

Gruppe 6 – Softwareprojekt 2014/2015 – Kognitive Suche

Technologierecherche
im Studiengang Medieninformatik
der Fakultät IMN

an der

HTWK Leipzig
Hochschule für Technik, Wirtschaft und Kultur (FH)

Sebastian Hügelmann

Leipzig, 2014

Inhaltsverzeichnis

Inhaltsverzeichnis	2
1 GitLab.....	3
1.1 Erste Schritte für das Softwareprojekt 2014/2015.....	3
1.2 Branches und Remotes zum Projekt.....	10

1 GitLab

GitLab ist eine Open-Source-Software um gemeinsam an einem Code für Softwareprojekte zu arbeiten. Innerhalb dieser Software können Repositories (sog. Lager) eingerichtet und verwaltet werden. Diese Lager werden über mehrere Ebenen der Zugriffskontrolle gesichert. Die Nutzer sind in der Lage, den geschriebenen und hochgeladenen Code von anderen Nutzern einzusehen und mit den vorhandenen zu verschmelzen. Dieses Verschmelzen erfolgt über die sog. *merge requests* und ist die Hauptfunktion für das Bestehen einer einheitlichen Version. Neben der Versionsverwaltung/-kontrolle haben die Nutzer die Möglichkeit, eigene Aufgaben festzulegen, ein eigenes Wiki zu erstellen oder auch andere Dateien hochzuladen um diese mit den Projektteilnehmenden zu teilen.

1.1 Erste Schritte für das Softwareprojekt 2014/2015

Es gibt verschiedene Software um die Versionsverwaltung durchzuführen. In dieser Anleitung wird der einfachste nicht der bequemste Weg beschrieben. Als erstes werden die nötigen Werkzeuge installiert um Git zu nutzen. Nutzer von MAC und Linux benutzen folgende Software: <http://git-scm.com/download/mac>

Für Windows gibt das Git Bash und Git GUI von msysgit unter: <http://msysgit.github.io/>




Abbildung 1: Webseite von msysgit.

Um Probleme mit dem Umgang von msysgit zu vermeiden sollte bei der Installation die Auswahl „über Windows Command Prompt“ starten verwendet werden.

den. Nun erkennt Windows die Git Befehle in der Kommandoparameterzeile (CMD.exe). Die CMD.exe ist über die Suche unter Start zu erreichen. Dabei sollte die CMD stets als Administrator ausgeführt werden. Nun folgt der Zugriff auf das Projekt. Der GitLab Server der HTWK ist über folgenden Link erreichbar:

https://141.57.10.222/users/sign_in

GitLab Community Edition



Open source software to collaborate on code
Manage git repositories with fine grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Sign in

LDAP Standard

mmustermann

.....


LDAP Sign in

Did not receive confirmation email? [Send again](#)

[Explore](#) [Documentation](#) [About GitLab](#)

Abbildung 2

Nach erfolgreichem Ignorieren der SSL Zertifizierung erscheint die in Abbildung 2 gezeigte Webseite. In den Feldern unter „Sign in“ ist der HTWK Zugang mit Benutzernamen und Passwort einzutragen. Da der Administrator bereits die aktuellen Softwareprojekte erstellt hat, muss dieser oder einer der Master im Team die Mitglieder zum jeweiligen Projekt einladen. Nach dem erfolgreichen Hinzufügen findet der Nutzer das Projekt mittig im Tab „Projects“.

 **Dashboard**

[Activity](#) [Projects](#) [Issues 0](#) [Merge Requests 0](#) [Help](#)

My Projects

All projects you have access to are listed here. Public projects are not included here unless you are a member

All 1

Administrator / Kognitive_Suche

Last activity: 34 minutes ago

Leave

Personal	0
Joined	1
Owned	0

Visibility

Private

Internal

Public

Abbildung 3: Übersicht der Projekte in denen der Nutzer eingetragen ist.

Nach dem Aufrufen des Projektes steht oben die SSH und Https Adresse des Projektes. In dieser Anleitung wird die Https Adresse verwendet.

