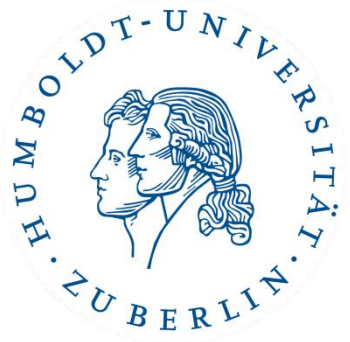


Semesterprojekt
Implementierung eines Brettspiels
(inklusive computergesteuerter Spieler)

Wintersemester 16/17

Versionierung und Bugtracking mit GitHub

Patrick Schäfer
patrick.schaefer@hu-berlin.de

Marc Bux
buxmarcn@informatik.hu-berlin.de

Ziele der Versionierung

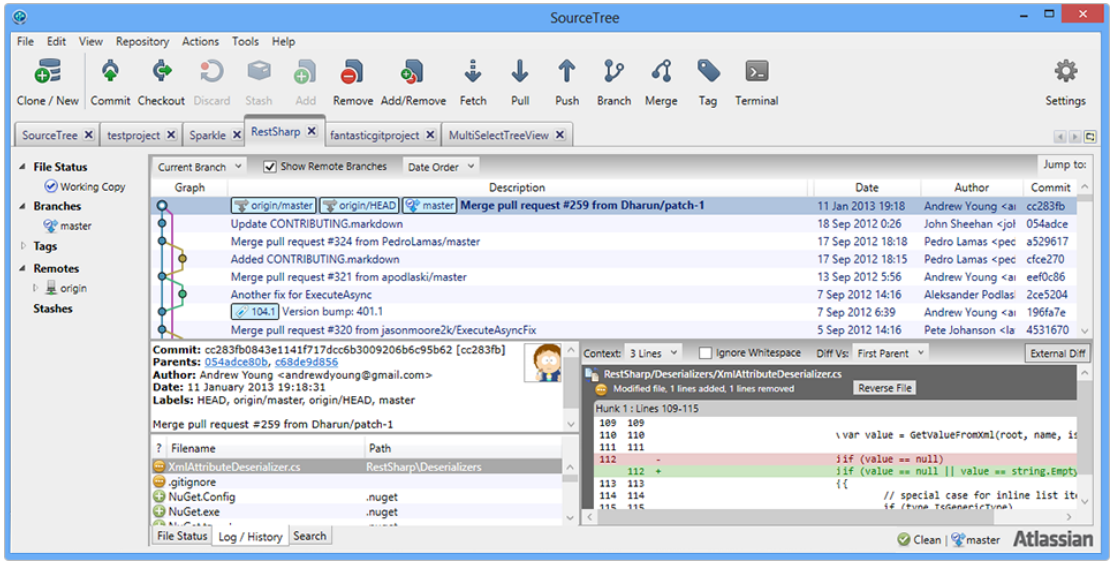
- Revisionsgeschichte eines Projekts erhalten und nachvollziehen
- Kollaboration
- Konfliktvermeidung und -handling
- „Backup“ um Fehler rückgängig zu machen
- Änderungen und Alternativen gefahrlos ausprobieren
- Information und Logs zu den Änderungen

Versionierungssysteme

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

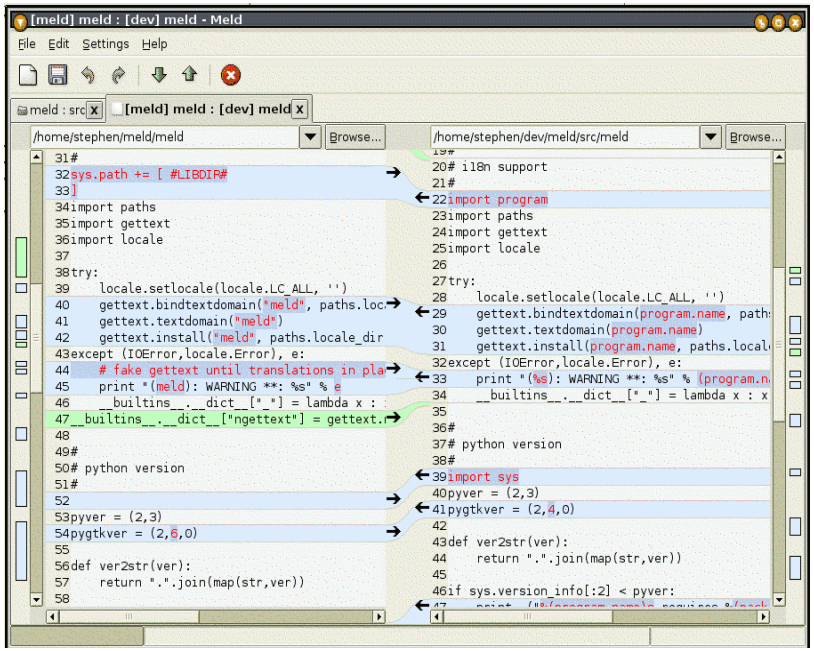
Git

- 2005 von Linus Torvalds entwickelt
- GitHub: öffentlicher Host von Git-Repositories
  - https://github.com/orgs/hu-berlin-semesterprojekte
- grundlegender Unterschied zu SVN: dezentralisiert
- grafische Tools:
  - SourceTree (Windows, Mac)



