

Änderungen verwalten mit git

Wie arbeitet man am besten an einem Protokoll zusammen?

Idee: Austausch über Mails

Mails: Probleme

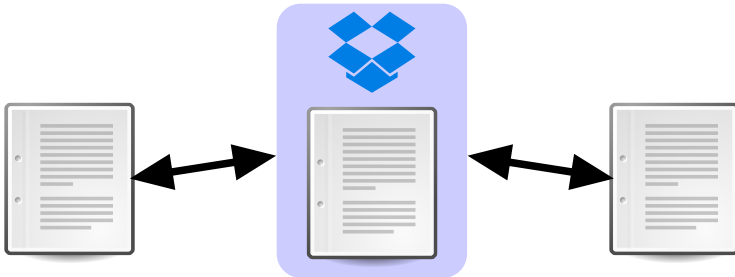


- Risiko, dass Änderungen vergessen werden, ist groß
- Bei jedem Abgleich muss jemand anders aktiv werden
 - Stört
 - Es kommt zu Verzögerungen

Fazit: Eine sehr unbequeme / riskante Lösung

Idee: Austausch über Dropbox

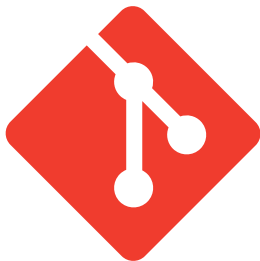
Dropbox: Probleme



- Man merkt nichts von Änderungen der Anderen
- Gleichzeitige Änderungen führen zu „In Konflikt stehende Kopie“-Dateien.
- Änderungen werden nicht zusammengeführt.

Fazit: Besser, aber hat deutliche Probleme

Lösung: Änderungen verwalten mit git



git

- Ein Versionskontrollsystem
- Ursprünglich entwickelt, um den Programmcode des Linux-Kernels zu verwalten (Linus Torvalds)
- Hat sich gegenüber ähnlichen Programmen (SVN, mercurial) durchgesetzt

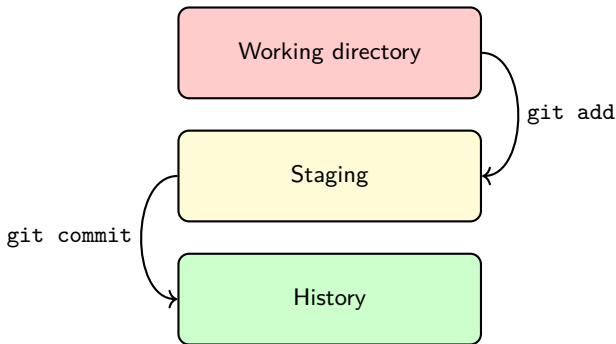
Was bringt git für Vorteile?

- Arbeit wird für andere sichtbar protokolliert
- Erlaubt Zurückspringen an einen früheren Zeitpunkt
- Kann die meisten Änderungen automatisch zusammenfügen
- Wirkt nebenbei auch als Backup

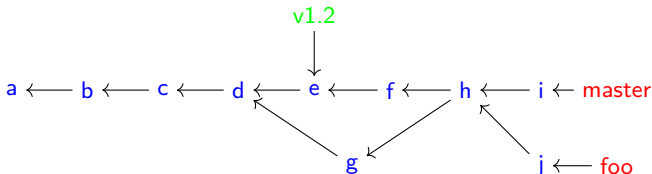
Einziges Problem: Man muss lernen, damit umzugehen

Zentrales Konzept: Das Repository

- Erzeugen mit `git init`



History



- **Commit**: Zustand/Inhalt des Arbeitsverzeichnisses zu einem Zeitpunkt
 - Snapshot, Name ist Hash des Inhalts, enthält Commit-Message (Beschreibung der Änderungen)
- **Branch**: benannter Zeiger auf einen Commit
 - Entwicklungszweig, im Praktikum nur master
- **Tag**: unveränderbarer Zeiger auf einen Commit
 - Wichtiges Ereignis, z.B. veröffentlichte Version