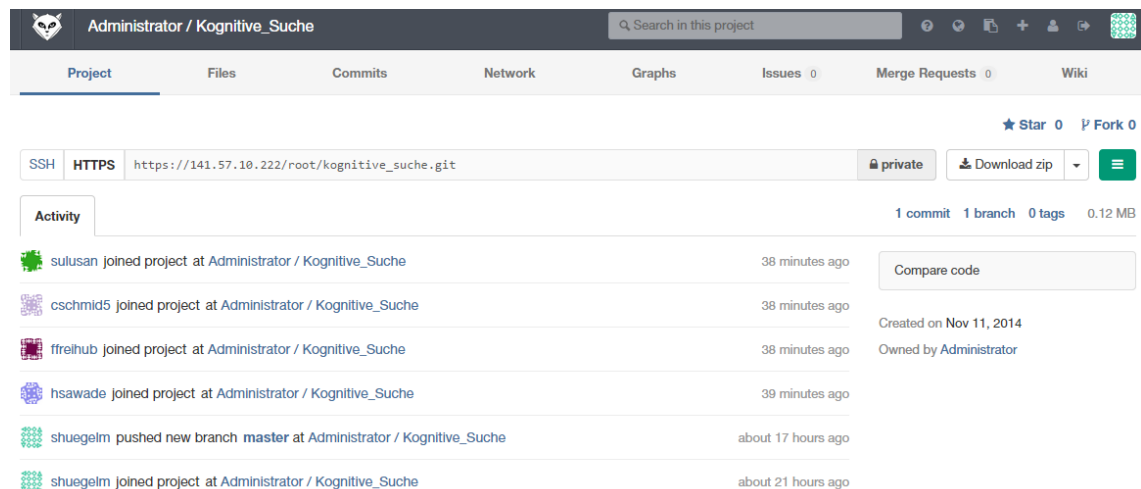


Abbildung 4: Log des Projektes mit SSH und Https Adresse.

Als erstes muss auf dem eigenen Computer der Username und die E-Mail eingetragen werden. Dies erfolgt über den Aufruf der CMD. Der Nutzer sollte sich einen Git Ordner anlegen. In dieser Anleitung hat der Nutzer den Pfad: „C:\Git“. Folgende Befehle müssen nun in der CMD eingegeben werden:

```
git config --global user.name "mmustermann"  
git config --global user.email max.mustermann@stud.htwk-leipzig.de  
git config --global http.sslVerify false
```

Dabei muss natürlich der Inhalt zwischen den „“ durch den eigenen Username und der eigenen E-Mail ersetzt werden (Beispiel: Abbildung 4).

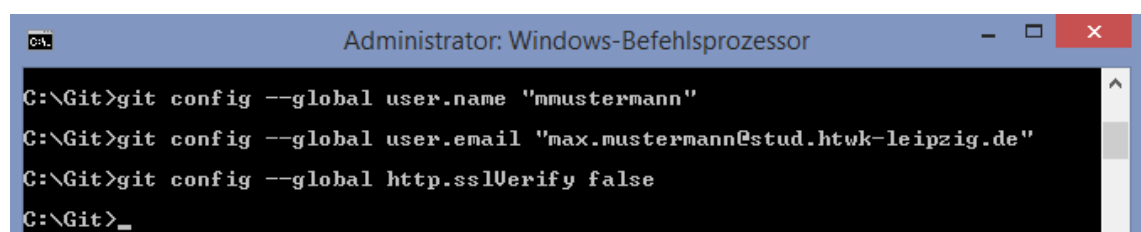


Abbildung 5: Konfiguration der globalen Config.

