Versionierung und Bugtracking mit GitHub

(basierend auf Folien von Marc Bux)

Patrick Schäfer

patrick.schaefer@hu-berlin.de

Agenda

- Sprint Planning
- Vorträge
  - Git
  - (Server-Protokoll, bei Bedarf)

Ziele der Versionierung

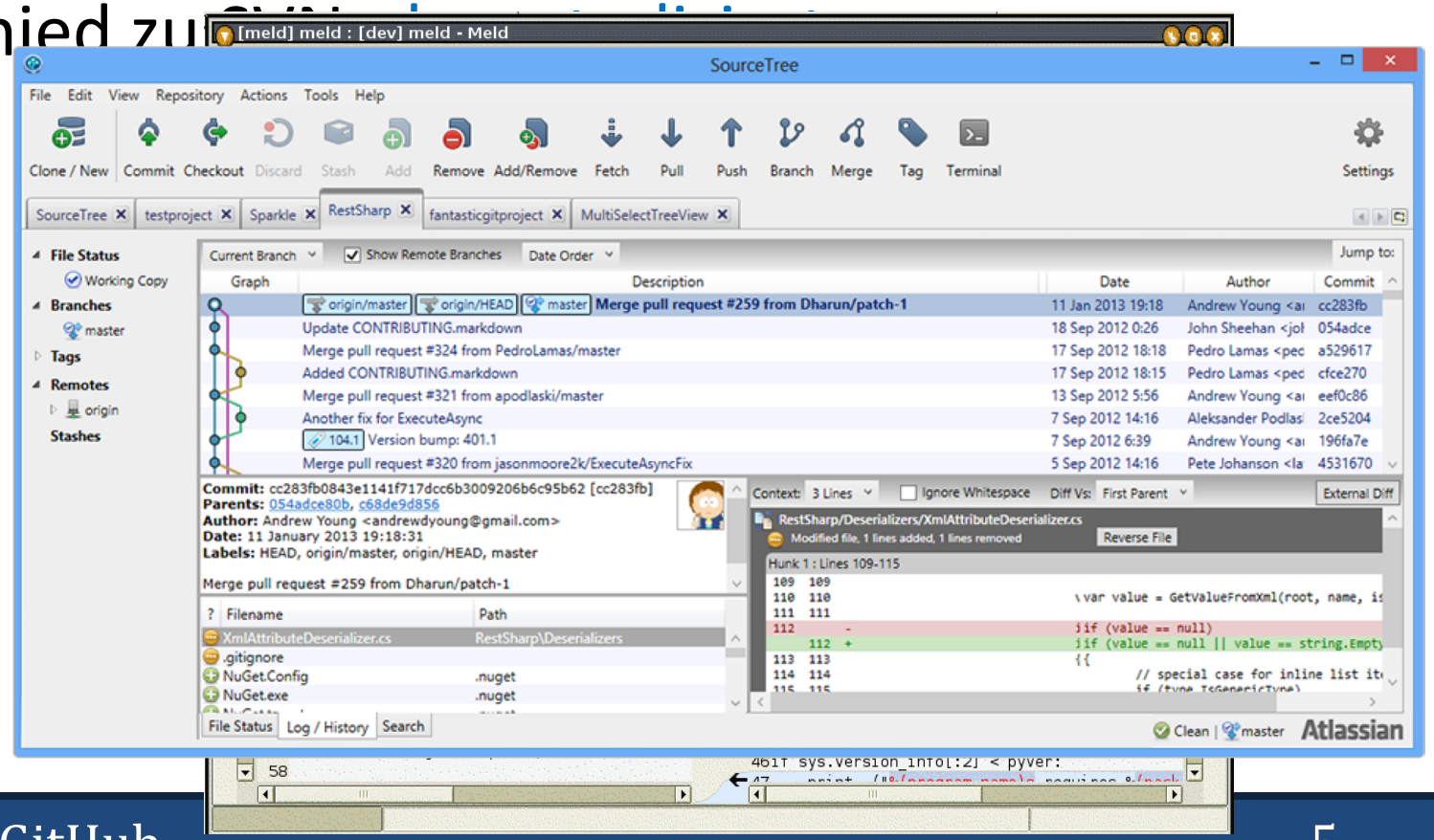
- Revisionsgeschichte eines Projekts erhalten und nachvollziehen
- Kollaboration
- Konfliktvermeidung und -handling
- „Backup“ um Fehler rückgängig zu machen
- Änderungen und Alternativen gefahrlos ausprobieren
- Information und Logs zu den Änderungen

