

## Versionsverwaltungssysteme

Die Verwendung von zeitgemäßen Entwicklungswerkzeugen eine Grundlage für das Erstellen von guter Software. Versionierungssysteme bzw. Versionsverwaltungssysteme (*version control system* VCS) sind eines der wichtigsten Tools im Werkzeugkoffer eines Softwareentwicklers (sie sind aber auch für andere Bereiche praktisch). Wie der Name schon sagt, dienen Versionsverwaltungssysteme dazu, verschiedene Versionen einer Datei zu verwalten. Damit kann man leichter Lösungsansätze ausprobieren und – falls etwas nicht funktioniert – leicht zur Ausgangsversion zurückkehren.

## Git

Git ist ein VCS, dass vom Linux-Vater Linus Torvalds entwickelt wurde. Es handelt sich dabei um ein verteiltes Versionsverwaltungssystem (*distributed version control system* DVCS). Dabei hat jeder Client die gesamte Versionsgeschichte eines Projekts gespeichert. So kann jeder Client auch als Server arbeiten, aber es ist empfohlen einen der Clients ausschließlich als Server zu betreiben.

### Wie arbeitet git?

Git arbeitet grundsätzlich lokal. Das bedeutet, dass zunächst die gesamte Versionierung auf dem lokalen Computer stattfindet. Dafür sind folgende Bereiche wichtig:

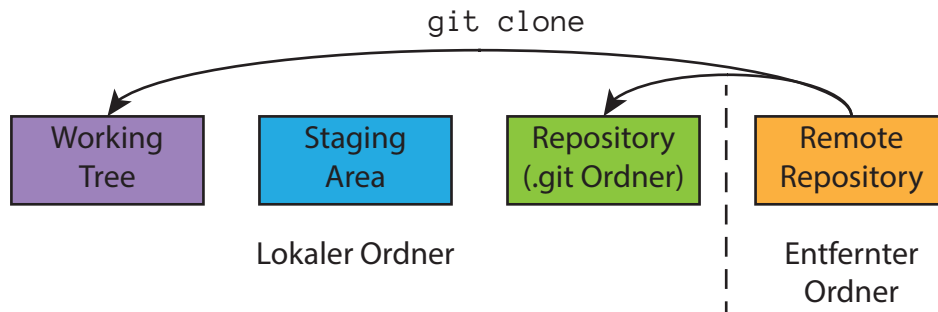


- **Repository:** Der Ort, an dem die Versionsverwaltung passiert. Dies wird durch einen eigenen Ordner namens `.git` realisiert. In diesem sind alle Dateiversionen gespeichert (das eigentliche Repository) sowie die notwendigen Daten für die Verwaltung der Versionierung.
- **Working tree:** Working tree bezeichnet den den aktuellen Zustand aller Dateien mit denen gerade in der Versionsverwaltung gearbeitet wird. Der Begriff wird oft synonym mit **working directory** oder **working copy** verwendet. Im Alltag wird jedoch der Begriff *working directory* auch als Begriff für den Ordner verwendet, wo die Dateien und Unterordner des *working trees* liegen. Dieser Ordner enthält auch den Repository-Ordner `.git`, kann aber auch Dateien beinhalten, die nicht in der Versionsverwaltung erfasst sind. Deshalb wird der Begriff **working tree** vorgezogen.
- **Index oder staging area:** Hier werden jene Dateiversionen zwischengespeichert, die für eine Übertragung ins Repository vorgesehen sind. Dieser Zwischenspeicher entspricht aber noch nicht dem tatsächlichen Repository, sondern dient nur der Datensammlung und der Vorbereitung einer Version.