Zur Kontrolle (siehe Abbildung 5): `git config --global -l`

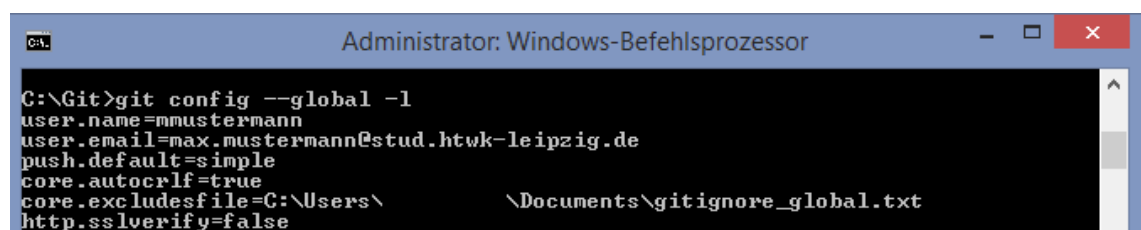
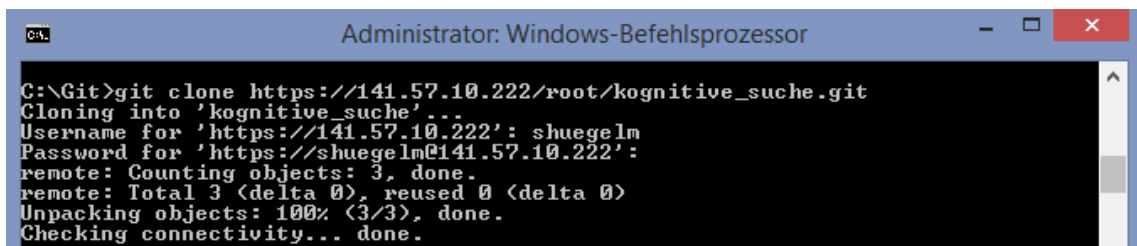


Abbildung 6: Überprüfung der Config.

Der Befehl „`http.sslVerify false`“ wurde eingegeben um zukünftige Zertifizierungsprobleme zu vermeiden. Im Anschluss erfolgt das Klonen der vorhandenen Projektdaten (engl.: working directory). Der Nutzer sollte sich im jeweiligen Git-Verzeichnis befinden. Ansonsten über den Befehl `cd` zum jeweiligen Verzeichnis navigieren. Um das jeweilige Lager (Repository) zu klonen muss der Nutzer folgenden Befehl eingeben (hier kommt die Https Adresse zum Einsatz):

```
git clone https://141.57.10.222/root/kognitive_suche.git
```