Versionierungssysteme

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

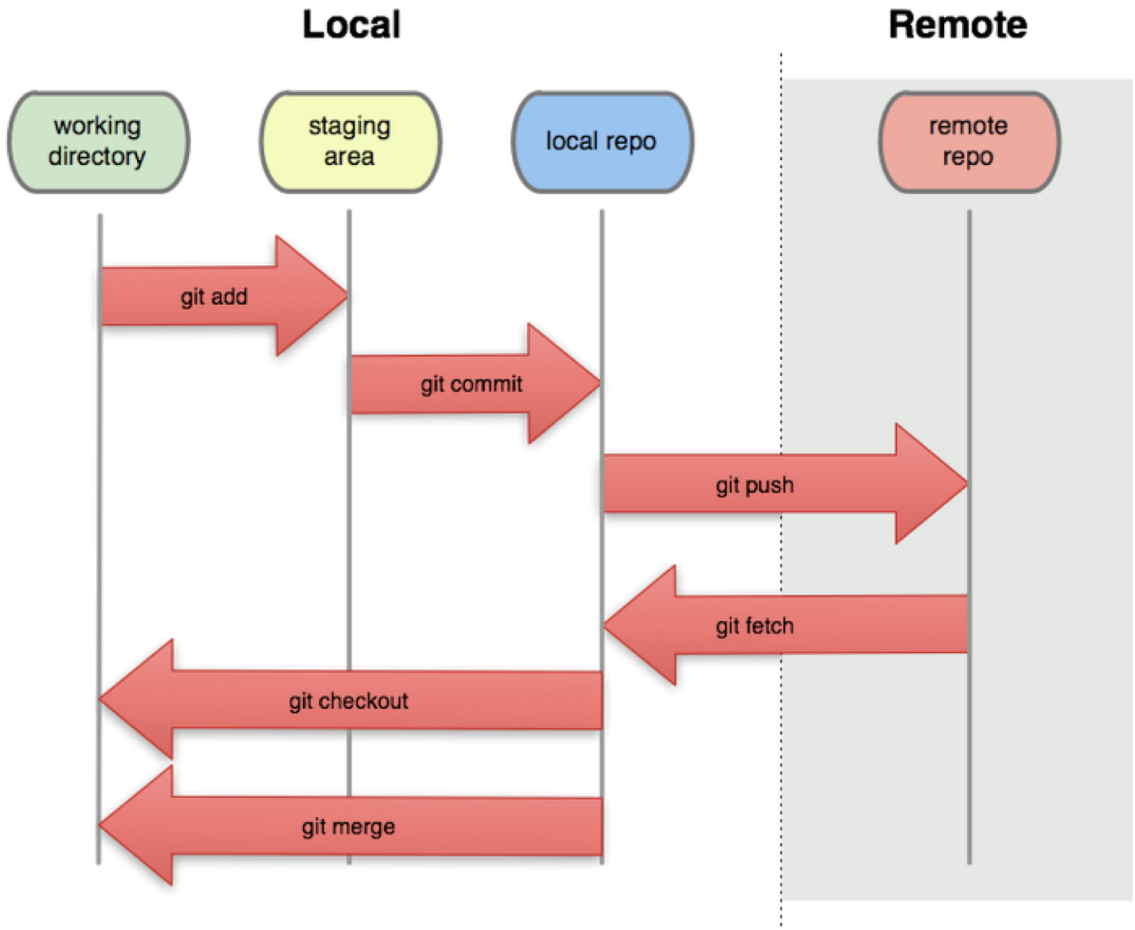
Git

- 2005 von Linus Torvalds entwickelt
- GitHub: öffentlicher Host von Git-Repositories
  - https://github.com/orgs/hu-berlin-semesterprojekte
- grundlegender Unterschied zu
- grafische Tools:
  - SourceTree (Windows, Mac)
  - **gitk (inklusive)**
  - TortoiseGit (Windows)
  - EGit (Eclipse)
  - Meld (diff & merge)



Git: Dezentralisierte Versionierung

- Remote Repository: Ein zentraler Server, der alle Dateien und Änderungen speichert
- Lokales Repository: Lokale Kopie des Remote Repository – erlaubt *offline* arbeiten
- Staging Area: Vorbereitung lokaler commits



