Git – eine *kurze* Einführung

Malte Schmitz ~ Mai 2012

→ 1

Ziele dieses Vortrags

1. Git installieren und einrichten können.
2. Idee von verteilter Versionskontrolle verstehen.
3. Idee der nichtlinearen Entwicklung verstehen.
4. Mit Git arbeiten können.

Inhaltsverzeichnis

→ 2
→ 3

1 Installation

1.1 Windows

- `http://code.google.com/p/gitextensions`
- Git Extensions Windows installer, complete including MSysGit and KDiff3

→ 4

1.2 Linux

Verwende den Paket-Manager deiner Wahl

```
sudo apt-get install git-core
```

Als graphische Oberfläche kann nativ `gitg` verwendet werden

```
sudo apt-get install gitg
```

Alternativ funktioniert auch Git Extensions über Mono

- zip-Archiv herunterladen und entpackane
- `mono GitExtensions.exe`

→ 5

1.3 don't use

- SmartGit – zu viel SVN im Wording
- TortoiseGit – macht Windows Explorer langsam
- Netbeans-Plugin – macht gerne alles kaputt
  - Legt *repository* so an, dass neue *branches* nicht gefunden werden.

→ 6

1.4 Einrichten

SSH-Key

- SSH-Private-Key muss unter `~/.ssh/id_rsa` liegen.
- Windows: `C:\Users\schmitz\.ssh\id_rsa`
- Linux: `/home/schmitz/.ssh/id_rsa`

→ 7

mehrere SSH-Keys

- SSH-Private-Key 1 `~/.ssh/schmitz`
- SSH-Private-Key 2 `~/.ssh/schmitz-isp`
- (neue) Konfiguration `~/.ssh/config`

```
Host *.isp.uni-luebeck.de
  IdentityFile ~/.ssh/schmitz-isp
  User schmitz

Host schul-logistik.de
  IdentityFile ~/.ssh/schmitz
  User malte@schmitz-sh.de
```

→ 8

Repository clonen

```
git clone gitolite@sshgate.isp.uni-luebeck.de:mfs_passat
```

→ 9

User-Name und eMail-Adresse setzen

```
git config user.name "Malte Schmitz"
git config user.email "m@schmitz-sh.de"
```

- Die Kommandos enthalten kein = und kein `--add`!
- Einstellungen können global oder pro Repository erfolgen.
- E-Mail-Adresse muss die E-Mail-Adresse sein, unter der Euch das ISP-Redmine kennt.

→ 10

Pull ohne Merge

```
git config pull.rebase true
```

→ 11

Wir brauchen einen Editor

```
git config core.editor vim
```

```
git config core.editor
"'C:/Programme/Notepad++/notepad++.exe'
-multiInst -notabbar -nosession -noPlugin"
```

→ 12

2 Die Idee von Git

2.1 Grundidee

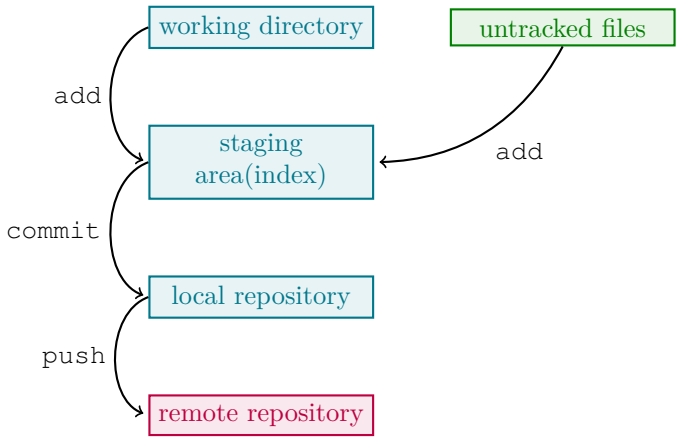
Unterschied / Fortschritte gegenüber SVN

- Git unterstützt *branches* und *tags* nativ.
  - *Branches* und *tags* sind keine Ordner.
- Repository besteht aus DAG von *commits*.
- *Branches* und *tags* sind Markierungen von *commits*.

- Git ist dezentral.
- Jeder hat das ganze *repository*.
- *Commit* ist lokal.
- Wir brauchen `push` und `pull`.

→ 13

untracked, unstaged, staged, lokal, entfernt



→ 14

2.2 Workflow

`git pull --ff-only`

- *Pull* lädt Änderungen vom *remote repository* direkt in den *workspace*.
- Entspricht *ungefähr* dem `svn up`.

Definition 2.1 (Fast-Forward).
- Pullen von Änderungen, die direkt auf den letzten lokalen *commit* folgen.
- Es gab nie simultane *commits* und kein *merge* wird benötigt.

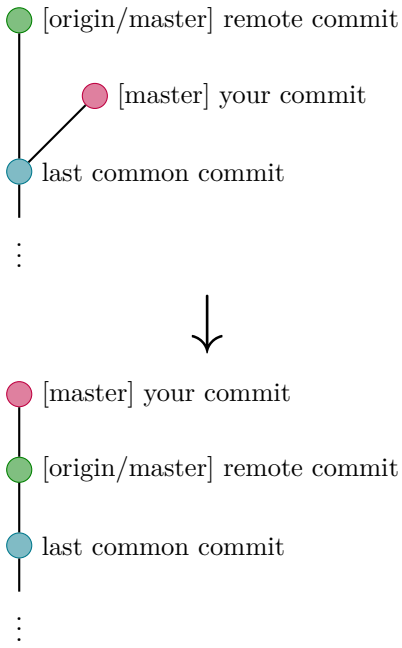
→ 15

`git pull --rebase`

- *Fast-forward* ist nicht immer möglich. `fatal: Not possible to fast-forward, aborting.`
- *Rebase* ist meist besser als *Merge*.

