

Git Workshop

VALUG - 8. April 2011

Florian Preinstorfer, Wolfgang Silbermayr

08.04.2011



Inhalt

Versionsverwaltung

Einstieg in git

Branching und Merging

Collaboration

Versionsverwaltung

Einstieg in git

Branching und Merging

Collaboration

Was ist Versionsverwaltung

Aus der Wikipedia:

Eine Versionsverwaltung ist ein System, das zur **Erfassung von Änderungen an Dokumenten oder Dateien verwendet wird**. Alle Versionen werden in einem **Archiv** mit Zeitstempel und Benutzerkennung gesichert und können später **wiederhergestellt** werden. Versionsverwaltungssysteme werden typischerweise in der Softwareentwicklung eingesetzt um Quelltexte zu verwalten. Versionsverwaltung kommt auch bei Büroanwendungen oder Content-Management-Systemen zum Einsatz.

Versionsverwaltungs-Arten I

- Lokale Versionsverwaltung:
 - Single-User
 - System kennt keine anderen Arbeitskopien oder Benutzer
 - Mittlerweile kaum noch Bedeutung
 - Beispiel: rcs

Versionsverwaltungs-Arten II

- Zentrale Versionsverwaltung
 - Ein „heiliges“ Repository für alle Benutzer
 - Repository meist auf einem Server
 - Jedes Einchecken von Änderungen benötigt Zugriff auf das Repository, also Netzwerkverbindung
 - Forks sind extrem aufwändig
 - Beispiele: CVS, Subversion

Versionsverwaltungs-Arten III

- Dezentrale Versionsverwaltung
 - Arbeitskopie beinhaltet automatisch Repository
 - Repositories können im Netz veröffentlicht werden
 - Veröffentlichte Repositories können geklont werden
 - Jede Arbeitskopie ist per Definition ein Fork
 - Beispiele: git, Bazaar, Mercurial, Monotone

Versionsverwaltung

Einstieg in git

Branching und Merging

Collaboration

Installation

- Linux:
 - Debian, Ubuntu: `[sudo] apt-get install git-core`
 - Arch: `[sudo] pacman -S git`
- Windows:
 - MySysGit: <http://code.google.com/p/msysgit>
 - TortoiseGit: <http://code.google.com/p/tortoisegit>
- Mac:
 - Git for OS X: <http://code.google.com/p/git-osx-installer>

initiale Konfiguration

- Identität festlegen:

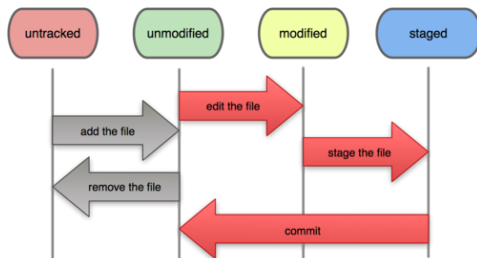
```
$ git config --global user.name "John_Doe"  
$ git config --global user.email johndoe@example.com
```

- Editor festlegen (optional):

```
$ git config --global core.editor whatever
```

Die möglichen Zustände einer Datei

File Status Lifecycle



Quelle: <http://progit.org>

- Nicht versioniert (**untracked**)
- Versioniert, aber unverändert (**unmodified**)
- Versioniert und verändert (**modified**)
- Versioniert, verändert und in der „staging area“ (**staged**)

Die ersten Schritte

- Initialisieren eines git Repositories

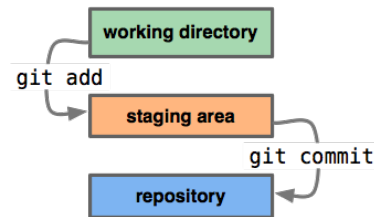
```
$ cd meinprojekt  
$ git init
```

- Dateien zur „staging area“ hinzufügen

```
$ git add <file>
```

- Änderungen der „staging area“ committen

```
$ git commit -m 'initial commit'
```



Quelle:

<http://whygitisbetterthanx.com>

Status betrachten

- Den aktuellen Status des Repositories betrachten

```
$ git status
```

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# modified: LICENSE
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working dir)
#
# modified: notes.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# test.txt
```

Ausgabe von git status

Änderungen betrachten

- Änderungen im Arbeitsverzeichnis betrachten

```
$ git diff
```

- Änderungen betrachten, die sich bereits in der „staging area“ befinden