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

Neues Repository anlegen

- Erstelle ein neues Verzeichnis und führe ...

```
git init
```

- aus
- Damit wird ein neues git-Repository angelegt
- Für eure Projekte existieren bereits Repositories:

https://github.com/hu-berlin-semesterprojekte/ttr-green-ws1819

Ein Repository auschecken

- Lokale Arbeitskopie eines Git-Repositories erstellen:

```
git clone git@github.com:hu-berlin-semesterprojekte/ttr-green-ws1819.git
```

- Hilfe

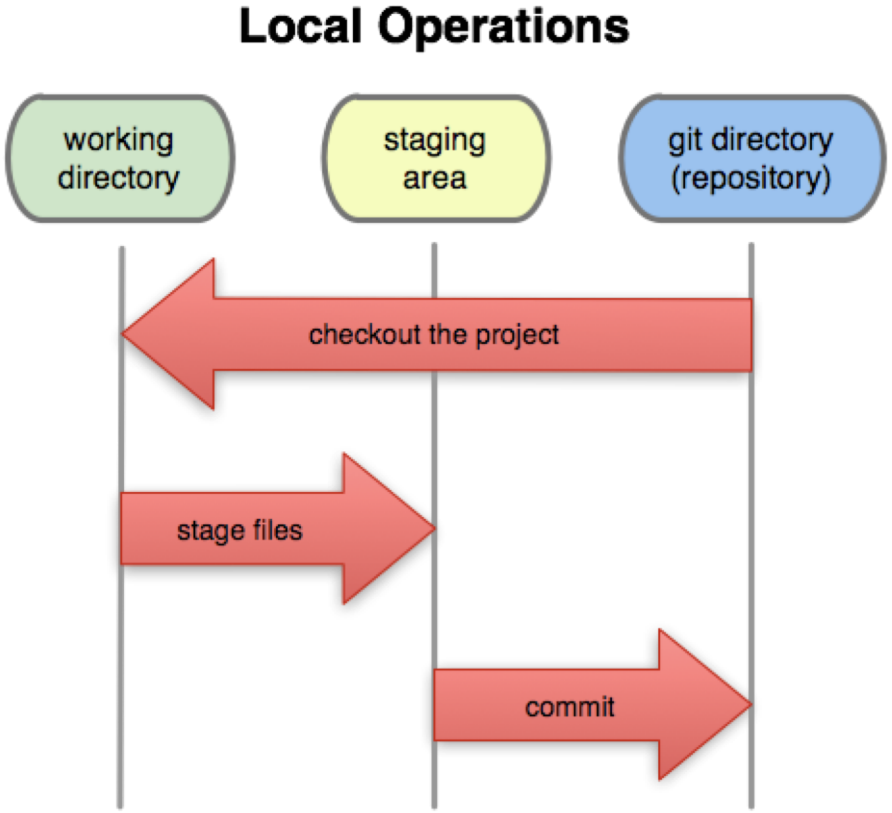
```
git help clone
```

- aktuellen Status abfragen
  - listet neue Dateien, veränderte Dateien, Konflikte etc. auf
  - vorher in das Verzeichnis des Repositories wechseln

```
git status
```

Lokales Repository

- Git arbeitet mit 3 (lokalen) Stages
  - Lokale Arbeitskopie
  - Staging Area
  - Git-Repository



https://git-scm.com/book/

- Der typische Workflow:
  - Ändere Dateien in der lokalen Arbeitskopie
  - Füge Dateien der Staging Area hinzu
  - Übertrage (Committe) Dateien von der Staging Area in das Repository

Lokales Repository

- Alle Änderungen anzeigen

  `git status`

- Datei zur Stage hinzufügen (Commit vorbereiten)

