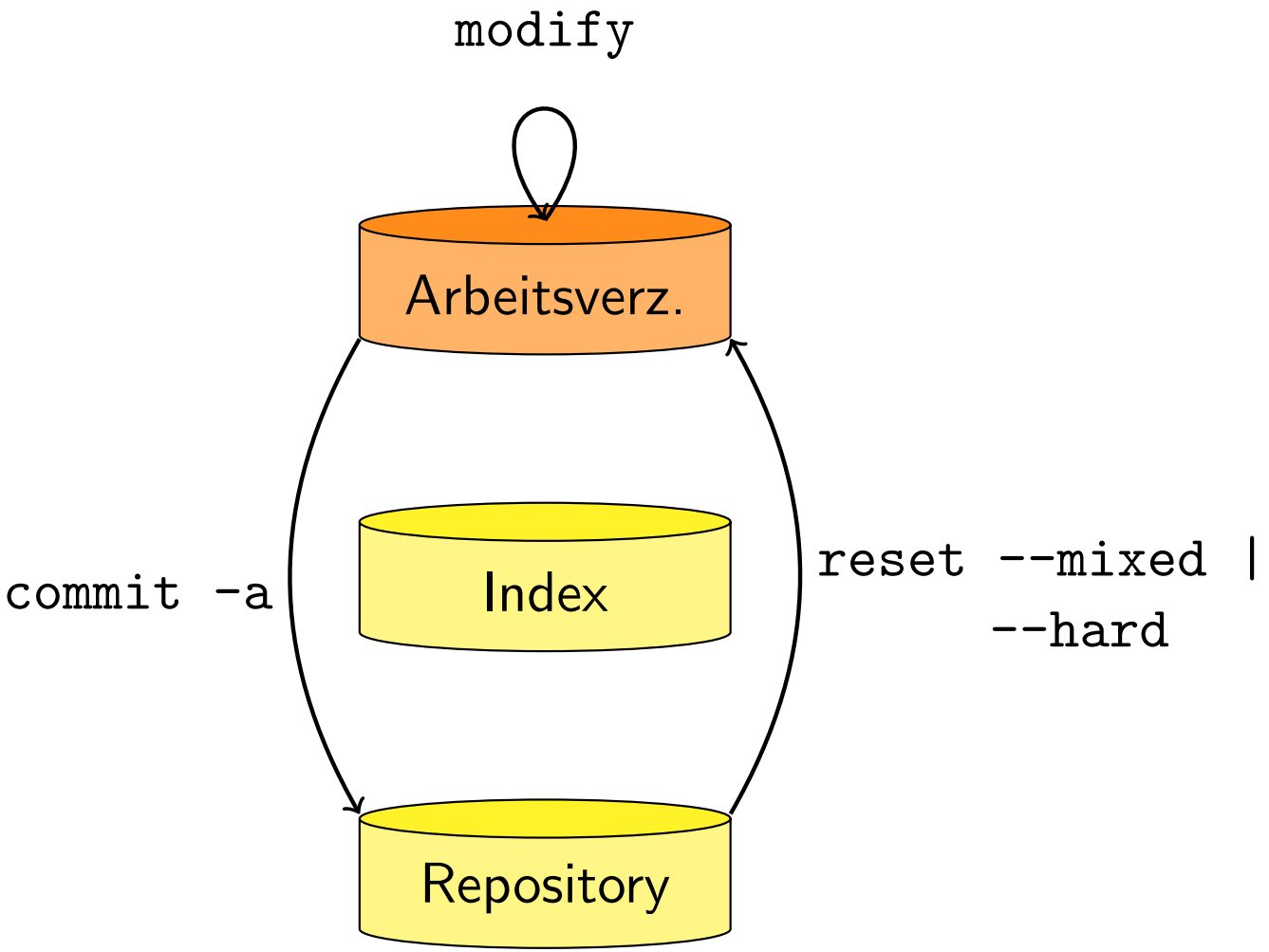



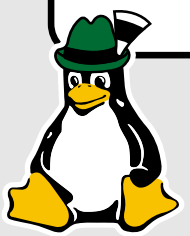
# Das Mysterium „Index“



# Das Mysterium „Index“



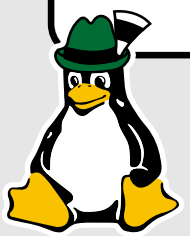
# Änderungen aufzeichnen

## Geschichte fortführen/ändern

```
$ git commit
```

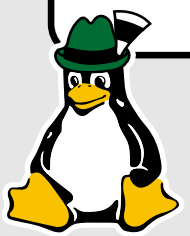
```
$ git reset --hard HEAD^
```

- `reset`, `revert`, `checkout`



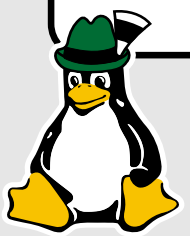
# Exkurs: Commit Meldungen schreiben

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



# Exkurs: Commit Meldungen schreiben

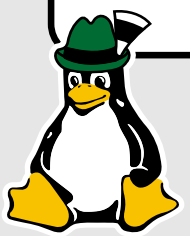
```
gitk: Speed up resolution of short SHA1 ids

On large repositories such as the Linux kernel, it can take quite
a noticeable time (several seconds) for gitk to resolve short
SHA1 IDs to their long form. This speeds up the process by
maintaining lists of IDs indexed by the first 4 characters of the
SHA1 ID, speeding up the search by a factor of 65536 on large
repositories.

Signed-off-by: Paul Mackerras <paulus@samba.org>
```



# Aus der Geschichte lernen

## Status der Arbeitskopie

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



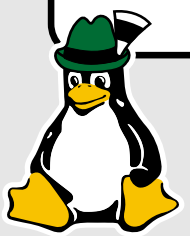
# Aus der Geschichte lernen

## Status der Arbeitskopie

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

## Historie betrachten

```
$ git log
$ tig
```



# Aus der Geschichte lernen

## Status der Arbeitskopie

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