Es gibt 2 Möglichkeiten diese Bereiche zu bekommen:

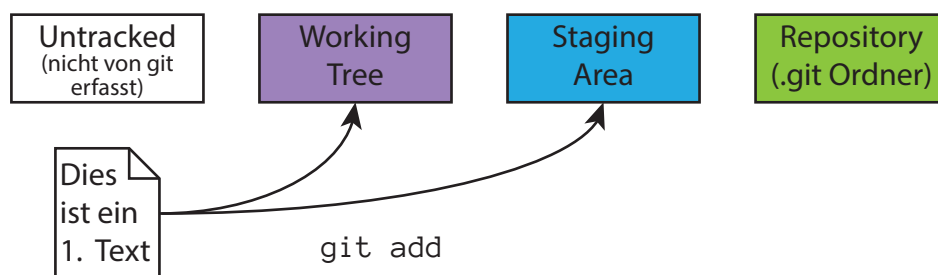
1. `git init`: Damit wird im aktuellen Verzeichnis ein `.git` Verzeichnis für das Repository und die Versionsverwaltung angelegt. Alle 3 Bereiche sind nach diesem Schritt leer (auch wenn im *working directory* Dateien vorhanden sind, diese werden nicht automatisch in den *working tree* übernommen).
2. `git clone`: Erstellt eine 1:1-Kopie eines anderen Repositories inkl. aller Versionspunkte. Dadurch wird sowohl das lokale Repository als auch der Working tree mit den Daten aus dem entfernten Repository (**remote repository**) befüllt. Als *remote repository* kann jedes beliebige git-Repository dienen (auch wenn es sich z.B. auf dem gleichen Computer befindet). Sehr häufig wird ein Repository von einem zentralen Dienst (wie z.B. Github) dafür verwendet, dies ist aber keinesfalls Voraussetzung. Ein `clone`-Aufruf von Github kann z.B. folgendermaßen aussehen:

```
git clone https://github.com/lvittori/emptyJavaProject.git
```

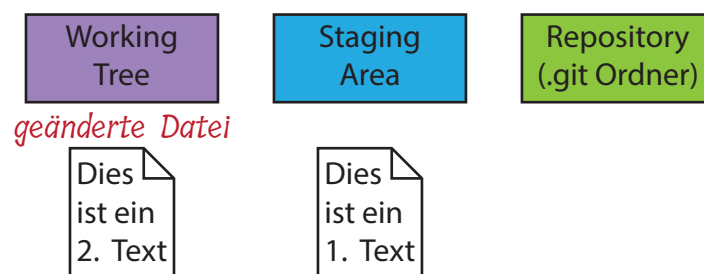


### Eine Datei Versionieren

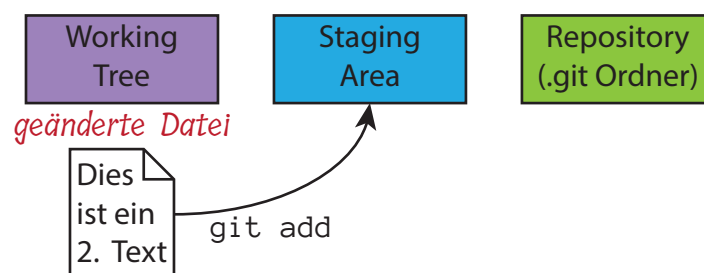
Bevor eine Datei im Repository als eigene Version abgelegt werden kann, muss sie mit `git add` zur Staging Area hinzugefügt werden. Eine Datei, die noch gar nicht von der Versionierung erfasst ist, wird damit auch zum Working Tree hinzugefügt.



Die Datei in der Staging Area ist als Snapshot abgelegt. Eine Änderung in der Datei im Working Tree wirkt sich nicht unmittelbar auf die Datei in der Staging Area aus.