  `git add src/file.txt` oder `git add src/*`

- Datei aus der Stage löschen

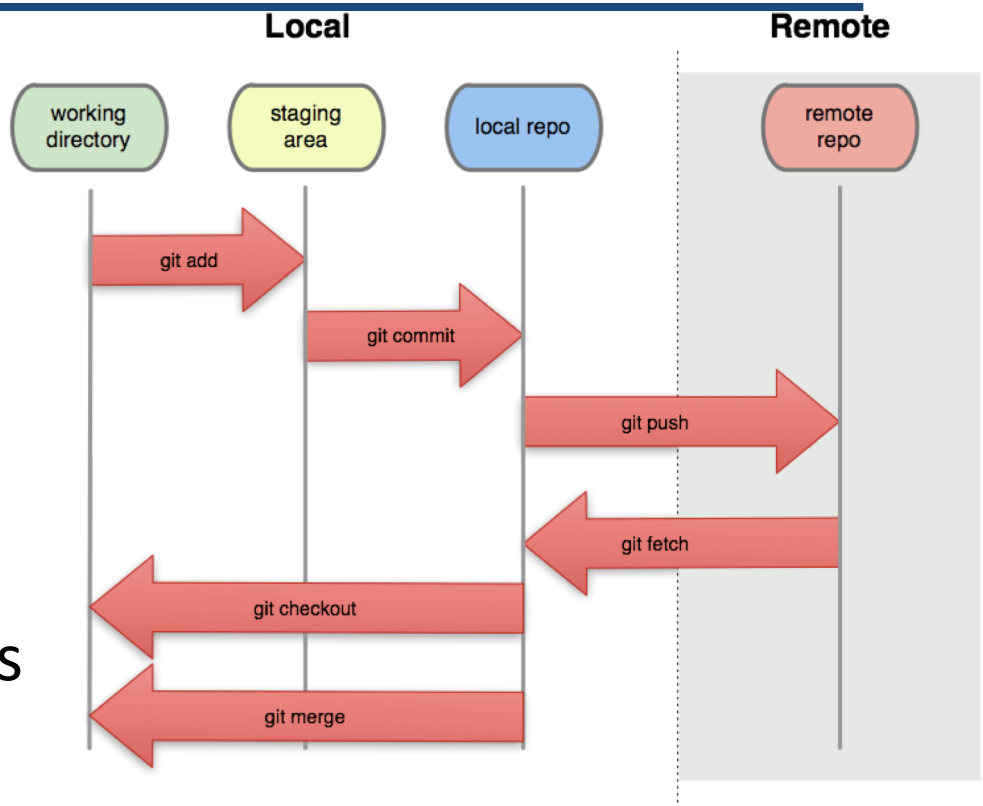
  `git rm file.txt`

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

  `git commit –m "commit message"`

- Vom lokalen Repository in einen Branch (master) des Remote-Repositories (origin) übertragen

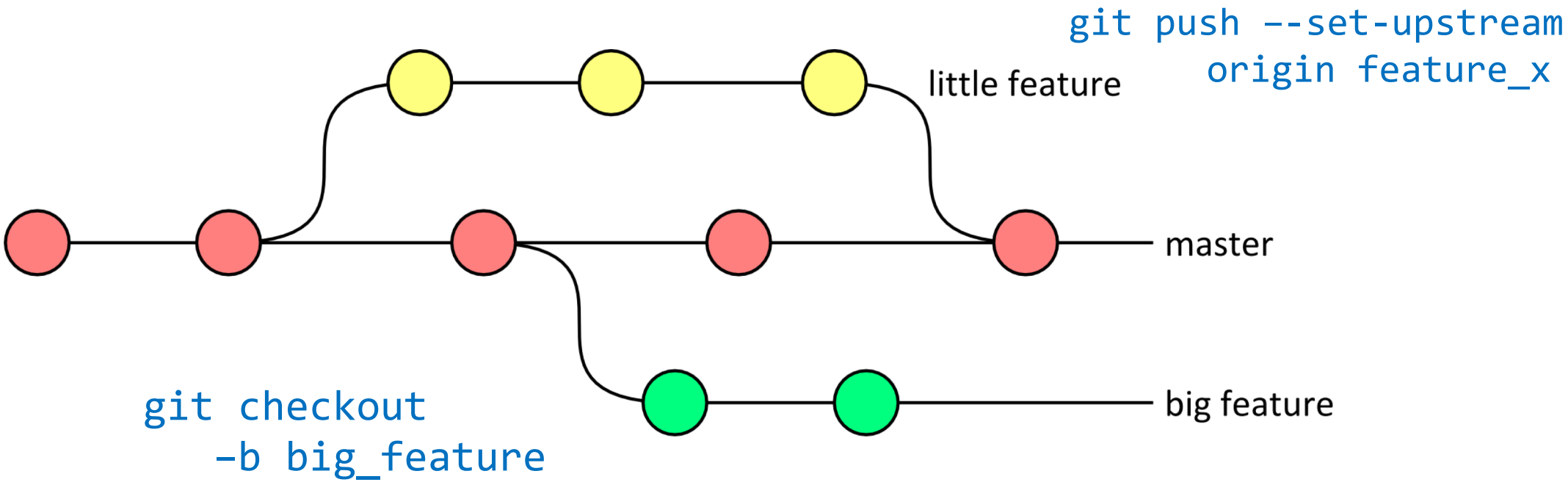
  `git push origin master`



https://git-scm.com/book/

Befehle: Branching

- Branches werden verwendet, um verschiedene Funktion isoliert voneinander zu entwickeln
- Branches werden später mit dem Master zusammengeführt