Danach fragt die Adresse nach dem HTWK Zugangs Benutzernamen und Passwort, wie beim „Sign in“ der GitLab Webseite. Das Passwort sollte beim Eingeben nicht sichtbar sein, wie im Beispiel in der Abbildung 6.



```
C:\Git>git clone https://141.57.10.222/root/kognitive_suche.git
Cloning into 'kognitive_suche'...
Username for 'https://141.57.10.222': shuegelm
Password for 'https://shuegelm@141.57.10.222':
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

Abbildung 7: CMD nach dem Eingeben des Klon-Befehls.

Jetzt befindet sich das gewünschte Repository im Git-Verzeichnis, wie in der Abbildung 7 zu sehen.

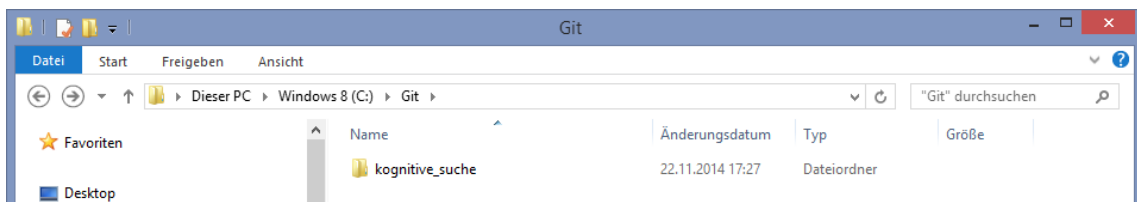
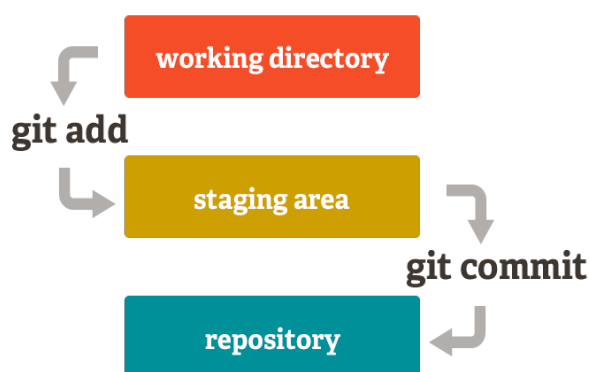


Abbildung 8: Geklontes Repository.



Dateien können im Git-Verzeichnis verschoben, erstellt und gelöscht werden. Jedoch müssen die veränderten, modifizierten oder gelöschten Dateien zum Git-Verzeichnis auf dem Server zurückgegeben werden. Dies erfolgt über die Vorgänge Staging und Committing (Bild links¹).

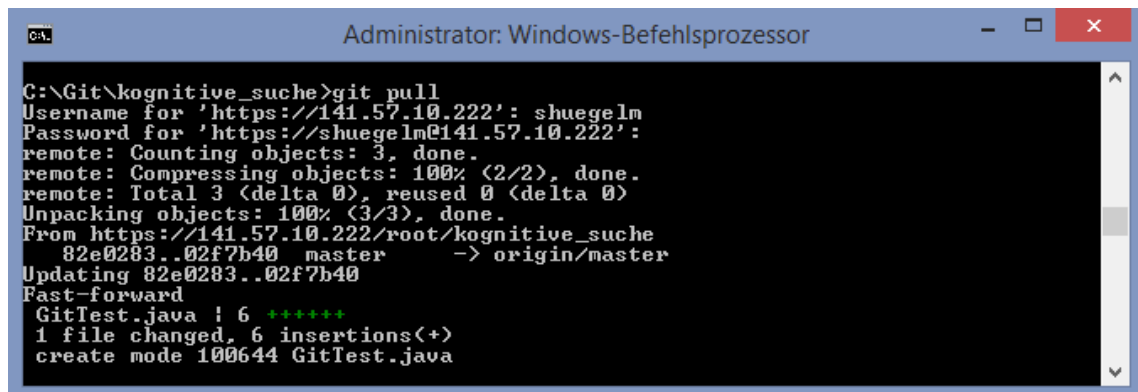
¹ Quelle: <http://git-scm.com/images/about/index1@2x.png>

Grundlegend funktioniert der Git Arbeitsprozess wie folgend:

1. Die Dateien (z.B. Java Klassen) werden im Arbeitsverzeichnis (working directory) bearbeitet
2. Mit dem „add“ Befehl werden diese Dateien in einer Datei im Git-Verzeichnis der sog. Staging Area markiert
3. Mit dem „commit“ Befehl werden diese Dateien dauerhaft in der lokalen Datenbank gespeichert
4. Mit dem „push“ Befehl werden diese Dateien zum Server gesendet

Mit den Befehlen im folgenden Beispiel soll der Nutzer die Dateien vom Lager ziehen und in der „GitTest.java“ seinen Namen eintragen. Anschließend soll er die veränderte Datei wieder hochladen.

```
git pull
```



```
C:\Git\kognitive_suche>git pull
Username for 'https://141.57.10.222': shuegelm
Password for 'https://shuegelm@141.57.10.222':
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://141.57.10.222/root/kognitive_suche
 82e0283..02f7b40 master -> origin/master
Updating 82e0283..02f7b40
Fast-forward
 GitTest.java | 6 ++++++
 1 file changed, 6 insertions(+)
 create mode 100644 GitTest.java
```

Abbildung 9: Ausgabe nach Eingabe des Befehls "git pull".

Der Befehl aus Abbildung 9 zieht die Dateien vom Repository und führte diese mit den eigenen Daten zusammen. Hingegen der Befehl „fetch“ die Daten nur zieht und dann mit „merge“ noch zusammengeführt werden muss.