Definition 2.2 (Rebase).
- Wird nötig, wenn lokale *commits* und entfernte *commits* gleichzeitig.
- Verhält sich wie `svn up`.
- Hängt alle lokalen *commits* an das Ende des DAGs.

→ 16



→ 17

`git commit`

- `git commit` committed nur die *staged files*.
- `git add` fügt Dateien dem *index* hinzu.
- `git commit -a` committed alle *modified files*. (Unabhängig davon, ob die Dateien bereits dem *index* hinzugefügt wurden, oder nicht.)
- Nur `git add` fügt *untracked files* dem *index* hinzu.
- *Commit* mit leerer *message* wird nicht durchgeführt.

gute commit messages

- Erste Zeile maximal 50 Zeichen.

- Dann leere Zeile.
- Dann umfangreiche Beschreibung.
- Auf englisch und aus Perspektive des *commits*.

→ 18

`git push`

- Neue *commits* vom *local repository* in das *remote repository* schieben.
- Funktioniert nur, wenn *local repository up to date*. `To prevent you from losing history, non-fast-forward updates were rejected. Merge the remote changes (e.g. 'git pull') before pushing again.`
- Es werden immer *alle branches* gepusht, wenn nicht anders verlangt. `git push origin HEAD` pusht nur den aktuellen *branch*. (Je nach Voreinstellung anders!)

→ 19

2.3 Branches verwenden

`git status`

- In welchem *branch* bin ich gerade?
- Sollte ich mal pushen / pullen?

```
$>git status
# On branch 112-foobar
# Your branch is ahead of 'origin/112-foobar' by 1 commit.
#
nothing to commit (working directory clean)
```

→ 20

`git branch -a`

- Welche *branches* existieren?
- *Niemals* direkt einen *remote branch* auschecken!

```
* 112-foobar
  3.1.x
  develop
  foo-feature
  ultra-feature
  ultra-feature-3.1.x
  remotes/origin/112-foobar
  remotes/origin/3.1.x
  remotes/origin/develop
```

→ 21

`git checkout -b 103-foobar`

```
git checkout master
```

Wechselt in den *master branch*.

```
git checkout 103-foobar
```

Wechselt in den *feature branch*.

```
git checkout -b 103-foobar
```

Legt einen neuen *feature branch* an.

Name eines fature branches

- Ticket-Nummer
- Bindestrich (-)
- Feature-Kurz-Name

→ 22

`git push origin HEAD`

- Durch `git push` wird ein neuer *branch* nicht automatisch mit gepusht.
- Verwende `git push origin HEAD` um (nur) den aktuellen (neuen) *branch* explizit zu pushen.

→ 23

`git merge 103-foobar`

1. In den *master branch* wechseln. `git checkout master`
2. Mergen. `git merge 103-foobar`
3. Konflikte auflösen. `git mergetool git commit`
4. Neuen *commit* ansehen. `git log`
5. Neuen *commit* pushen. `git push`

Hinweis

Konflikte können auch beim *rebase* auftreten.

→ 24

```
$>git checkout master
Switched to branch 'master'

$>git merge 112-foobar
Auto-merging foobar.txt
CONFLICT (content): Merge conflict in foobar.txt
Automatic merge failed; fix conflicts and then commit the result.

$>git mergetool
Merging:
foobar.txt

Normal merge conflict for 'foobar.txt':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (kdiff3):

$>git commit
[master 345abe2] Merge branch '112-foobar' into master
```

```
$>git log
commit 345abe253b838e635408c88082954ec14fa2a386
Merge: 79f920f b3ecc05
Author: malteschmitz <malte@schmitz-sh.de>
Date:   Fri Jun 1 02:21:18 2012 +0200

    Merge branch '112-foobar' into 3.1.x

    Conflicts:
        foobar.txt
```

→ 25

Nimm das!

```
git checkout --ours index.html
```

Verwende unsere `index.html` und verwirf *remote changes*.

```
git checkout --theirs default.html
```

Verwende die *remote* `default.html` und verwirf *local changes*.

→ 26

2.4 Mergen ohne Branch

`git chery-pick a6c56d78`

Given one or more existing commits, apply the change each one introduces, recording a new commit for each. This requires your working tree to be clean (no modifications from the HEAD commit).

→ 27

2.5 Rückgängig machen

`git commit --amend`

Used to amend the tip of the current branch. Prepare the tree object you would want to replace the latest commit as usual (this includes the usual -i/-o and explicit paths), and the commit log editor is seeded with the commit message from the tip of the current branch. The commit you create replaces the current tip.

→ 28

`git reset HEAD foobar.java`

```
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   foobar.txt
```

Without further arguments `git reset` copies entries from `HEAD` to the index. Can be used to set the current branch head, too.

→ 29

`git checkout -- foobar.java`

```
git checkout -- foobar.java
```

Restores `foobar.java` from the index. → 30

`git revert a6c56d78`

- `revert` macht einen existierenden *commit* rückgängig.
- Der Commit verschwindet nicht, sondern ein neuer *commit* negiert Auswirkungen des ursprünglichen *commits*.

don't break the history

Mit `git reset` kann man viel zaubern. Steht zu euren Fehlern. Verwendet `git revert`. → 31

Zusammenfassung

1. Git ist eine dezentrale Versionsverwaltung (`git pull`, `git push`).
2. Git-Repositorys sind DAGs von *commits*. *Tags* und *commits* sind keine Ordner mehr.
3. Jedes Feature bekommt einen eigenen *branch*. *Merges* tun mit Git nicht mehr weh.
4. Wir setzen alle Benutzername und E-Mail-Adresse und aktivieren automatisches *rebase* (`git config`)! → 32