Befehle: Branching

- Neuen Branch erstellen und zu diesem Branch wechseln

```
git checkout –b feature_x
```

- ... mit dem Branch arbeiten ...

```
git add file1.src file2.src
git commit –m “feature xyz“
git ...
```

- Änderungen hochladen

```
git push –-set-upstream origin feature_x
```

- Zurück zum master-Branch wechseln

```
git checkout master
```

Update & Merge

- Änderungen am zentralen Repositoriy in das lokale Repository übernehmen

```
git pull origin master
```

- Anderen Branch mit deinem zusammenführen

```
git merge <other_branch>
```

```
git mergetool
```
(grafisch)

- Manuell behobene Konflikte als behoben markieren

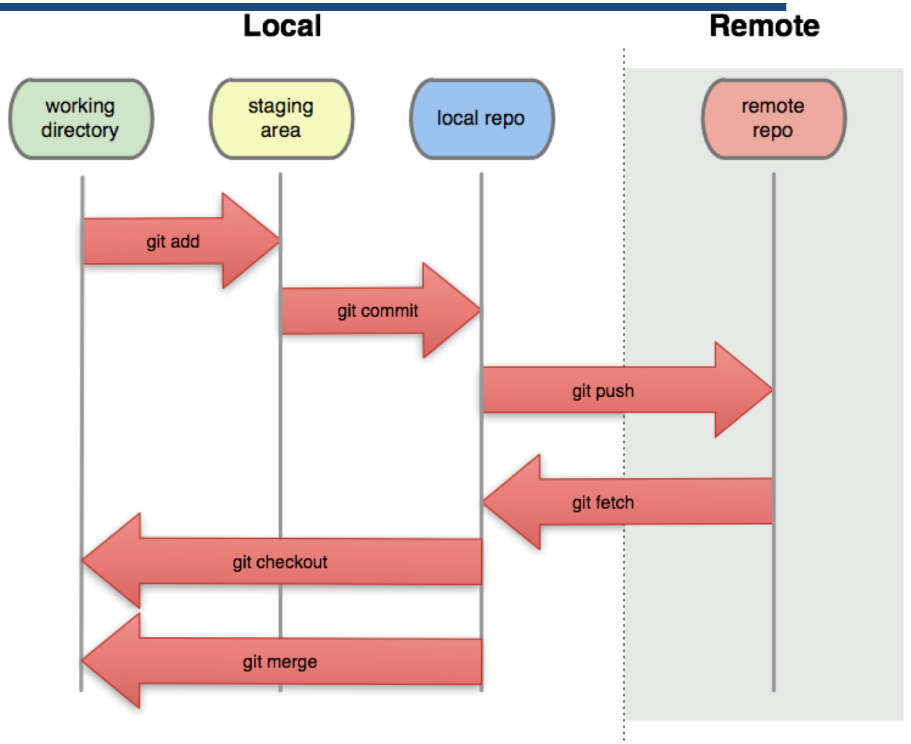
```
git add file.txt
```

- Probleme: lokale Datei durch HEAD-Revision ersetzen

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

- lokale Änderungen komplett rückgängig machen

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



https://git-scm.com/book/

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 (zB kompilierte Dateien), 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 diverse Projekte

- Eine Ansammlung nützlicher .gitignore-Dateien:
  https://github.com/github/gitignore

GitHub with Unity

- Weiterführende Informationen
  http://www.studica.com/blog/how-to-setup-github-with-unity-step-by-step-instructions

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 kann man jdemem Repository ein Wiki hinzufügen
  - Bilder können per URL eingebunden werden
- Hervorragend geeignet zur Bearbeitung gemeinsamer Dokumente
  - Roadmap
  - Übersicht
  - Use-Cases
  - Architektur
  - Erste Schritte
  - Verwendung der Software
  - etc.

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-Freitag im Team treffen (!)
  - „Daily“ Scrum
  - Evtl. Technical Refinement
- Nächste Woche Montag ab 13 Uhr:
  - Backlog Grooming: Vorstellen neuer User Stories (Observer)
  - Weitere Vorträge: Unit Tests, Continuous Integration, Coding Guidelines, ...
- Fragen? Probleme? → E-Mail an Scrum-Master oder an mich