```
$ git diff --cached
```

```
diff --git a/slides/sections/git.tex b/slides/sections/git.tex
index 12c5ea5..800639c 100644
@@ -1,4 +1,4 @@
-\\section{git}
+\\section{Einstieg in git}

\\begin{frame}
  \\tableofcontents[currentsection]
```

Ausgabe von git diff

typischer Workflow

- Bestehende Dateien verändern
- Neue Dateien zum Repository hinzufügen

```
$ git add <newfile>
```

- Den Status des Repository betrachten

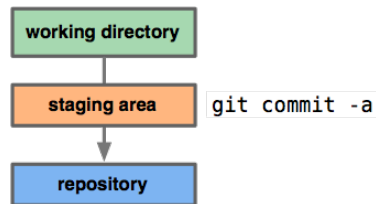
```
$ git status
```

- Die Änderungen betrachten

```
$ git diff  
$ git diff --cached
```

- Alle neuen und veränderten Dateien committen

```
$ git commit -a -m 'add foo'
```



Quelle:

<http://whygitisbetterthanx.com>

Historie betrachten I

- Historie in umgekehrt chronologischer Reihenfolge

```
$ git log
```

```
commit ebae10784096e1ce3c2d4a8f2dd4cff32edfeaa0
Author: Wolfgang Silbermayr <wolfgang@example.com>
Date:   Sat Mar 19 17:38:35 2011 +0100
```

```
    Add git-transport image
```

```
commit 7198128a60a2c5b20d428722fa0341e0aaaae206
Author: Florian Preinstorfer <florian@example.com>
Date:   Sat Mar 19 17:27:12 2011 +0100
```

```
    add hosting and collaboration frame
```


Historie betrachten II

- Git log erlaubt verschiedene Formatierungsoptionen
- Beispiel:

```
$ git log --pretty=format:'%h : %s' --topo-order --graph
```

```
*    ade7f24 : Merge branch 'master' of gitorious.org:valug/git-slides
|\
| * 0bb70f1 : change colors; add caption
| * eaf4a3b : use color in git status listing
| *    808256f : Merge branch 'master' of gitorious.org:valug/git-slides
|  |\
| * | e63e26a : add status file
* | | 714815a : Add branching commands
|  | /
| / |
* | 1ad271e : Some special cases for diff lstlisting highlighting
```

Grafische git Tools I

The screenshot shows the gitk interface with the commit history on the left and a diff view on the right. The commit history lists several commits, including 'add a frame about tagging' and 'Merge branch 'master' of gitorious.org:valug/git-slides'. The diff view shows the changes in the file 'slides/sections/git.tex', highlighting the addition of a frame about tagging.

SHA1 ID: c7dbcd4a200f32adf6bfe968ed947488bb3801e6

Suche: nächste vorige Version nach Beschreibung: Exakt Überschrift

◆ Vergleich ◆ Alte Version ◆ Neue Version Kontextzeilen: 3 Leerzeichenänderungen ignorieren

slides/sections/git.tex

```
index 595e69f..f954d3f 100644
@@ -127,4 +127,31 @@ two together.
 \end{lstlisting}
 \end{frame}