```
git fetch //zieht die Daten vom Remote=origin und Branch=master
git merge //im aktuellen Branch
```

Der Nutzer hat jetzt die Möglichkeit, die GitTest.java in seinem Editor zu öffnen, dort zu bearbeiten, abzuspeichern und anschließend wieder zum Server zu schicken. Folgende Befehle können dabei verwendet werden:

```
//schiebt modifizierte, neue und gelöschte Dateien in die Staging Area
git add -A
//schiebt modifizierte und neue Dateien in die Staging Area
git add .
//schiebt modifizierte und gelöschte Dateien in die Staging Area
git add -u
//sendet nur die benannte Datei in die Staging Area
```

```

git add GitTest.java
//sendet alle Dateien aus der Staging Area mit einer Nachricht(-m) zur
//Datenbank
git commit -m "Datei GitTest.java geaendert"
//sendet nur die benannte Datei mit einer Nachricht zum Server
git commit GitTest.java -m "Name eingetragen!"
//Übermittelt die Dateien in das Repository auf dem Server
git push [OPTIONEN] [REMOTE] [BRANCH]
//Durch das Klonen ist REMOTE=origin und BRANCH=master
git push origin master

```

Dies sind die wichtigsten Befehle zum Klonen, Herunterladen und Updaten. In unserem Beispiel haben wir das Repository bereits vor der Abbildung 8 geklont und dort befindet sich die „GitTest.java“ bereits im Ordner. Sollten neue Dateien hinzugekommen sein, muss der Nutzer das Repository neu „ziehen“ mit:

```

//zieht Dateien aus dem Repository=origin und aus dem Branch=master
git pull origin master

```

```

C:\Git>cd kognitive_suche

C:\Git\kognitive_suche>git pull origin master
Username for 'https://141.57.10.222': shuegelm
Password for 'https://shuegelm@141.57.10.222':
From https://141.57.10.222/root/kognitive_suche
* branch                master       -> FETCH_HEAD
Already up-to-date.

C:\Git\kognitive_suche>

```

Abbildung 10: Ausgabe nach dem Befehl "git pull origin master" bei aktuellen Verzeichnis.

Nun soll der Nutzer seinen Namen mit einem Editor in die GitTest.java eintragen, also soll folgender Befehl sich unter den anderen einreihen:

```
System.out.println("Name");
```

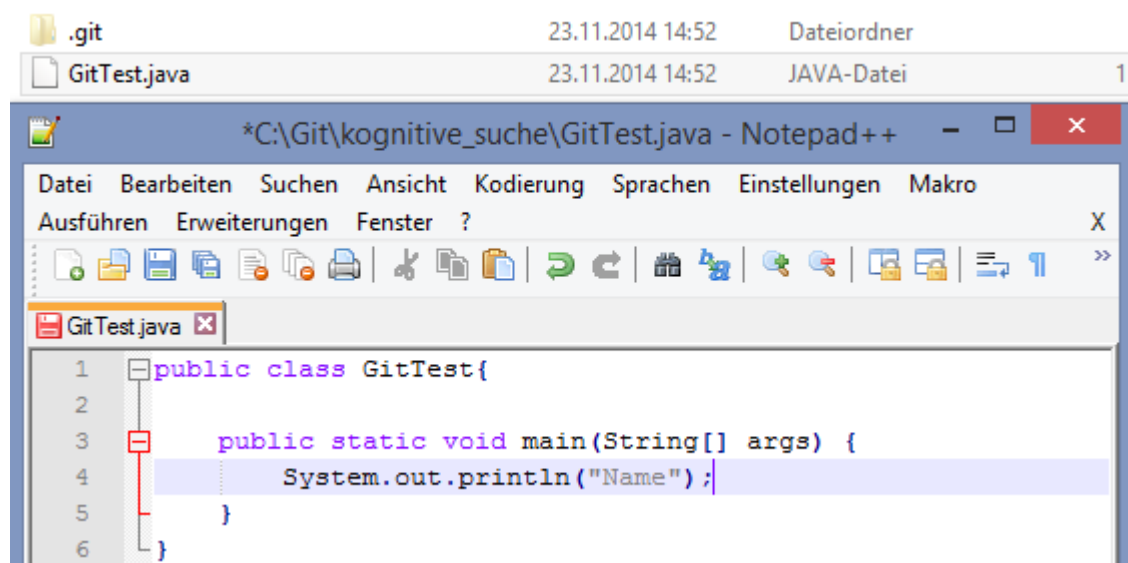
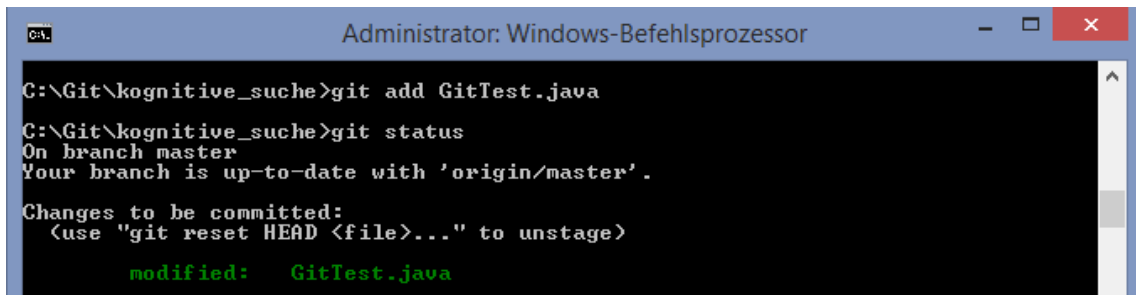


Abbildung 11: Programmcode der GitTest.java Datei.

Nach erfolgreichem Abspeichern muss die Datei nun hochgeladen werden:

