Semesterprojekt
Verteilte Echtzeitrecherche in Genomdaten

Versionierung und Bugtracking mit Git(Hub)

Marc Bux (bux@informatik.hu-berlin.de)

Ziele der Versionierung

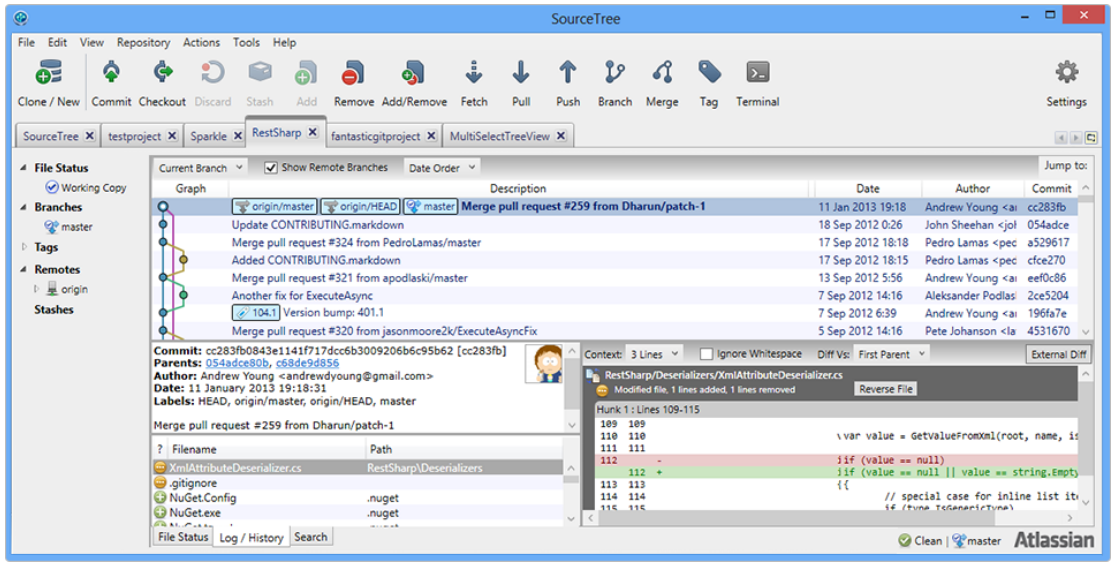
- Revisionsgeschichte eines Projekts erhalten und nachvollziehen
- Kollaboration
- Konfliktvermeidung und -handling
- „Backup“ um Fehler rückgängig zu machen
- Änderungen und Alternativen gefahrlos ausprobieren
- Information und Logs zu den Änderungen

Versionierungssysteme

- CVS (Concurrent file system)
- SVN (Subversion)
- Git
- Mercurial
- Bitkeeper
- GNU arch
- ...

Git

- 2005 von Linus Torvalds entwickelt
- GitHub: öffentlicher Host von Git-Repositories
  - https://github.com/orgs/hu-semesterprojekt-genombrowser
- Grundlegender Unterschied zu SVN: dezentralisiert
- Grafische Tools:
  - SourceTree (Win, Mac)
  - gitk (inklusive)
  - TortoiseGit (Win)
  - EGit (Eclipse)
  - Meld (diff & merge)



Begriffe

- Revision: Version (einer Datei oder Kopie des Repositories)
- Commit: Änderungen (auch neue Datei) dem Repository hinzufügen
- Diff: Unterschied zwischen zwei Revisionen
- HEAD: Aktuelle Entwicklungsversion (Revision)
- Branches: Isolierte Nebenentwicklung(en)
- Tags: Releases (Beta, release candidates, …) der Software

Befehle: Basics

- Lokale Kopie eines Repositories erstellen:

```
git clone https://github.com/hu-semesterprojekt-genombrowser/tutorial.git
```

```
git clone git@github.com:hu-semesterprojekt-genombrowser/tutorial.git
```

- Hilfe

```
git help clone
```

- Aktuellen Status abfragen
  - Listet neue Dateien, veränderte Dateien, Konflikte etc. auf
  - Vorher in das Verzeichnis des Repositories wechseln

```
git status
```