+ \begin{frame}[fragile]{Tipps und Tricks}
+ \begin{itemize}
+ \item Tagging
+ \begin{itemize}
+ \item verpasst einem bestimmten Versionsstand eine Markierung
+ \item wird oft verwendet, um Releases zu markieren (Bsp: \texttt{v1.1})
+ \item ermöglicht das schnelle Arbeiten mit unterschiedlichen Versionen des Repositories
+ \end{itemize}
+ \end{itemize}
+ \item alle existierenden Tags anzeigen
+ \begin{itemize}
+ \item git tag
+ \end{itemize}
+ \item einen einfachen Tag erstellen
+ \begin{itemize}
+ \item git tag v1.1
+ \end{itemize}
+ \item einen ausführlichen Tag erstellen (mit Name, commit message, \ldots)
+ \begin{itemize}
```

gitk

The screenshot shows the git GUI interface with the commit history on the left and a diff view on the right. The commit history lists several commits, including 'add a frame about tagging' and 'Merge branch 'master' of gitorious.org:valug/git-slides'. The diff view shows the changes in the file 'slides/sections/git.tex', highlighting the addition of a frame about tagging.

Chronik Einspielen

Zweig: master

Betreff	Autor	Datum
master gitorious/master add a frame about tagging	Florian Preinstorfer	Sam 19 Mär 2011 12:32:44 CE
Merge branch 'master' of gitorious.org:valug/git-slides	Wolfgang Silbermayr	Sam 19 Mär 2011 12:04:10 CE
ExampleTag Add description of different VCS types	Wolfgang Silbermayr	Sam 19 Mär 2011 12:03:23 CE
rework notes	Florian Preinstorfer	Sam 19 Mär 2011 11:57:00 CE
remove TODO	Florian Preinstorfer	Sam 19 Mär 2011 11:26:52 CE
use better arguments for good commit messages	Florian Preinstorfer	Sam 19 Mär 2011 11:25:00 CE

Details Änderungen Dateien

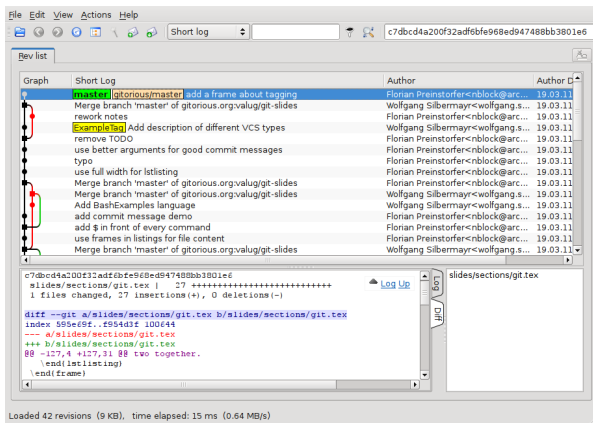
slides/sections/git.tex

```
diff --git a/slides/sections/git.tex b/slides/sections/git.tex
index 595e69f..f954d3f 100644
@@ -127,4 +127,31 @@ two together.
 \end{lstlisting}
 \end{frame}

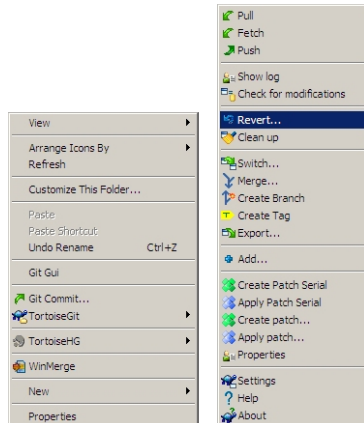
130 + \begin{frame}[fragile]{Tipps und Tricks}
131 + \begin{itemize}
132 + \item Tagging
133 + \begin{itemize}
134 + \item verpasst einem bestimmten Versionsstand eine Markierung
135 + \item wird oft verwendet, um Releases zu markieren (Bsp: \texttt{v1.1})
136 + \item ermöglicht das schnelle Arbeiten mit unterschiedlichen Ver
137 + \end{itemize}
138 + \end{itemize}
```

gitg

Grafische git Tools II



qgit



tortoisegit

Tipps und Tricks I

- Keine temporären Dateien in das git Repository aufnehmen
 - Erschwert das effiziente Arbeiten mit git
 - Bläht das Repository unnötig auf

```
*.swp          #alle swp Dateien  
doc/*.aux      #alle aux Dateien im Verzeichnis doc/  
tmp/**/*       #alle Dateien im Verzeichnis tmp/
```

Inhalt der Datei .gitignore

Tipps und Tricks II

- „Gute“ commit messages ...
 - Vereinfachen die Zusammenarbeit
 - Ermöglichen das schnelle Verfolgen von Änderungen

```
Short (50 chars or less) summary of changes
```

```
More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body. The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.
```

Quelle: <http://tbagery.com/2008/04/19/a-note-about-git-commit-messages.html>

Tipps und Tricks III

- Tagging
 - Verpasst einem bestimmten Versionsstand eine Markierung
 - Wird oft verwendet, um Releases zu markieren (Bsp: v1.1)
 - Ermöglicht das schnelle Arbeiten mit unterschiedlichen Versionen des Repositories

- Alle existierenden Tags anzeigen

```
$ git tag
```

- Einen einfachen Tag erstellen

```
$ git tag v1.1
```

- Einen ausführlichen Tag erstellen (mit Name, commit message, ...)