```
git add GitTest.java
git status
```



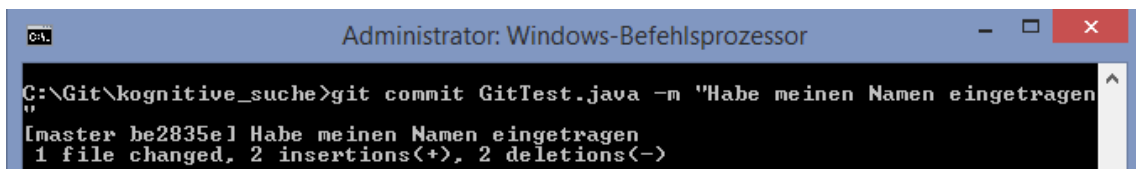
```
C:\Git\kognitive_suche>git add GitTest.java
C:\Git\kognitive_suche>git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   GitTest.java
```

Abbildung 12: Zeigt den momentanen Inhalt der Staging Area.

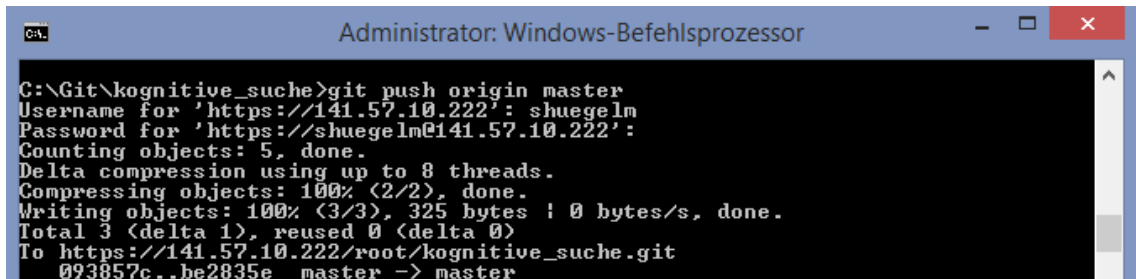
```
git commit GitTest.java -m "Habe meinen Namen eingetragen"
```



```
C:\Git\kognitive_suche>git commit GitTest.java -m "Habe meinen Namen eingetragen"
[master be2835e] Habe meinen Namen eingetragen
1 file changed, 2 insertions(+), 2 deletions(-)
```

Abbildung 13: Zeigt die Nachricht des commits und die Anzahl modifizierter, gelöschter und neuer Dateien.