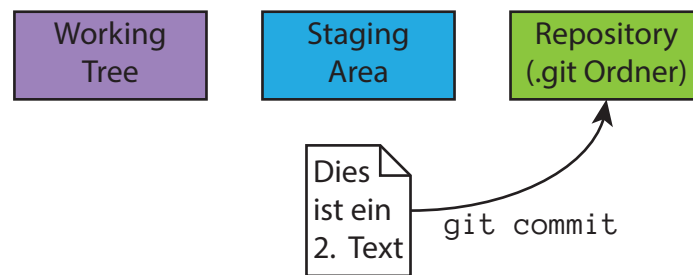
Möchte man diese geänderte Datei zur nächsten Version hinzufügen, muss zunächst wieder die geänderte Datei mit `git add` in die Staging Area kopiert werden.



Achtung! Der 1. Text wurde dabei nicht von git als eigene Version erfasst. Die Staging Area dient nur der Vorbereitung einer Version. In dem gezeigten Ablauf ist der erste Text vollkommen vom zweiten Text ersetzt worden und nicht mehr als eigene Version herstellbar.

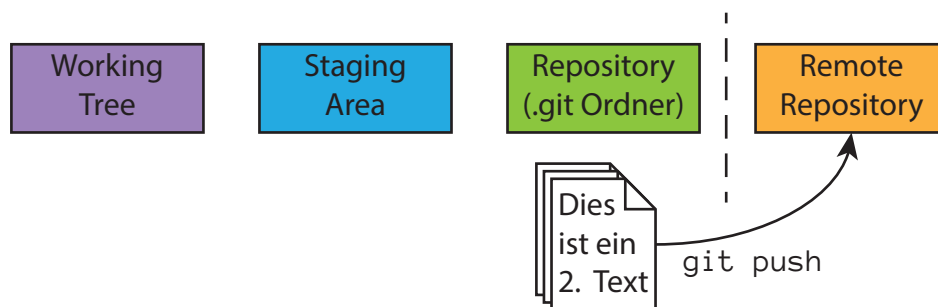
Eine eigene Version wird durch den Befehl `git commit` hergestellt. Erst dann werden alle Snapshots von der Staging Area in das eigentliche Repository übertragen (sofern sie gegenüber den Dateien im Repository Änderungen aufweisen). Ein commit sollte immer von einer aussagekräftigen commit-Message begleitet werden, die über die Änderungen dieser Version Auskunft gibt. Daher öffnet sich auch bei der Eingabe von `git commit` ein Editor-Fenster, in dem die commit-Message

eingetragen werden kann. Um dies zu vermeiden, kann mit der Option `-m` gleich direkt eine Nachricht angegeben werden, z.B: `git commit -m "Fehlerbehebung in der ersten Zeile"`



Dieser Vorgang kann beliebig oft wiederholt werden, um im lokalen Repository verschiedene Versionspunkte des Working Trees herzustellen. Dabei wird die aktuelle Version (die Version des letzten commits) als **HEAD** bezeichnet.

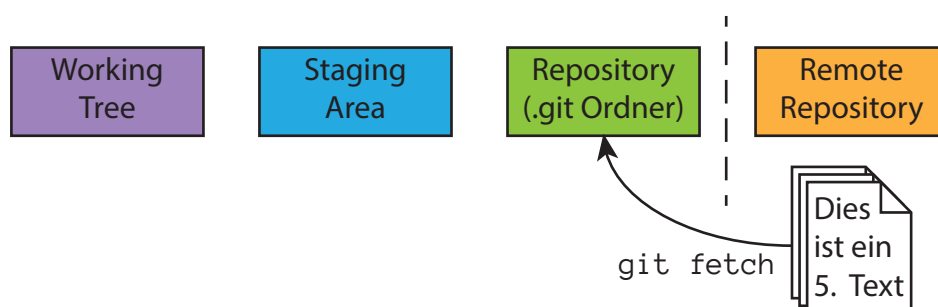
Um die Änderungen auch auf ein entferntes Repository zu übertragen, muss der Befehl `git push` verwendet werden. Dazu sollte für dieses entfernte Repository mittels `git remote` noch ein eigener Kurzname konfiguriert werden. Wenn ein Repository mittels `git clone` erstellt wurde, ist die URL der Quelle mit dem Namen `origin` bereits vor konfiguriert.