## Historie betrachten

```
$ git log
$ tig
```

## Objekte betrachten

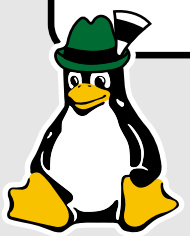
```
$ git show
$ git show HEAD:foo                    (siehe gitrevisions(7))
```

- Commits, Trees, Blobs, Tags

# 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

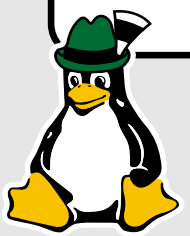


# Branching und Merging

- 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>
```

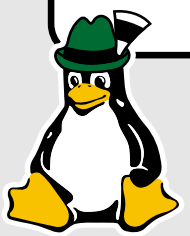


# Branching und Merging

- 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

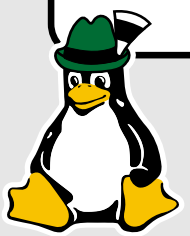
```
$ git checkout -b <Name>

$ git branch
  master
* <Name>
```



# Branching und Merging

## Branches zusammenführen

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



# Branching und Merging

## Branches zusammenführen

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

## 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`

# Arbeiten mit anderen Repositories

## Repository klonen

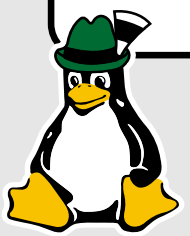
```
$ git clone <rep>
```

## Austauschen von Änderungen

```
$ git pull
$ git push
```

Alternativ:

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

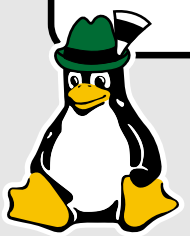


# Arbeiten mit anderen Repositories

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

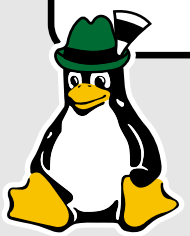
## Arbeiten mit „remotes“