```
git push origin master
```



```
C:\Git\kognitive_suche>git push origin master
Username for 'https://141.57.10.222': shuegelm
Password for 'https://shuegelm@141.57.10.222':
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 325 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://141.57.10.222/root/kognitive_suche.git
    093857c..be2835e  master -> master
```

Abbildung 14: Zugangsabfrage und Information zum hochladen der geänderten Objekte.

Danach sollte ein Eintrag in der Aktivitätsübersicht des Projektes mit Nutzernamen, Ort des Pushs, Commit-Nummer (be2835e8) und Nachricht stehen.



Abbildung 15: Auszug aus der Aktivitätsübersicht nach erfolgreichem Push.

Auf der Übersicht der Dateien im Reiter „Files“ steht die letzte Commit-Nachricht und die dazugehörige Commit-Nummer. Klickt der Nutzer auf diese

Commit-Nummer sieht er die Änderungen. Im Beispiel wurden die Leerzeichen hinter den Namen entfernt.

Habe meinen Namen eingetragen

Showing 1 changed file with 2 additions and 2 deletions

Show diff stats ▼

GitTest.java		
1	1	public class GitTest{
2	2	
3	3	public static void main(String[] args) {
4	-	System.out.println("Tim ");
5	-	System.out.println("Sebastian ");
	4	System.out.println("Tim");
	5	System.out.println("Sebastian");
6	6	}
7	7	}
8	8	\ No newline at end of file
...	...	

Abbildung 16: Beispielsicht von Änderungen in einer Datei.

1.2 Branches und Remotes zum Projekt

Remotes sind Repositories von anderen Nutzern. Das Haupt-Repository ist das automatisch generierte sog. „origin“. Es entsteht beim Anlegen des Projektes oder wird durch das Klonen übernommen. Dort sind im Allgemeinen alle Daten zu finden die für das Projekt von Bedeutung sind. Im Beispiel des Softwareprojektes 2014/2015 werden alle Nutzer innerhalb der Sprints an einem Repository arbeiten. Um zu vermeiden, dass die Software durch fehlerhafte Programmierung durch einen Push nicht mehr funktioniert werden Branches (dt.: Zweige) angelegt. Der ebenfalls automatisch generierte und übernommene Hauptzweig ist der sog. „master“. Bei den Sprints wird dieser auf schreibgeschützt gestellt und kann nicht überschrieben werden. Die Nutzer sind in der Lage, alle Dateien vom Master-Branch herunterzuladen, aber pushen die Änderungen in einen anderen Zweig z.B. den Development-Branch. Dort haben die Nutzer die Option, einen Merge-Request zu stellen damit die Änderungen in den Master-Branch übernommen werden (zusammengeführt). Dies erfolgt meistens durch einen Maintainer (dt.: Instandhalter, Pfleger). Aus jedem aktuellen Stand der Entwicklung kein Zweig entspringen und zu einem späteren Zeitpunkt gelöscht

oder auch mit einem anderen Zweig zusammengeführt werden. Diese Zweige sind nichts anderes als Pointer auf bereits getätigte Commits.

```
//Branch wird erstellt
git branch BRANCHNAME
//Branch wird gelöscht
git branch rm BRANCHNAME
//Aktueller Branch mit einem benannten zusammengeführt
Git branch NICHTAKTUELLERBRANCH
//Befehl zum Wechseln zwischen Branches
git checkout BRANCHNAME
//Neuer Branch wird hochgeladen
git push origin NEUERBRANCHNAME
```

So ist jeder Nutzer in der Lage an seiner Aufgabe zu arbeiten und sein Ergebnis in einen eigenen Branch hochzuladen außer dem Development-Branch. Am Ende können alle nacheinander zusammengeführt werden.

Weitere Informationen zu Git findet sich gesammelt und auf Deutsch unter:

<http://git-scm.com/book/de/v1>