

Änderungen verwalten mit git

PeP et al. Toolbox Workshop



PEP ET AL. E.V.
PHYSIKSTUDIERENDE UND
EHMALIGE PHYSIKSTUDIERENDE
DER TU DORTMUND

2015

Wie arbeitet man am besten an einem Protokoll
zusammen?

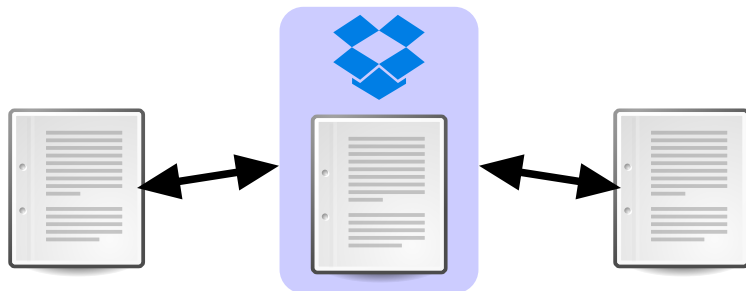
Idee: Austausch über Mails



- 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

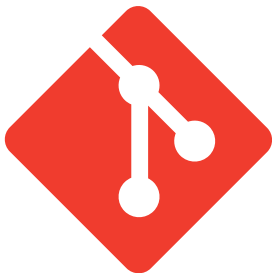
Idee: Austausch über Dropbox



- 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
- Wird in der Regel über die Kommandozeile benutzt

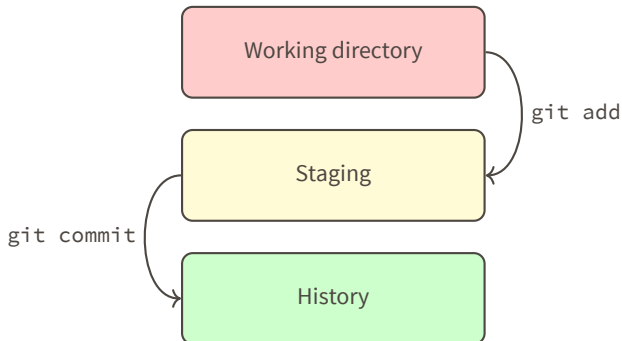
Was bringt git für Vorteile?

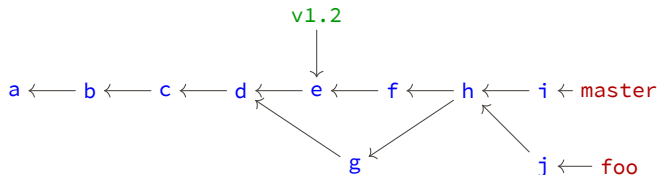
- Arbeit wird für andere sichtbar protokolliert
- Erlaubt Zurückspringen an einen früheren Zeitpunkt
- Kann die meisten Änderungen automatisch zusammenfügen
- Wirkt nebenbei auch als Backup

Einzige Herausforderung: Man muss lernen, damit umzugehen

Zentrales Konzept: Das Repository

- Erzeugen mit `git init`
- Damit wird der jetzige Ordner zu einem Repository





- **Commit**: Zustand/Inhalt des Arbeitsverzeichnisses zu einem Zeitpunkt
 - Enthält Commit-Message (Beschreibung der Änderungen)
 - Wird über einen Hash-Code identifiziert
- **Branch**: benannter Zeiger auf einen Commit
 - Entwicklungszweig
 - Im Praktikum reicht bereits die Standard-Branch: master
- **Tag**: unveränderbarer Zeiger auf einen Commit
 - Wichtiges Ereignis, z.B. veröffentlichte Version

1. Repository erzeugen oder klonen: `git init`, `git clone`
2. Arbeiten
 - 2.1 Dateien bearbeiten und testen
 - 2.2 Änderungen vorbereiten: `git add`
 - 2.3 Änderungen als *commit* speichern: `git commit`
3. Commits anderer herunterladen und integrieren: `git pull`
4. Eigene Commits hochladen: `git push`

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

- `git status` zeigt Status des Repos (welche Dateien sind neu, gelöscht, verschoben, bearbeitet)
- `git log` listet Commits in aktuellem Branch

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

<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

<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)

- Wichtig: Sinnvolle Commit-Messages
 - Erster Satz ist Zusammenfassung
- Logische Commits erstellen, für jede logische Einheit ein Commit
 - `git add -p` ist hier nützlich
- Hochgeladene Commits sollte man nicht mehr ändern

`git pull` Commits herunterladen

`git push` Commits hochladen

→ Aus der Installationsanleitung:

```
git config --global pull.rebase true
```

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:

- 3.1 `git add ...`
- 3.2 `git rebase --continue`

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

<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)

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

- 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__/
```

Git kann auf mehrere 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)

GitHub

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



- kostenlose private Repos mit höchstens fünf Leuten
- keine Speicherbegrenzungen



- open-source
- keine Begrenzungen an privaten Repos
- kann man selbst auf einem eigenen Server betreiben