```
$ git tag -a v1.1
```

- Einen ausführlichen, mit GPG signierten, Tag erstellen

```
$ git tag -s v1.1
```


Versionsverwaltung

Einstieg in git

Branching und Merging

Collaboration

Branching und Merging

A vertical line on the left side of the table represents the Git commit history. It features colored circles (orange, blue, yellow, grey, purple) corresponding to the commit messages. Branches are indicated by horizontal lines extending from the main vertical line: a yellow branch for 'change colors; add caption' and 'use color in git status listing', and a purple branch for 'Add branch-merge section' and 'Escape diff highlighting'.

Merge branch 'master' of gitorious.org:valug/git-slides	Wolfgang Silbermayr	Sam 19 Mär 2011 15:32:41 CET
Add branching commands	Wolfgang Silbermayr	Sam 19 Mär 2011 15:32:31 CET
change colors; add caption	Florian Preinstorfer	Sam 19 Mär 2011 15:32:12 CET
use color in git status listing	Florian Preinstorfer	Sam 19 Mär 2011 15:21:32 CET
Merge branch 'master' of gitorious.org:valug/git-slides	Florian Preinstorfer	Sam 19 Mär 2011 15:14:12 CET
add status file	Florian Preinstorfer	Sam 19 Mär 2011 15:14:06 CET
Some special cases for diff listing highlighting	Wolfgang Silbermayr	Sam 19 Mär 2011 15:12:18 CET
Add branch-merge section	Wolfgang Silbermayr	Sam 19 Mär 2011 15:12:01 CET
Escape diff highlighting	Wolfgang Silbermayr	Sam 19 Mär 2011 14:54:49 CET
add diff	Florian Preinstorfer	Sam 19 Mär 2011 14:53:18 CET

Branching und Merging visuell dargestellt (gitg)

Branching I

- Erzeugen von Verzweigungen
- Unter git sehr leichtgewichtig
- Jeder Branch hat einen Namen
- Standardmäßig wird ein `master`-Branch verwendet

Branching II

- Anzeigen der lokalen Branches:

```
$ git branch
```

- Anlegen eines neuen Branches:

```
$ git branch <branchname>
```

- Wechseln auf einen anderen Branch:

```
$ git checkout <branchname>
```

- Neuen Branch anlegen und auschecken in einem Schritt:

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

Merging

- Zusammenführen mehrerer Branches
- Umfangreiche Unterstützung bei Konflikten
- Merging:

