Programmierung: Java 2

Jürgen Hermes – IDH – SoSe 2025

Versionskontrolle mit git und GitHub

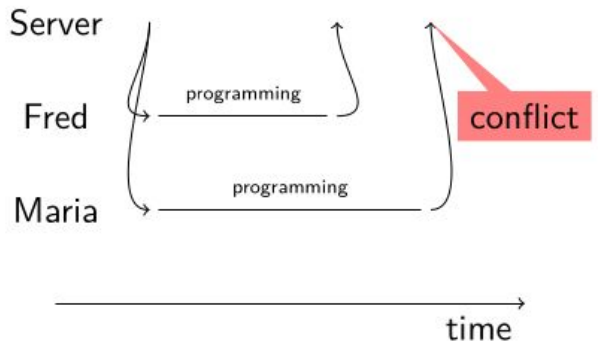
- Wofür brauchen wir Versionskontrolle?
- Um was handelt es sich bei den Begriffen git – GitHub – GitLab?
- Wie funktioniert das theoretisch?
- Wie funktioniert das praktisch bei der Abgabe der Hausaufgaben?
- Was lernen wir erst später über git?

Warum Versionskontrolle?

- Verwaltung von verschiedenen (geschichtlichen oder konkurrierenden) Versionen von SouceCode (oder auch von anderen Daten).
- Geschichtliche Änderungen lassen sich verfolgen.
- Konkurrierende Versionen lassen sich gemeinsam verwalten.
- Generell: Behandlung von Komplexität
    - Große Teams arbeiten an demselben Projekt über eine lange Zeit.
    - Windows 2000: 28 Mio Codezeilen
    - CorefAnnotator (Projekt von Nils Reiter): 27.000 Codezeilen
    - Konzeptuelle Änderungen wirken sich ggfs. auf viele unterschiedliche Stellen aus.

Wie könnte man zusammen am gleichen Code schreiben?

- Vorschläge?
- Was kann schon schiefgehen?



- Welche Optionen gibt es da?
  - Ignorieren, Marias Code überschreibt den von Fred (Ignore, keine gute Strategie)
  - Erzeuge eine zweite Kopie (Copy, Dropbox-Strategie)
  - Zwinge Maria, den Code explizit mit dem von Fred abzugleichen (Merge, git-Strategie)

Was ist git?

- Nicht die erste Versionskontrollsoftware, aber die gegenwärtig meistgenutze frei verfügbare.
- Open Source Projekt von den Leuten, die auch Linux-Kernel-Entwicklern – Download (auch Sourcecode) https://git-scm.com
- Verteiltes System, kommt (theoretisch) ohne zentralen Server aus.
- Schnell und Datensicher (checksum-geprüft)
- Stand-alone nutzbar, aber auch integriert in verschiedenen Tools (z. B. Eclipse)
- Literatur: Scott Chacon and Ben Straub: “Pro Git”, 2nd edition, Apress, 2014. https://git-scm.com/book/en/v2

GitHub und GitLab

- GitHub: Kommerzielle Webplattform (https://github.com)
  - Gegründet 2008, seit 2028 von Microsoft betrieben, kostenlose Version nutzbar
  - Stellt zentralen Server bereit für git repositories und weitere Services (Organisations- und Nutzerverwaltung, wiki, ticket system, statische Webseiten zur Dokumentation u.v.m.)

- GitLab: Open-Source-Software (https://about.gitlab.com)
  - Stellt Software bereit, mit dem man auf seinem eigenen Server einen zentralen Server für git repositories installieren kann.
  - “GitHub for yout own server”

- Bsp: autodone https://github.com/dh-cologne/autodone

Commit: Konzept

- Commit: Version eines Verzeichnisses zu einem bestimmten Zeitpunkt.
- Zeitpunkt wird von Entwicklern festgelegt (etwa, wenn ein bestimmtes Problem gelöst wurde)
- Jeder Commit kennt seinen unmittelbaren Vorgänger-Commit.
- Jeder Commit ist über einen Hash-Wert identifizierbar.
- Ein Commit kann mehrere Änderungen in mehreren Dateien umfassen.
- Registrierung der Änderungen erfolgt zweistufig:
  - Sammlung der zu commitenden Änderungen in einer Staging-Area
  - Commiten aller gesammelten Änderungen

Versionskontrolle – für was (nicht)?

- Gut geeignet für alle Arten von Plaintext-Dateien. Dazu gehören
  - Source Code – python, java, R, perl, prolog etc.
  - (Plain)Texte – txt, latex, markdown
  - Primärdaten – markup, csv
  - Vektorgrafiken – svg
- Nicht geeignet für Binärdaten, archivierte oder codierte Dateien
  - Word-, Excel- oder pdf-Dokumente
  - Bilder in jpg oder png-Format
  - Kompilierter Code wie class-Files oder exe-Dateien
  - Gepackte Dateien wie zip tar.gz

→ Ausnahmen sind (wie meistens) möglich.

Basis-Workflow (Kommandozeile oder Eclipse o.a.)

1. **`git pull`** – aktuelle Änderungen aus Server in lokales Verzeichnis speichern
2. Neue Dateien hinzufügen oder vorhandene editieren
3. **`git add / git remove`** – Dateien in Staging Area überführen / löschen
4. **`git commit`** – Dateien senden (nützliche Beschreibung hinzufügen)
5. **`git push`** – gesammelte Änderungen auf den Server schieben

→ Bsp. in Eclipse

Branching: Konzept

- Verschiedene Versionen, die parallel zueinander existieren können.
- Zu jeder Zeit ist nur ein einzelner Branch aktiv
- Default-Branch: `main` oder `master`
- Wechseln zu Branches:
  - Bereits existierend: `git checkout <BRANCHNAME>`
  - Zu neu erstelltem: `git checkout -b <BRANCHNAME>`
- Commits und Branches resultieren in einem Baum: Jeder Knoten hat genau einen Vorgängerknoten, aber kann mehrere Nachfolgeknoten haben.
- Zusammenführung von Branches mit `git merge` (später im Kurs)

Remote: Konzept

- Git Repositories können mit *remote* Repositories assoziiert werden. Diese können auf einem anderen Rechner liegen (z.B. bei GitHub)
- Lokales und remote-Repository müssen manuell miteinander synchronisiert werden.
- `git clone <REPOURL>`: Erzeugt eine lokale Kopie des remote-Repositories und setzt dieses als “origin”
- `git push`: Transferiert die commits des lokalen branches zum remote.
- `git pull`: Transferiert die commits des remote-branches zum lokalen.