```
$ git remote add <Name> URL
$ git remote update <Name> $ git push <Name> # ggf. zusätzlich
Branch angeben
```



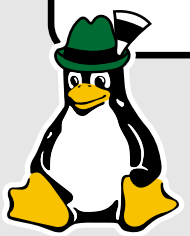
# Repository URLs

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



# Frontends

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

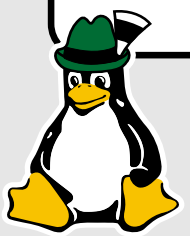


# Spiel, Spaß und Spannung mit Git

Vielen Dank für die Aufmerksamkeit!

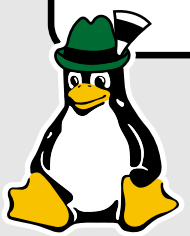
## Gibt es Fragen?

Kontakt:
Sebastian „tokkee“ Harl
<tokkee@debian.org>

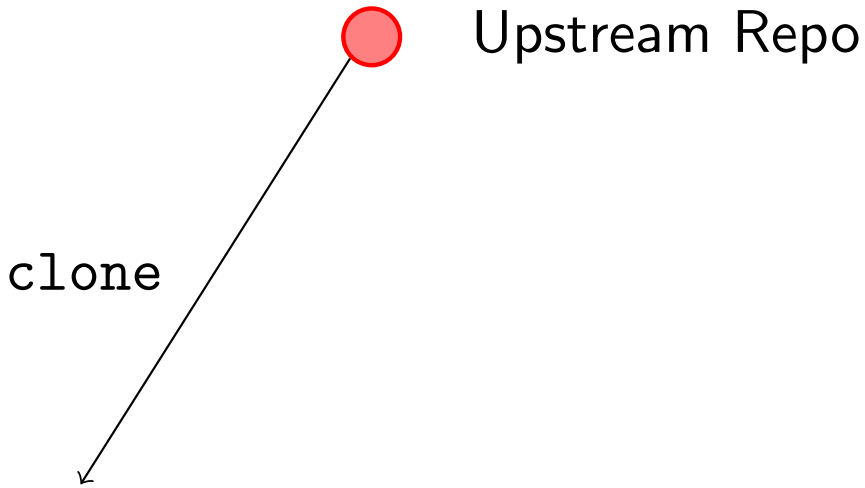


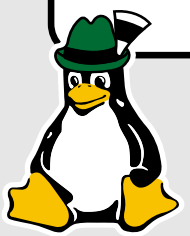
# Ein Beispiel . . .

- Upstream Repo

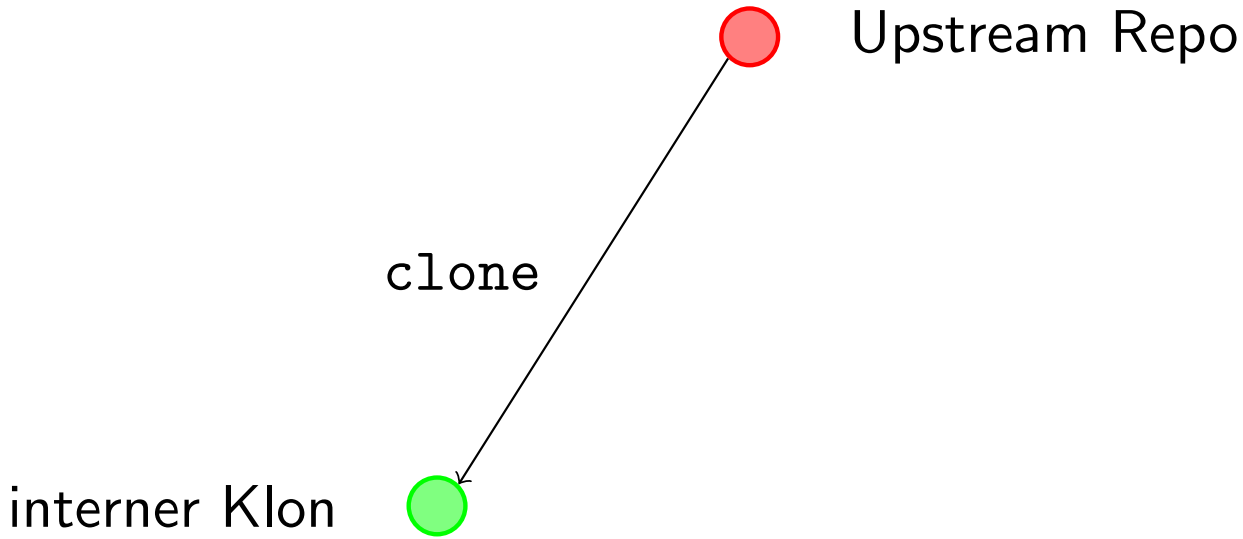


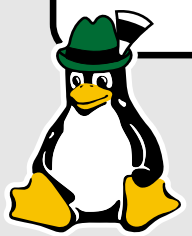
# Ein Beispiel ...



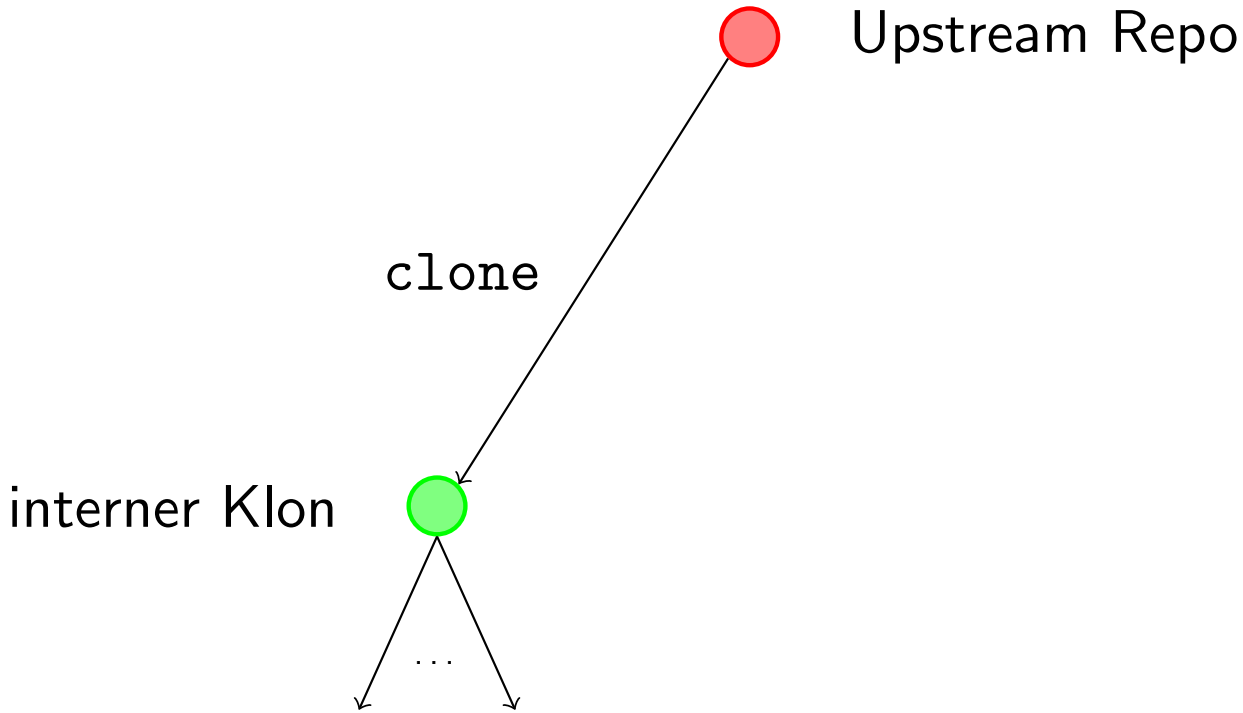



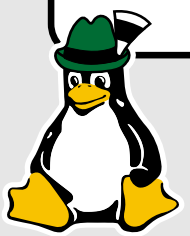
# Ein Beispiel ...



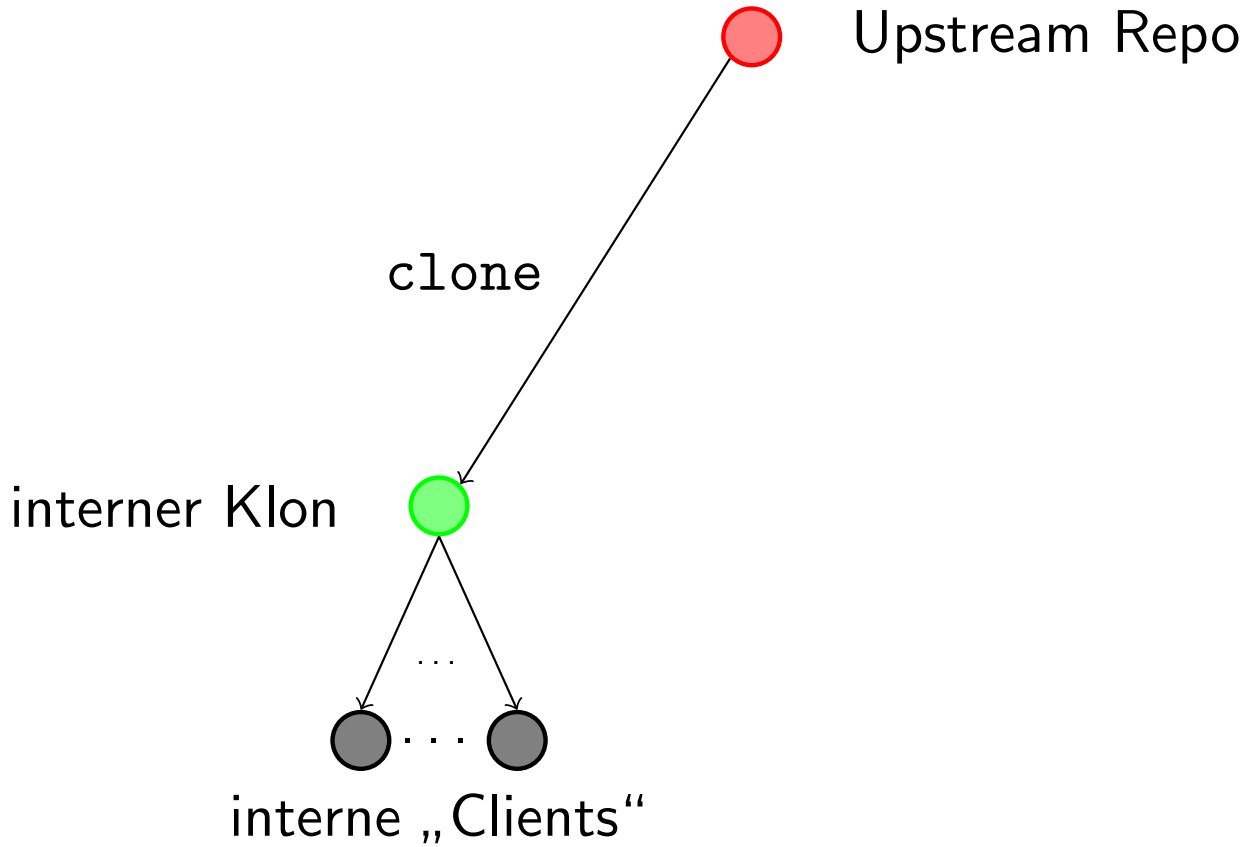