Git

- 2005 von Linus Torvalds entwickelt
- GitHub: öffentlicher Host von Git-Repositories
  - https://github.com/orgs/hu-berlin-semesterprojekte
- grundlegender Unterschied zu SVN: dezentralisiert
- grafische Tools:
  - SourceTree (Windows, Mac)
  - gitk (inklusive)
  - TortoiseGit (Windows)
  - 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)
- branch: isolierte Nebenentwicklung(en)
- master: Standard-Branch eines neuen Repositories
- tag: Releases (Beta, release candidates, …) der Software

Befehle: Basics

- lokale Kopie eines Repositories erstellen:

```
git clone https://github.com/hu-berlin-semesterprojekte/ttr-blue.git
```

```
git clone git@github.com:hu-berlin-semesterprojekte/ttr-red.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
```

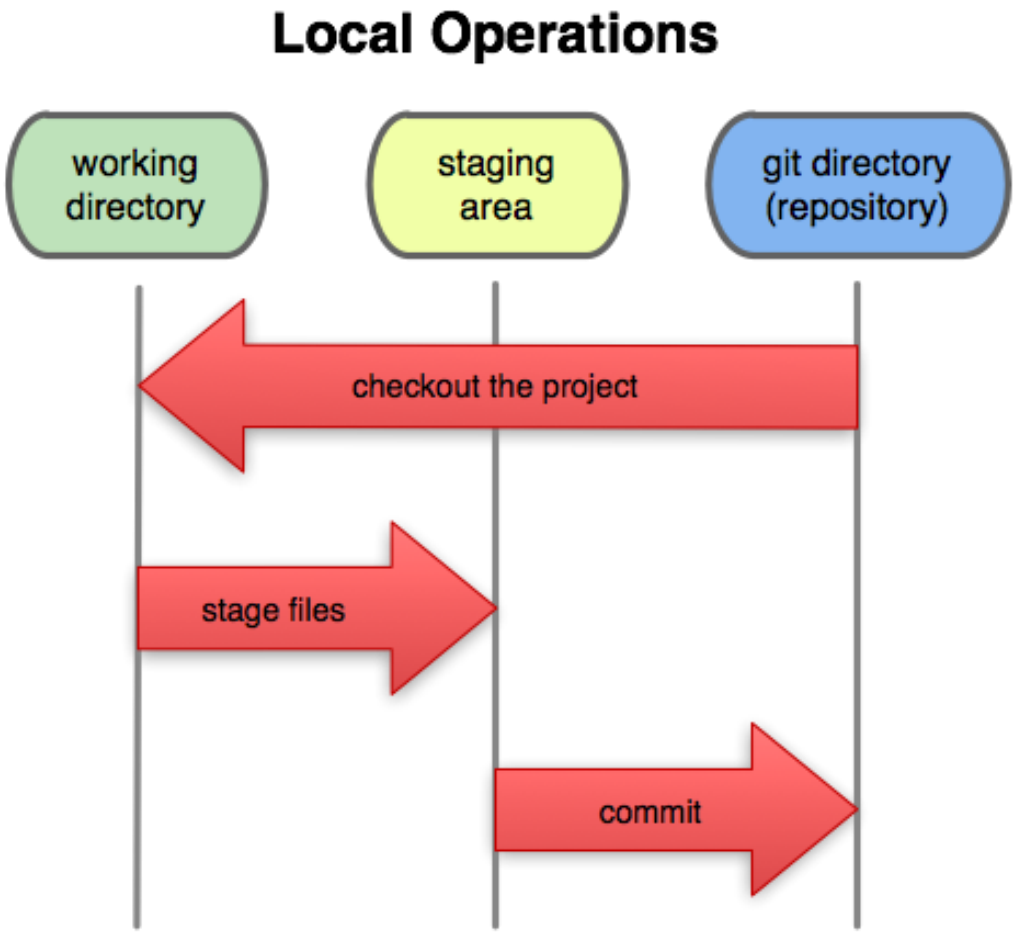
- alle Änderungen aus der Stage (permanent) in das (lokale) Repository einfügen

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

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

```
git push origin master
```