Mit `git push` werden alle Versionen im lokalen Repository an das Remote Repository übertragen. Bei einer Team-Arbeit empfiehlt es sich aber zuerst mit `git fetch` eventuelle zwischenzeitlich erfolgten Änderungen vom Remote Repository abzufragen und damit lokal zu verarbeiten, um Konflikte am Remote Repository zu vermeiden.

#### Eine Version aus dem Repository laden

Wie schon zuvor erwähnt lädt der Befehl `git fetch` die in einem Remote-Repository verfügbaren Versionen in das lokale Repository (sofern diese nicht bereits dort vorhanden sind).



Mit `git checkout` können dann Dateien aus dem Repository in den Working Tree übernommen werden, wobei je nach übergebenen Parametern entweder Dateien aus dem Repository oder der Staging Area verwendet werden.

Achtung! Dabei kann es notwendig sein, Dateien aus dem Repository mit den bereits vorhandenen Dateien zusammenzuführen. Diesen Vorgang nennt man „**merging**“. Mehr dazu findet ihr unter <https://git-scm.com/book/de/v1/Git-Branching-Einfaches-Branching-und-Merging> .

Weitere wichtige Befehle:

- `git status`: zeigt den Status deines Working Trees und deiner Staging Area an
- `git log`: zeigt die Commits im Repository an

## Github

Github ist eine Plattform für Softwareprojekte, die als Server für Git dienen kann. Durch seine einfache Weboberfläche ist Github sehr beliebt und wird zunehmend auch für andere Daten, wie z.B. dem Austausch von Dokumenten verwendet. Github ist benutzerzentriert, d.h. jedes Software-Projekt wird unter einem bestimmten Benutzernamen geführt.

### Voraussetzungen

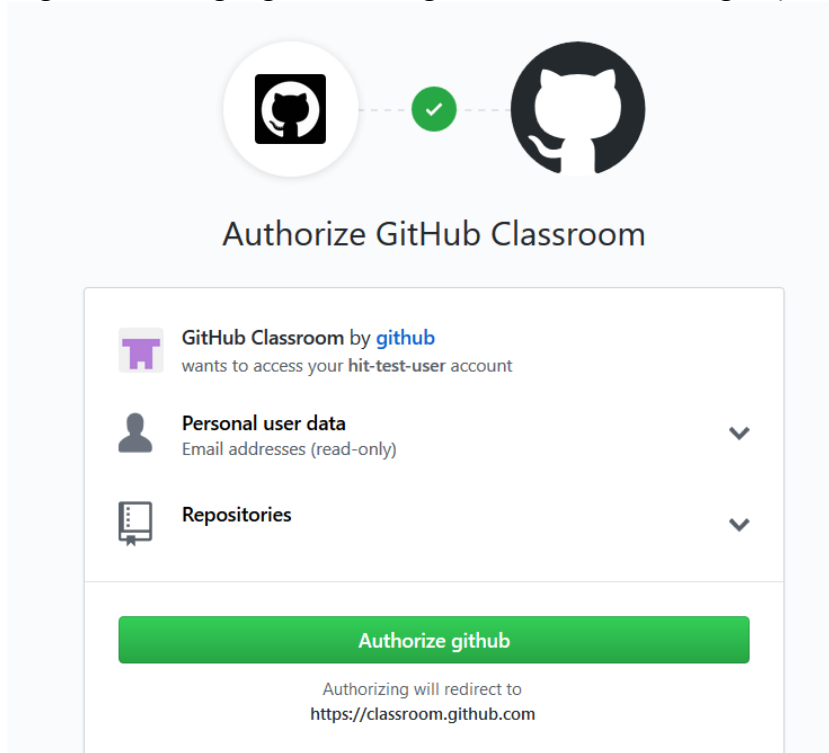
Damit Github classroom verwendet werden kann, muss ein eigener Github-Account vorhanden sein. Wer noch keinen Github-Account hat, kann sich auf <https://github.com> gratis registrieren. Der Account kann erst dann mit Github classroom verwendet werden, wenn die Email-Adresse bestätigt wurde.

