Softwareentwicklung I (IB)
— Versionsverwaltung mit Git —

Prof. Dr. Oliver Braun

Letzte Änderung: 12.10.2017 06:55

Versionen

- egal was Sie bearbeiten, es wird meist mehrere Versionen geben
- Beispiel: Brief an Opa

1. Initialer/leerer Brief
2. Adressen eingefügt
3. Anrede “lieber Opa”
4. Anrede geändert in “Hey Alter”
5. zurück zu Version 3
6. Absatz über mein Studium
7. Absatz über meine Eltern und Geschwister
8. Absatz über Eltern und Geschwister gelöscht
9. Absatz fehlendes Geld mit Bitte mir was zu schicken
10. Absatz aus Version 8 doch wieder eingefügt

https://github.com/ob-seiib-ws17/ob-seiib-ws17

Versionskontrollsysteme (VCS)

- nicht adäquat
  - Dateien zum Verwalten von Versionen verschieden benennen, z.B.

    ```
    brief_2009_07_15.txt
    brief_2009_07_16.txt
    brief.txt_old
    brief_soAbgeschickt.txt
    ```

- alles das machen Sie **besser** mit einem VCS und ihr Arbeitsverzeichnis ist viel aufgeräumter!
  - ein VCS ist ein Programm, mit dem Sie Versionen verwalten können
  - die Versionen werden, für Sie “unsichtbar”, vom VCS aufbewahrt

Git

- modernes, sehr weit verbreitetes **D**istributed VCS
- Arbeitsverzeichnis heisst **Working Copy**
- von Git verwaltete Daten in in einem sog. **Repository**
- Beispiel: livecoding-Repository

Working Copy → Staging Area → Repository

- wir bearbeiten die Datei `a.txt`
  - Änderungen nur im Working Copy vorhanden
- wir **stagen** die Datei `a.txt`

  ```
  git add a.txt
  ```

  - Änderungen zum nächsten Commit an Git "gemeldet"
- wir **committen** die Datei `a.txt`

  ```
  git commit -m 'Änderungen ...'
  ```

  - Änderungen sind jetzt als eigener Commit (Version) im Repository

Wie bekommen Sie meine Dateien und meine Änderungen?

- bisher alles lokal auf meinem Rechner
- ich könnte Ihnen jetzt mein Verzeichnis auf einen USB-Stick kopieren, in die Dropbox stellen, ...
  - SCHMARRN!!!
- das kann auch Git für mich verwalten: **Distributed** VCS

Bare Repository

- zur Zusammenarbeit bietet es sich an ein zentrales Repository zu nutzen
  - zentralisierte VCS (z.B. Subversion) nutzen nur ein zentrales Repo
- bei Git bilden Working Copy und Repository normalerweise eine Einheit
- Repository ohne Working Copy heisst **bare Repository**
- kann z.B. auf beliebigem Server liegen, auf den alle Beteiligten Zugriff haben
- es gibt auch spezielle Git-Hoster, z.B. GitHub oder Bitbucket
  - wir nutzen GitHub

Von Repository zu Repository

- **in** ein Repository können Änderungen *geschoben* werden

  `git push`

- **von** einem Repository können Änderungen *gezogen* werden

  `git pull`

- aber woher weiß Git in welches bzw. von welchem Repository?

Der Ursprung

- üblicherweise lege ich erst ein bare Repository auf dem Server an
  - z.B. auf GitHub
- dann *klone* ich mir das Repository auf meinen Rechner

  `git clone <repo-url>`
- damit erzeugt Git
  - ein identisches Repository auf meinem Rechner
  - ein Working Copy in dem der aktuelle Stand ist
  - ein *Bookmark* mit dem Namen `origin`
- wenn ich in Zukunft nur `git pull` oder `git push` schreibe, wird als Quelle bzw. Ziel `origin` genommen

Und jetzt zu Ihnen...

- genau wie ich, klonen Sie als Erstes das Repository
- wenn ich eine Änderung mache
  - stage ich sie,

    `git add ...`
  - committe ich sie

    `git commit ...`
  - und pushe ich sie

    `git push`
- Sie holen sich anschließend diese Änderungen mit

  `git pull`
- ...und umgekehrt analog

Git pull revisited

- wo sind eigentlich die Änderungen nach einem `git pull`?
- im lokalen Repository
  - das macht eigentlich

    `git fetch`
- im lokalen Working Copy
  - das macht eigentlich

    `git merge`
- `git pull` ist die sinnvolle Abkürzung für `git fetch` gefolgt von `git merge`

Ist das schon alles?

- Git kann noch viel viel mehr
- was ich Ihnen gezeigt habe, reicht uns aber für den Moment
- später kommt noch ein bisschen mehr