Modifikation des Repositories



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

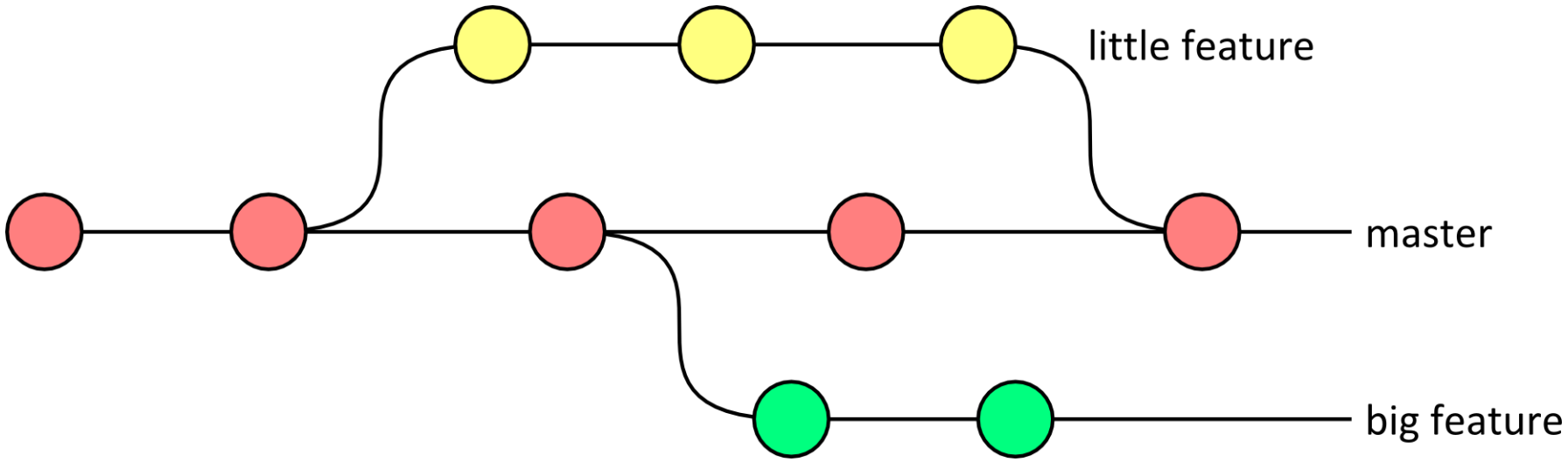
```
git push origin my_branch
```

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

```
git merge other_branch
```

- Branching & Teamwork mit Git: http://nvie.com/posts/a-successful-git-branching-model/

Branching



Befehle: Update des Repositories

- Änderungen am zentralen Repositoriy in das lokale Repository übernehmen

```
git pull origin master
```

- 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 für neue Features anlegen / verwenden
  - master-Branch sollte immer lauffähig sein
    → Vortrag Continuous Integration
- 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
  - möglichst nur lauffähige Versionen
  - vorher Tests schreiben / ausführen → Vortrag Unit Tests

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
```

.gitignore für Java-Projekte

```
*.class

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files #
*.jar
*.war
*.ear

# virtual machine crash logs
hs_err_pid*
```

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.

GitHub with Unity

- `git clone <repo-URL> <folder>`
- Unity-Projekt in Ordner <folder> verschieben
- In Unity: Edit → Project Settings → Editor
- Version Control auf „Visible Meta Files“ setzen
- Asset Serialization auf „Force Text“ setzen
- `.gitignore` (siehe rechts) hinzufügen
- `git add *`
- `git commit -am "initial commit"`
- `git push`
- Weiterführende Informationen:
  - http://www.studica.com/blog/how-to-setup-github-with-unity-step-by-step-instructions

```
/[Ll]ibrary/
/[Tt]emp/
/[Oo]bj/
/[Bb]uild/
/[Bb]uilds/
/Assets/AssetStoreTools*

# Autogenerated project files
ExportedObj/
*.csproj
*.unityproj
*.sln
*.suo
*.tmp
*.user
*.userprefs
*.pidb
*.booproj
*.svd

# Unity3D meta files
*.pidb.meta

# Unity3D crash reports
sysinfo.txt

# Builds
*.apk
*.unitypackage
```

Nächste Schritte

- 15-Minuten-Tutorial durchführen: https://try.github.io/
- Weiteres (deutschsprachiges) Tutorial ansehen: https://rogerdudler.github.io/git-guide/index.de.html
- Tasks aus dem Sprint-Backlog in Trello selbständig auswählen, selbst zuweisen und bearbeiten
- Dienstag/Mittwoch/Donnerstag & nächster Montag ab 13 Uhr:
  - „Daily“ Scrum
  - weitere optionale Treffen
- nächste Woche Montag ab 15 Uhr:
  - Backlog Grooming / Vorstellen neuer User Stories
  - Vorträge zu Unit Tests und Continuous Integration
  - evtl. Vortrag zu Coding Guidelines
- Fragen? Probleme? → E-Mail an Scrum-Master oder an uns