### Github assignment – Erste Anmeldung

Ausgangspunkt für das Arbeiten mit Github classroom ist ein Link zu einem Assignment (in den Aufgaben auf der elearning-Plattform angegeben).

Bei der erstmaligen Nutzung von Github classroom müssen folgende Schritte durchgeführt werden (Screenshots von Walter Rafeiner-Magor zur Verfügung gestellt):

1. Zugriffsberechtigungen auf den github-Account bestätigen (nicht bei allen erforderlich):




2. Den eigenen Namen aus der Klassenliste auswählen, damit euer Account mit dem Eintrag der Klassenliste verknüpft wird.

Sollte euer Name dort nicht aufscheinen, dann gibt es zwei Möglichkeiten:

- a. ein anderer Schüler/eine andere Schülerin hat sich unter eurem Namen registriert

b. euer Name war auf der ausgeschickten Klassenliste nicht vorhanden

**Bitte nicht einfach überspringen**, sondern wirklich einen Namen auswählen – oder falls nicht vorhanden beim Lehrer bzw. bei der Lehrerin melden.

 Join the classroom roster


Your teacher has configured this classroom to pair GitHub accounts with identifiers. Please select yourself from the list below. You can also skip this step for now.

nachname, vorname, klasse
Test User
Zvachula Christian 3C
Yesildag Emre 3C
Thach Kevin 3C
Schuster Adrian 3C
Schmutz Stefan 3C
Schachenhofer Simon 3C

## Aufgabe annehmen

Nachdem der eigene GitHub-Account mit Github classroom verbunden wurde, können Aufgaben über den Aufgabenlink angenommen werden.

**TGM-HIT**  
@TGM-HIT


 Accept the **Hangman** assignment

Accepting this assignment will give you access to the **hangman-hit-test-user** repository in the **@TGM-HIT** organization on GitHub.

Accept this assignment

Nachdem die Aufgabe angenommen ist erzeugt GitHub ein persönliches Repository für diese Aufgabe. Je nach Aufgabe kann es sein, dass auch schon vorgegebener Code oder die Daten für ein Eclipse-Projekt in euer Repository hineinkopiert werden. Wenn alles funktioniert hat, bekommt ihr die Erfolgsmeldung zusammen mit dem Link auf euer persönliches Repository:

**TGM-HIT**  
@TGM-HIT

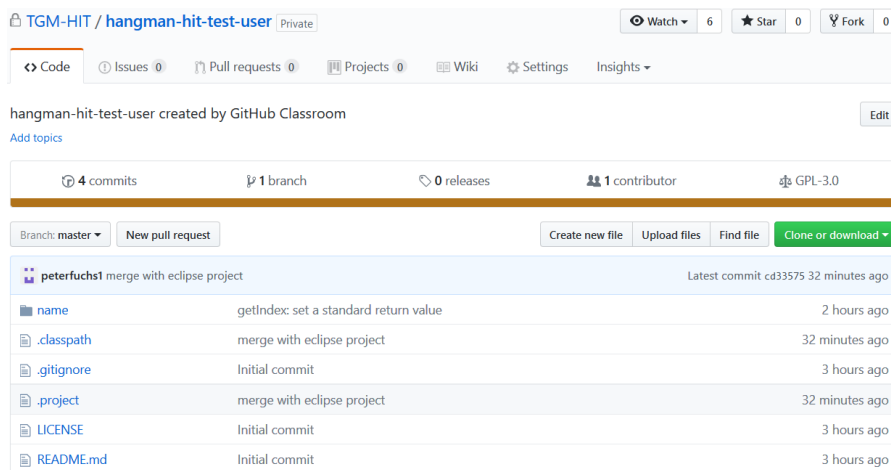
 Accepted the **Hangman** assignment

**You are ready to go!**

You may receive an invitation to join **@TGM-HIT** via email invitation on your behalf. No further action is necessary.

