# Software Engineering – Blatt 5

Rasmus Diederichsen Felix Breuninger
{rdiederichse, fbreunin}@uos.de

26. November 2014

### Aufgabe 5.1: SVN und GIT

#### a) SVN

Ein neues leeres Repository wird serverseitig mit

```
svnadmin create <directory>
```

angelegt. Der lokale Benutzer kann mit

```
svn checkout <url> <directory>
```

das hinter der `url` befindliche Repository in das lokale Verzeichnis `dir` klonen. Damit werden alle im Repository enthaltenen Dateien in das Verzeichnis kopiert. Hat man im lokalen Verzeichnis neue Dateien erzeugt, kann man diese per

```
svn add <file1> <file2> ...
```

dem lokalen Repository hinzufügen. Mit

```
svn commit -m <message>
```

werden die Änderungen (Hinzugefügtes, Entferntes, Geändertes) auf den Server übertragen. Möchte man eine lokale Änderung rückgängig machen, so kann man dies vor dem Commit mit

```
svn revert <file>
```

tun. Die Datei wird dann auf den Status nach dem letzten `update` zurückgesetzt. Möchte man Dateien umbenennen oder Verschieben, muss man `svn` darüber in Kenntnis setzen und muss den Betriebssystem-kommandos `mkdir`, `mv`, `cp` und `rm` ein "`svn`" voranstellen, um die Änderungen auch dem Repository mitzuteilen. Beim nächsten Commit werden diese dann auf den Server übertragen.

Will man einen neuen Branch erzeugen, kann man dies direkt auf dem Server tun, ohne selbst eine Arbeitskopie zu besitzen. Für SVN ist ein Branch einfach nur eine Kopie eines Verzeichnisbaumes. Per Konvention enthält jedes Projekt die Unterverzeichnisse `trunk` (Hauptentwicklungspfad), `branches` (Verzweigungen) und `tags` (stabile Snapshots). Neue Branches werden in `branches` angelegt, z.B. durch

```
svn copy file://<path to repo>/<project name>/trunk  \
         file://<path to repo>/<project name>/branches/dev \
         -m ‘‘<message>’’
```

. Hierbei wird ein neuer Branch `dev` auf dem Server erstellt. Nach einem `update` kann man nun darin arbeiten. Nach dem selben Muster erstellt man Tags als namentlich ausgezeichnete Unterverzeichnisse von `tags`.

Will man den Branch wieder in den `trunk` mergen, navigiert man in ihn hinein (in der lokalen Kopie) und führt

```
svn merge ^/<project name>/trunk
```

aus, wobei "^" für die URL des Repos steht. Man merged einen Branch `b1` in einen Branch `b2`, indem man sich im Verzeichnis `b2` befindet und

```
svn merge ^/<project name>/b1
```

aufruft. Änderungen in `b1` werden so in `b2` übertragen. Will man einen privaten Branch wieder in den `trunk` mergen, ruft man

```
svn merge --reintegrate ^/<project name>[/branches]/<private branch>
```

aus dem `trunk`-Verzeichnis auf. Wurden an derselben Datei in beiden Branches unvereinbare Änderungen durchgeführt, so müssen diese aufgelöst werden, z.B. manuell. Nachdem man Konflikte bereinigt hat, kann man die Änderungen `commit`ten und den obsoleten privaten Branch löschen.

### b) GIT

Durch Aufruf von

```
git init
```

wird das aktuelle Arbeitsverzeichnis zu einem neuen Repository deklariert. Durch

```
git clone <url>
```

wird das Remote-Repository unter `url` in das aktuelle Arbeitsverzeichnis gespiegelt.

Ein bestehendes Arbeitsverzeichnis kann aktualisiert werden durch

```
git fetch
git merge
```

oder alternativ durch

```
git pull
```

welches beide o.g. Befehle in Einem zusammenfasst.

Dateien können einem lokalen Repository hinzugefügt werden durch

```
git add <file1> <file2> ...
```

und darauf folgend, neben allen weiteren Änderungen gegenüber des Remote-Repositories, durch

```
git commit -am <message>
```

als neuer Snapshot mit der Bemerkung `message` angelegt. Die angelegten Snapshots werden mit

```
git push <alias> <branch>
```

an die Remote-Quelle `alias` (im üblichen Fall `origin`) vom lokalen `branch`-Zweig übertragen und dort als `branch`-Zweig angelegt.

Lokale Änderungen können zurückgesetzt werden auf den Stand des letzten commits durch

```
git checkout -- <file1>
```

für eine einzelne Datei oder

```
git reset --hard HEAD
```

für das gesamte Repository.

Einzelne Dateien oder Ordner können umbenannt/verschoben werden durch

```
mv <file1> <file2>
git add <file2>
git rm <file1>
```

was als

```
git mv <file1> <file2>
```

zusammengefasst werden kann. Dateien können gelöscht werden durch

```
git rm <file1>
```

Um dem HEAD, also dem aktuellsten commit einen Tag `name` mit Anmerkung `message` hinzuzufügen, wird

```
git tag -a <name> -m <message>
```

ausgeführt. Ein Tag kann auch einem speziellen commit hinzugefügt werden durch

```
git tag -a <name> -m <message> <commit>
```

und eine Liste aller Tags kann mit

```
git tag
```

eingesehen werden. Um Tags auch remote verfügbar zu machen, wird push mit dem Parameter –tags ausgeführt

```
git push origin --tags
```

Der aktive Branch kann gewechselt werden zu `branch` durch

```
git checkout <branch>
```

, gegebenenfalls muss vorher ein Branch angelegt werden durch

```
git branch <branch>
```

. Um die Änderungen eines Branches `branch` in den HEAD-Branch zu übernehmen wird

```
git merge <branch>
```

ausgeführt, hierbei werden Änderungen die nicht im Konflikt stehen auch in der selben Datei automatisch kombiniert.

Ein Konflikt tritt allerdings dann auf, wenn z.B. die selbe Zeile in HEAD und <branch> auf unterschiedliche Weise modifiziert wurde. GIT markiert die im Konflikt stehenden Zeilen und der Konflikt muss lokal durch entsprechendes anpassen der Quelldatei behoben werden.

```
<<<<<<< HEAD
Der aktuelle Branch ist master
=======
Der aktuelle Branch ist BRANCH1
>>>>>>> branch1
```