Befehle: Modifikation des Repositories

- Datei zur Stage hinzufügen (für Commit vorbereiten)

```
git add file.txt
```

- Alle neuen Dateien zur Stage hinzufügen

```
git add *
```

- Datei aus der Stage löschen

```
git rm file.txt
```

- Alle Änderungen aus der Stage (permanent) in das (lokale) Repository einfügen

```
git commit –m "commit message"
```

- Änderungen im lokalen Repository in einen Branch (master) des Remote-Repositories (origin) einpflegen

```
git push origin master
```

Befehle: Branching

- Neuen Branch erstellen und zu diesem Branch wechseln

```
git checkout –b my_branch
```

- Zum master-Branch wechseln

```
git checkout master
```

- Den zuvor erstellten Branch löschen

```
git branch –d my_branch
```

- Den neuen Branch für Kollaborateure verfügbar machen

```
git push origin my_branch
```

Befehle: Update des Repositories

- Änderungen am zentralen Repositoriy in das lokale Repository übernehmen

```
git pull origin master
```

- Einen anderen Branch in den aktuellen eigenen Branch (z.b. master) integrieren

```
git merge other_branch
```

- Manuell behobene Konflikte als behoben markieren

```
git add file.txt
```

- Eine lokale Datei durch die HEAD-Revision ersetzen

```
git checkout -- file.txt
```

- Alle lokalen Änderungen rückgängig machen

```
git fetch origin
git reset --hard origin/master
```

Befehle: Logging, Blaming, Praising

- Die History des Repositories anzeigen

```
git log
git log --author=marc
git log --pretty=oneline
```

- Anzeigen, welcher Autor für welche Zeile einer Datei verantwortlich war

```
git blame file.txt
```

Best Practices

- Zu Beginn der Arbeit und vor dem Einchecken das Repository updaten
- Branches verwenden
- Keine großen, binären Dateien in das Repository stellen
- Dateien, die im Repository nichts verloren haben, in der `.gitignore`-Datei erwähnen
- Häufig committen
  (jedes separierbare Feature, Bugfix, etc.), aber:
  - Mit aussagekräftigen Commit-Nachrichten versehen
  - Nur lauffähige Versionen
  - Optimal vorher Unit-Testfälle ausführen

Beispiel für .gitignore

```
# Compiled source #
###################
*.com
*.class
*.dll
*.exe
*.o
*.so

# Packages #
############
# it's better to unpack these files
# and commit the raw source, since git
# has its own built in compression methods
*.7z
*.dmg
*.gz
*.iso
*.jar
*.rar
*.tar
*.zip
```

```
# Logs and databases #
######################
*.log
*.sql
*.sqlite

# OS generated files #
######################
.DS_Store
.DS_Store?
._*
.Spotlight-V100
.Trashes
ehthumbs.db
Thumbs.db
```

Bug Tracking

- In GitHub in Form von „Issues“ verfügbar (inkl. „Milestones“)
- Dokumentation von Bugs und deren Lösung
- Kommunikation zwischen Anwender und Entwickler
- Diskussion zwischen Entwicklern
- Problem reproduzierbar darstellen
  - „Sometimes the program crashes...“ hilft nicht
  - Screenshots, Logdateien, Stacktrace
- Wichtigkeit einstufen und ggf. mit Tags versehen
- Problem ggf. einem Entwickler zuweisen

Wiki

- In GitHub hat jedes Repository ein Wiki
  - Bilder können per URL eingebunden werden
- Hervorragend geeignet zur Bearbeitung gemeinsamer Dokumente
  - Roadmap
  - Übersicht
  - Use Cases
  - Architektur
  - Erste Schritte
  - Verwendung der Software
  - etc.

Weiterführende Links

- 15-Minuten-Tutorial: https://try.github.io/
- Branching & Teamwork mit Git: http://nvie.com/posts/a-successful-git-branching-model/
- Informationen zu Pull-Requests: https://help.github.com/articles/using-pull-requests/