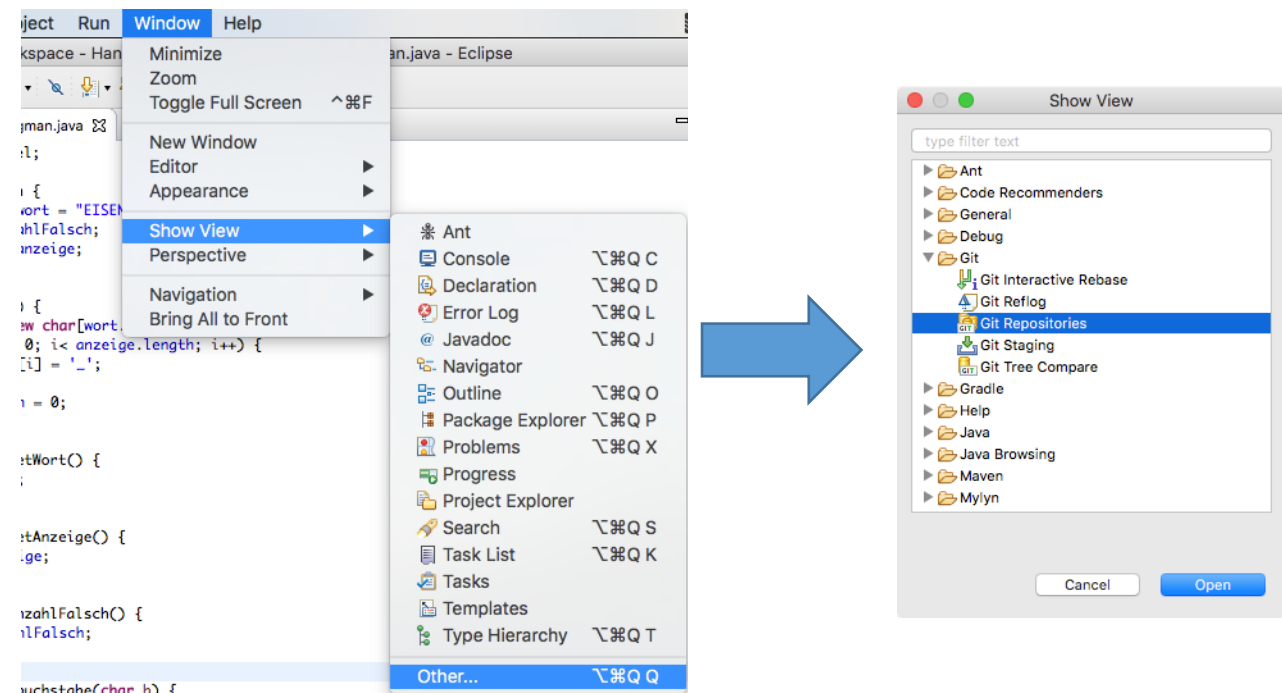
Your assignment has been created here: <https://github.com/TGM-HIT/hangman-hit-test-user>

Wenn ihr dem Link folgt seht ihr die Dateien eures nun am Server erstellten Repositories. Damit ist alles bereit, um das Repository auf den lokalen Rechner zu transferieren – zu klonen.