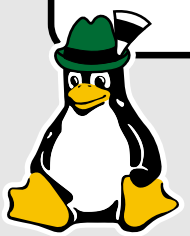
# Ein Beispiel . . .



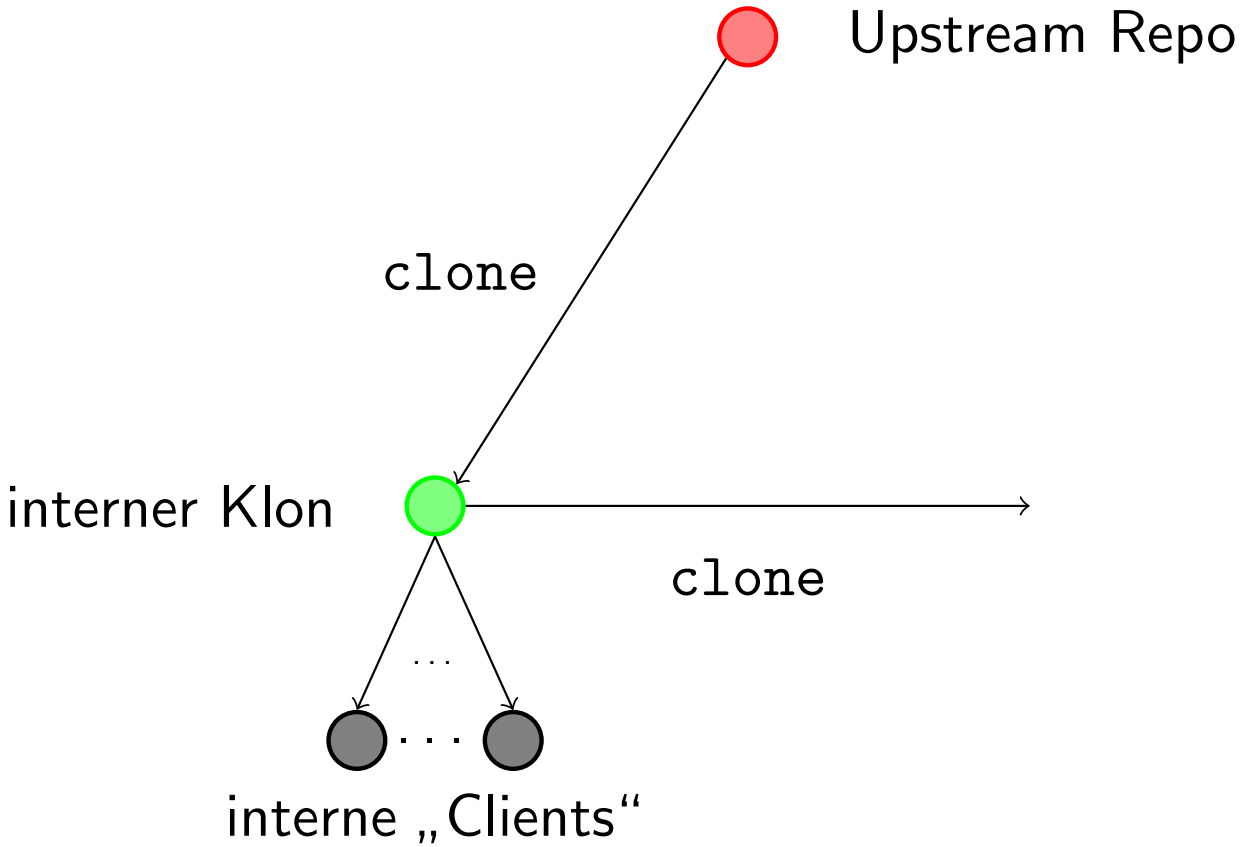



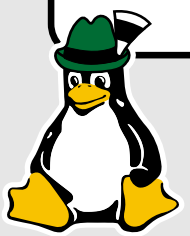
# Ein Beispiel . . .



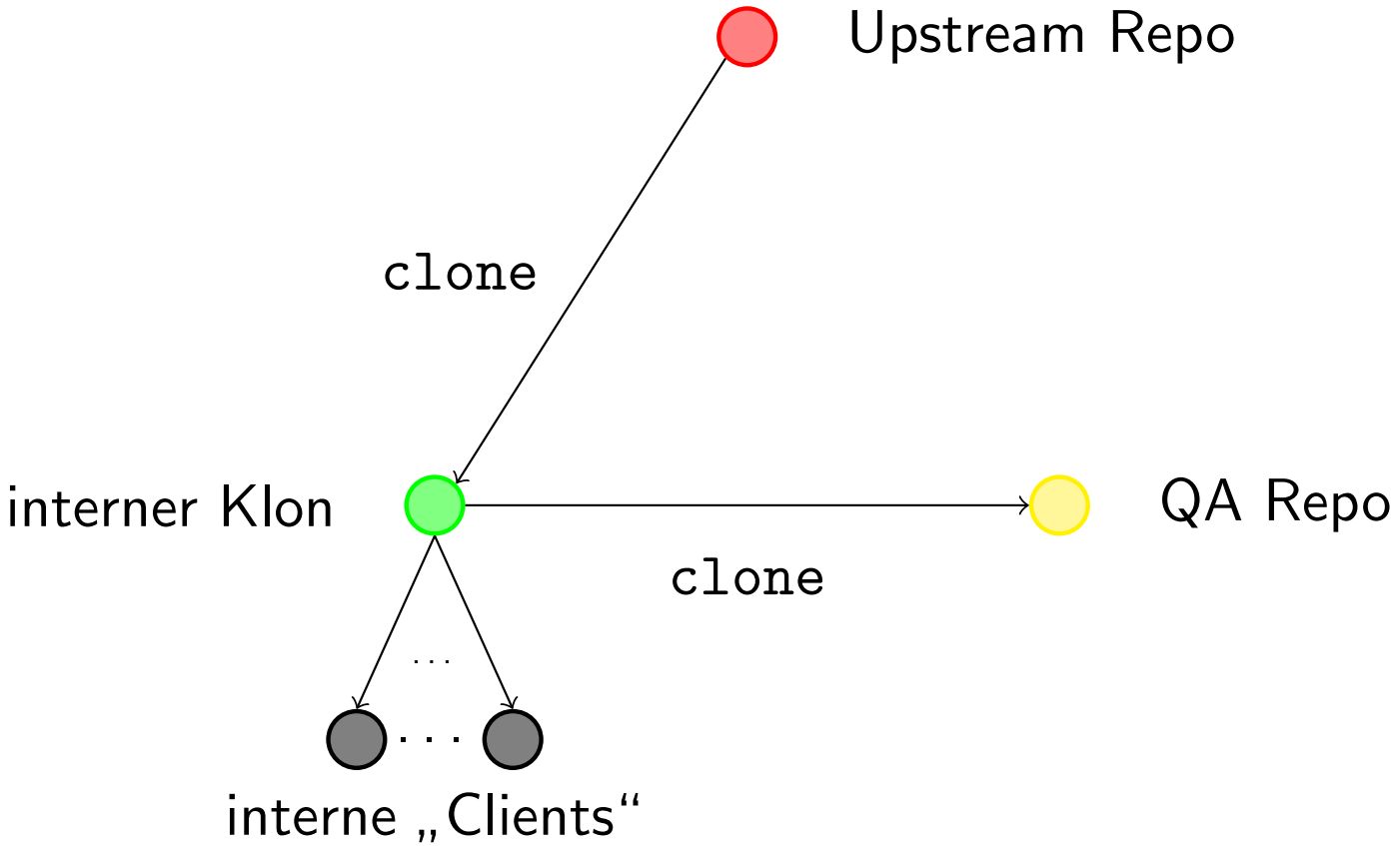



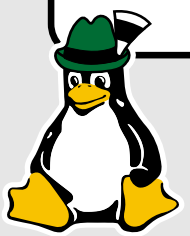
# Ein Beispiel . . .



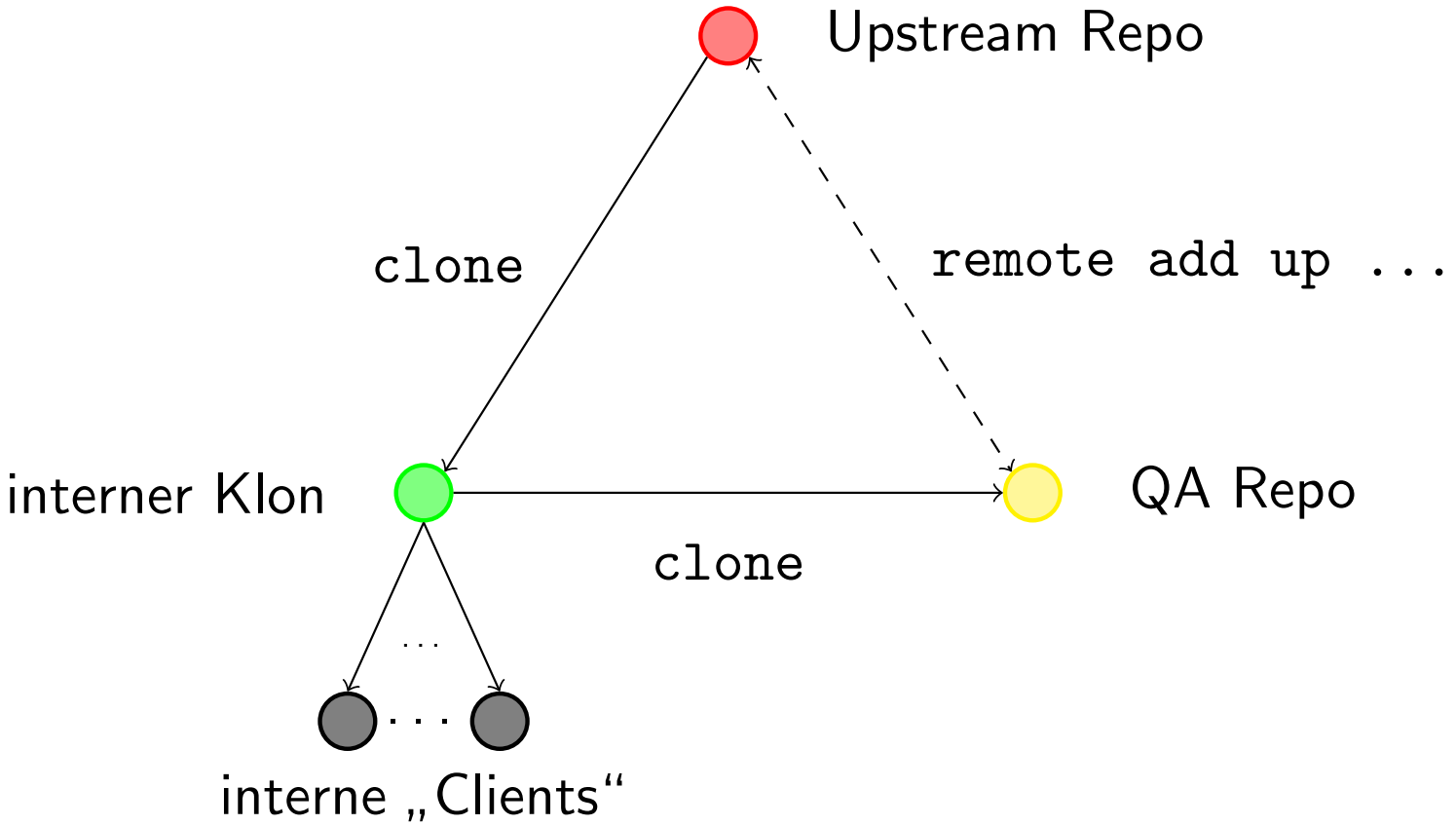



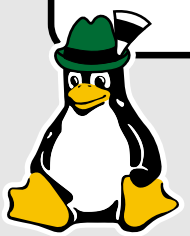
# Ein Beispiel . . .



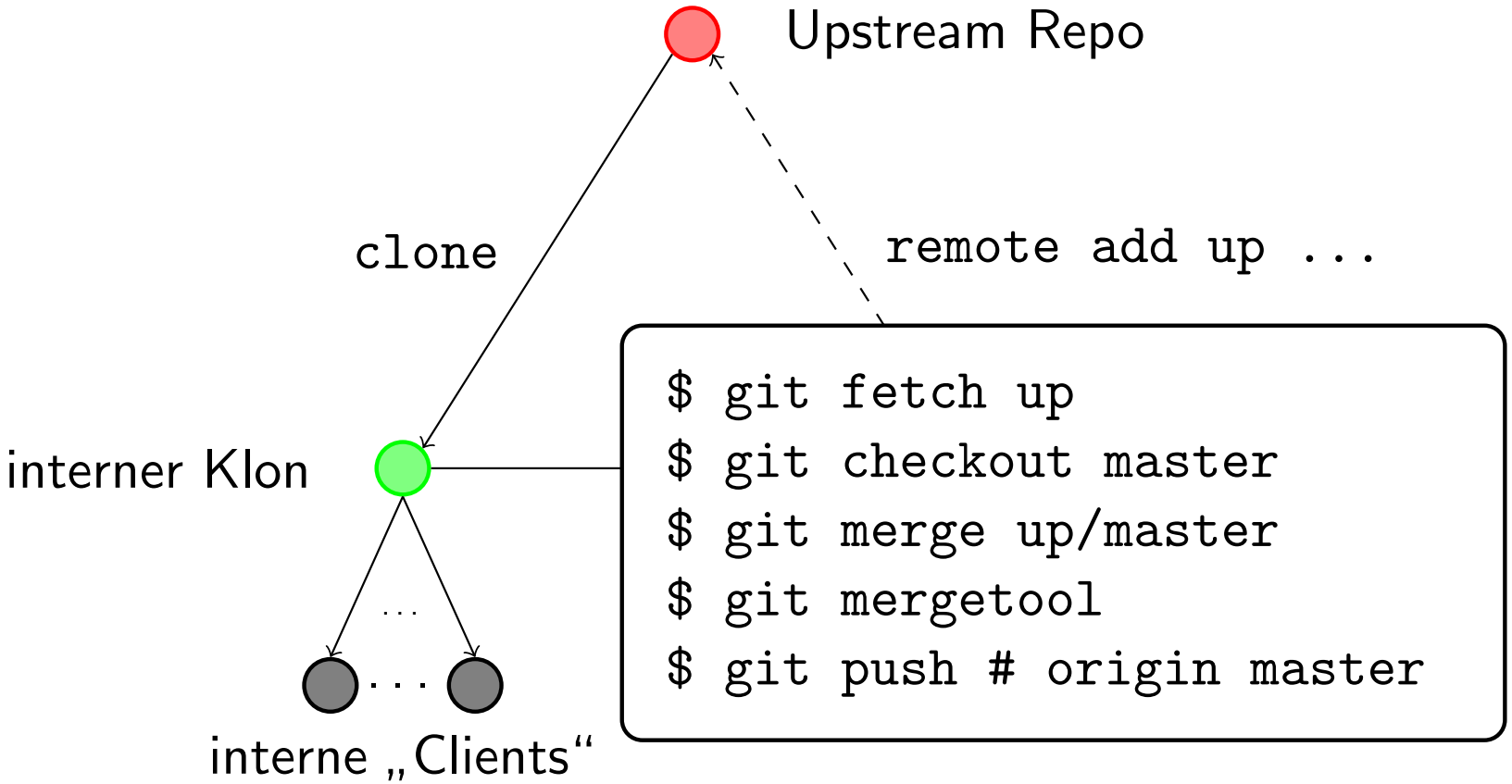