```
$ git checkout <branch>           # checkout <branch>
$ git merge <otherbranch>         # merge <otherbranch> into <branch>
$ git branch -d <otherbranch>    # optionally delete <otherbranch>
```

Versionsverwaltung

Einstieg in git

Branching und Merging

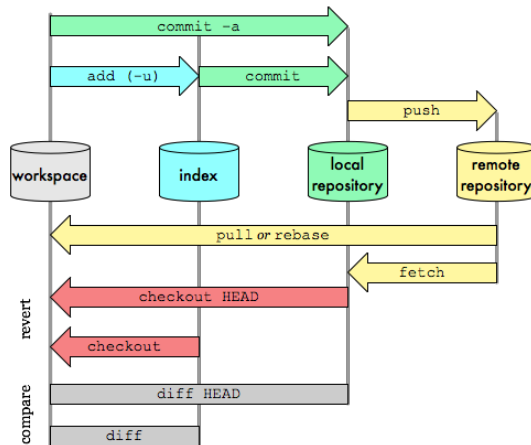
Collaboration

Collaboration

- Jedes git Repository enthält die vollständige Versionshistorie
- Jeder Benutzer kann Repositories veröffentlichen (push)
- Unterstützte Protokolle:
 - File
 - Git
 - Http(s)
 - Ssh
- Git unterstützt mehrere Entwicklungsmodelle (Distributed Workflows)

Git Data Transport Commands

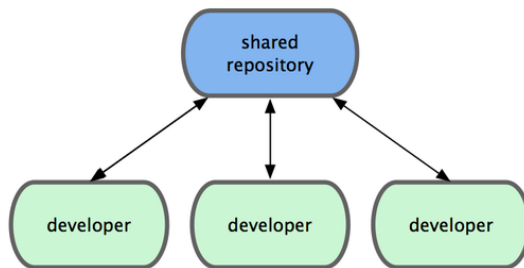
<http://osteele.com>



Quelle: <http://osteele.com/archives/2008/05/my-git-workflow>

Distributed Workflows

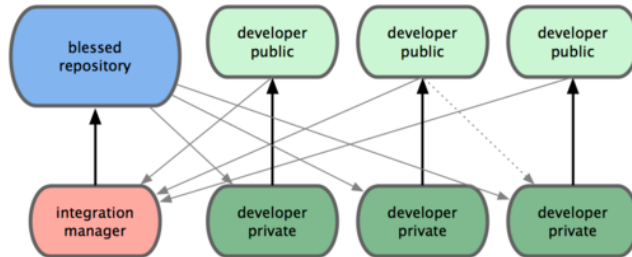
Centralized Workflow



Quelle: <http://progit.org>

Distributed Workflows

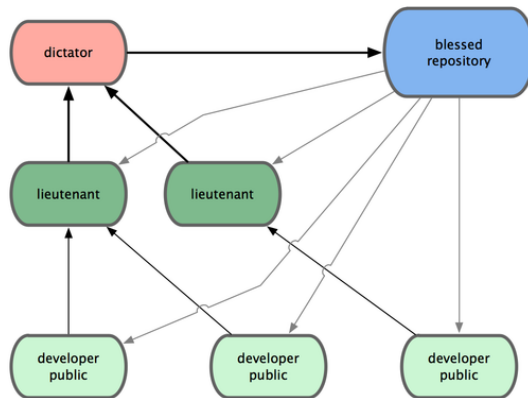
Integration-Manager Workflow



Quelle: <http://progit.org>

Distributed Workflows

Dictator and Lieutenants Workflow



Quelle: <http://progit.org>

Git Hosting I

- Github:
 - Hosting unter: <http://github.com>
 - Hohe Popularität
 - Viele Features
 - Quellcode nicht verfügbar
- Gitorious:
 - Hosting unter: <http://gitorious.org>
 - Integrierte Benutzerverwaltung
 - Freie Software (AGPL, <http://gitorious.org/gitorious>)

Git Hosting II

- Gitis:
 - Einfach, gut geeignet für kleineres Setup
 - Freie Software (GPLv2, <http://eagain.net/gitweb/?p=gitis.git>)
- Gitolite:
 - Komplex, aber sehr flexibel
 - Freie Software (GPLv2, <https://github.com/sitaramc/gitolite>)

Git Repositories klonen

- Git kopiert beim Klonen immer die vollständige Versionshistorie
- Die Adresse des geklonten Repositories wird automatisch mit dem Namen „origin“ angelegt
- Die Quellen dieses Workshops klonen:

```
$ git clone git://gitorious.org/valug/git-slides.git      #klonen via git
$ git clone git@gitorious.org:valug/git-slides.git      #klonen via ssh
$ git clone http://gitorious.org/valug/git-slides       #klonen via http
```

Remotes I

- Git merkt sich mittels Remotes,
 - von wo Änderungen abgeholt werden können
 - wo Änderungen publiziert werden können
- Bei `git clone` wird das Repository automatisch als **origin** konfiguriert
- Remotes auflisten:

```
$ git remote      # Die Namen der Remotes auflisten  
$ git remote -v  # Die Namen und URLs der Remotes auflisten
```

Remotes II

- Remotes hinzufügen:

```
$ git remote add <remotename> <url>
```

- Remotes entfernen:

```
$ git remote rm <remotename>
```

Die Änderungen vom Remote Repository abholen

- Den aktuellen Versionsstand vom Remote Repository abholen, aber noch nicht in die eigene Arbeitskopie einpflegen:

```
$ git fetch  
$ git fetch <remotename>
```

- Die Unterschiede zwischen dem eigenen und dem entfernten „master“-branch ansehen:

```
$ git diff master origin/master
```

Die Änderungen vom Remote Repository einpflegen

- Den aktuellen Versionsstand vom Remote Repository abholen und in die eigene Arbeitskopie einpflegen:

```
$ git pull  
$ git pull <remotename> <branchname>
```


Die eigenen Änderungen veröffentlichen

- Git kann die eigenen Änderungen wieder veröffentlichen

```
$ git push
```

```
$ git push <remotename> <branchname>
```

Referenzen

- <http://git-scm.com>
- <http://gitready.com>
- <http://progit.org>
- <http://whygitisbetterthanx.com>

Danke!



This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 Austria license (CC-BY-SA).