Workflow

- 1 Repo erzeugen/klonen: `git init`, `git clone`
- 2 Arbeiten
 - 1 Dateien bearbeiten, testen: `vim`, `make`
 - 2 Änderungen in Staging schieben: `git add`
 - 3 Commit erzeugen: `git commit`
- 3 Commits anderer herunterladen und integrieren: `git pull`
- 4 Eigene Commits hochladen: `git push`

git init, git clone

<code>git init</code>	initialisiert ein git-Repo im jetzigen Verzeichnis
<code>git clone <i>url</i></code>	klont das Repo aus <i>url</i>
<code>rm -rf .git</code>	löscht alle Spuren von git aus dem Repo

git status, git log

- | | |
|-------------------------|--|
| <code>git status</code> | zeigt Status des Repos (welche Dateien sind neu, gelöscht, verschoben, bearbeitet) |
| <code>git log</code> | listet Commits in aktuellem Branch |

git add, git mv, git rm, git reset

- git add *file* ... fügt Dateien/Verzeichnisse zum Staging-Bereich hinzu
- git add -p ... fügt Teile einer Datei zum Staging-Bereich hinzu
- git mv wie mv (automatisch in Staging)
- git rm wie rm (automatisch in Staging)
- git reset *file* entfernt Dateien/Verzeichnisse aus Staging

git diff

<code>git diff</code>	zeigt Unterschiede zwischen Staging und Arbeitsverzeichnis
<code>git diff --staged</code>	zeigt Unterschiede zwischen letzten Commit und Staging
<code>git diff <i>commit1</i> <i>commit2</i></code>	zeigt Unterschiede zwischen zwei Commits

git commit

<code>git commit</code>	erzeugt Commit aus jetzigem Staging-Bereich, öffnet Editor für Commit-Message
<code>git commit -m "message"</code>	Commit mit <i>message</i> als Message
<code>git commit --amend</code>	letzten Commit ändern (fügt aktuellen Staging hinzu, Message bearbeitbar)

- Sinnvolle Commit-Messages schreiben!
 - Erster Satz ist Zusammenfassung
- Logische Commits erstellen, für jede logische Einheit ein Commit
 - `git add -p` sehr nützlich
- Hochgeladene Commits nicht mehr ändern!

git pull, git push

- | | |
|--------------------------------|---|
| <code>git pull</code> | Commits herunterladen (mit Merge-Commit) |
| <code>git pull --rebase</code> | Commits herunterladen (ohne Merge-Commit) |
| <code>git push</code> | Commits hochladen |
- (falls gewollt) `--rebase` standardmäßig:
`git config --global pull.rebase true`

Achtung: Merge conflicts

Don't Panic

Entstehen, wenn git nicht automatisch mergen kann (selbe Zeile geändert, etc.)

- 1 Die betroffenen Dateien öffnen
- 2 Markierungen finden und die Stelle selbst mergen (meist wenige Zeilen)

```
<<<<<<< HEAD
foo
|||||| merged common ancestors
bar
=====
baz
>>>>>>> Commit-Message
```

- 3 Merge abschließen: `git add ...`
 - kein `--rebase`: `git commit` ausführen, um zu bestätigen
 - `--rebase`: `git rebase --continue`

Nützlich: `git config --global merge.conflictstyle diff3`

git checkout

- | | |
|---|---|
| <code>git checkout <i>commit</i></code> | Commit ins Arbeitsverzeichnis laden |
| <code>git checkout <i>file</i></code> | Änderungen an Dateien verwerfen (zum letzten Commit zurückkehren) |

git stash

<code>git stash</code>	Änderungen kurz zur Seite schieben
<code>git stash pop</code>	Änderungen zurückholen aus Stash

.gitignore

- Man möchte nicht alle Dateien von git beobachten lassen
- z.B. build-Ordner

Lösung: .gitignore-Datei

- einfache Textdatei
- enthält Regeln für Dateien, die nicht beobachtet werden sollen

Beispiel:

```
build/  
*.pdf  
__pycache__
```

SSH-Keys

Git kann auf zwei Arten mit einem Server kommunizieren:

- HTTPS: funktioniert immer, keine Einstellungen erforderlich, Passwort muss für jede Kommunikation eingegeben werden
- SSH: Keys müssen erzeugt und eingestellt werden, keine Passwörter mehr erforderlich

SSH-Keys:

- 1 `ssh-keygen`
- 2 Standardeinstellungen ok (kein Passwort!)
- 3 `cat ~/.ssh/id_rsa.pub`
- 4 Ausgabe ist Public-Key, beim Server eintragen (im Browser)

Hoster

GitHub

- größter Hoster
- viele open-source Projekte
- keine (kostenlosen) privaten Repos

Atlassian
 **Bitbucket**

- kostenlose private Repos
- keine Speicherbegrenzungen