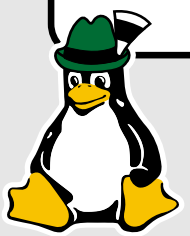
# Ein Beispiel ...



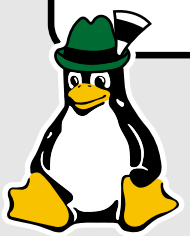
# Ein Beispiel ...



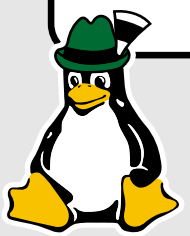
# Goodies

- `git reflog`



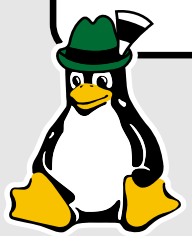
# Goodies

- `git reflog`
- `git rebase -i`



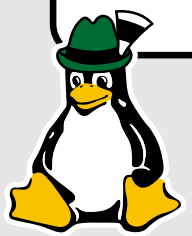
# Goodies

- `git reflog`
- `git rebase -i`
- `git commit --amend`



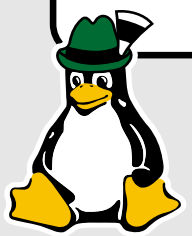
# Goodies

- `git reflog`
- `git rebase -i`
- `git commit --amend`
- `git add -p`



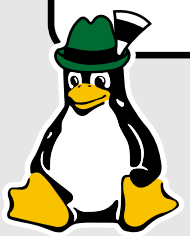
# Goodies

- `git reflog`
- `git rebase -i`
- `git commit --amend`
- `git add -p`
- `git stash`



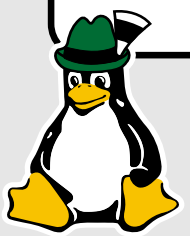
# Goodies

- `git reflog`
- `git rebase -i`
- `git commit --amend`
- `git add -p`
- `git stash`
- `git bisect`



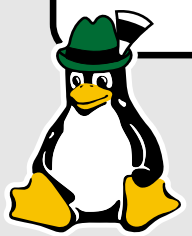
# Goodies

- `git reflog`
- `git rebase -i`
- `git commit --amend`
- `git add -p`
- `git stash`
- `git bisect`
- `git cherry` / `git-wtf`



# Goodies

- `git reflog`
- `git rebase -i`
- `git commit --amend`
- `git add -p`
- `git stash`
- `git bisect`
- `git cherry` / `git-wtf`
- `git diff --color-words`



# Goodies

- `git reflog`
- `git rebase -i`
- `git commit --amend`
- `git add -p`
- `git stash`
- `git bisect`
- `git cherry` / `git-wtf`
- `git diff --color-words`
- `git svn`