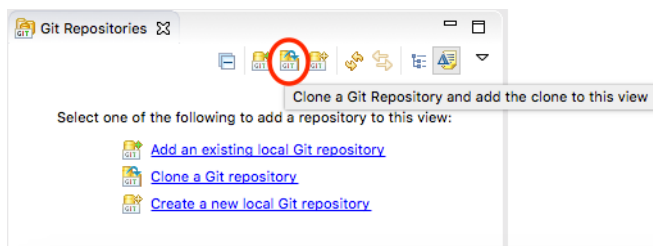


## Ein Github repository in Eclipse einbinden

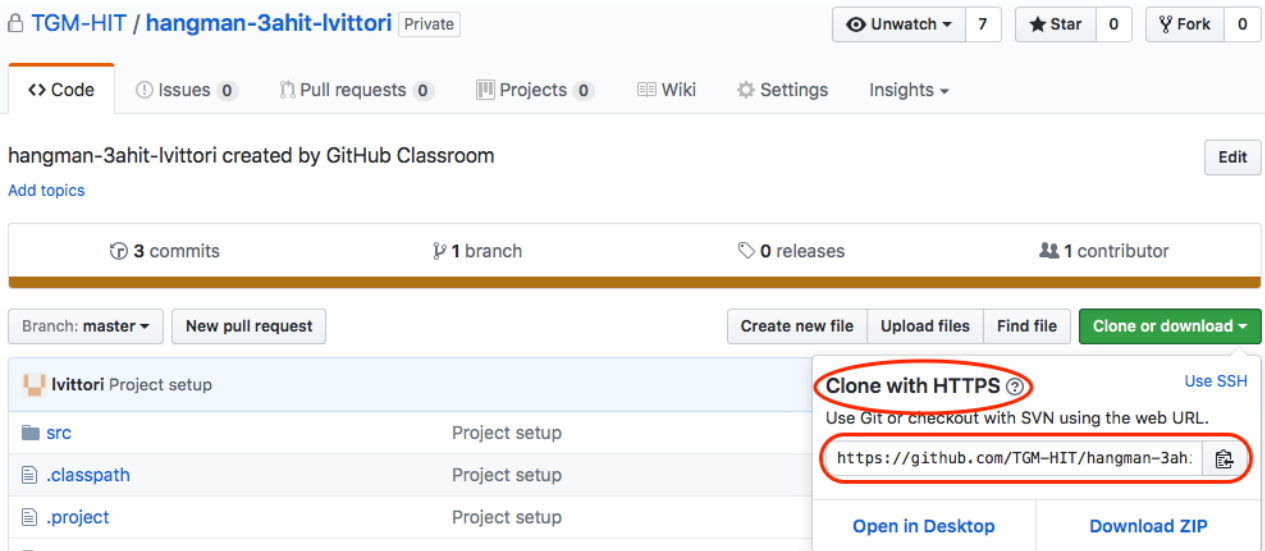
Im nächsten Schritt muss nun das für die Aufgabe erstellte Github repository auf den lokalen Rechner transferiert werden und ein Arbeitsverzeichnis (das gleichzeitig auch als Projektverzeichnis fungiert) angelegt werden. Dazu muss in Eclipse der Bereich für die Git Repositories eingeblendet werden:



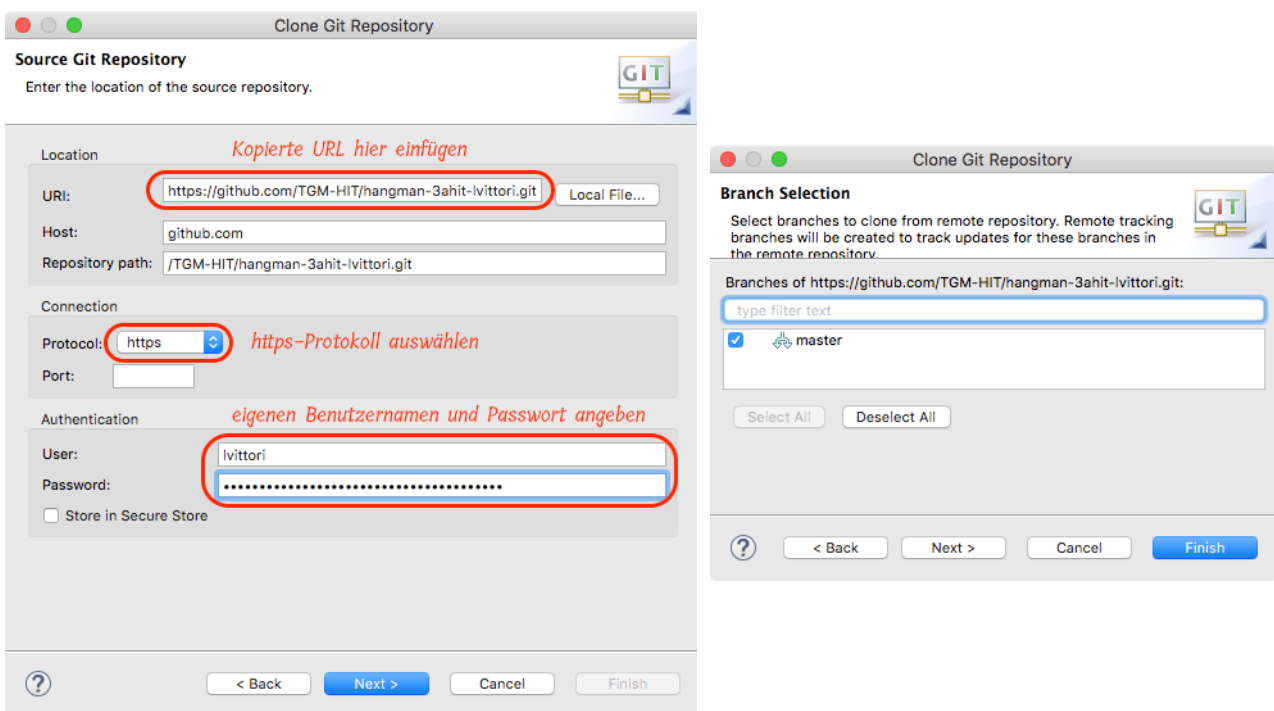
Nun wird ein eigener Bereich, der zur Darstellung der Git Repositories geöffnet. Hier findet sich die Schaltfläche, um ein Git repository von einem remote zu klonen und in Eclipse einzubinden.



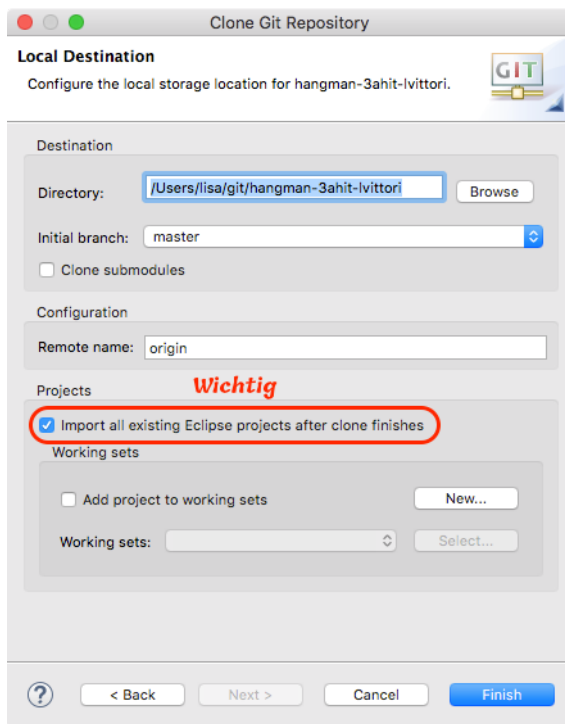
Für den nächsten Schritt braucht man die URI (*uniform resource identifier* ... für Daten im Internet ist das die URL ... *uniform resource locator*) des Github repositories. Diese kann man sich auf der Repository-Webseite anzeigen lassen:



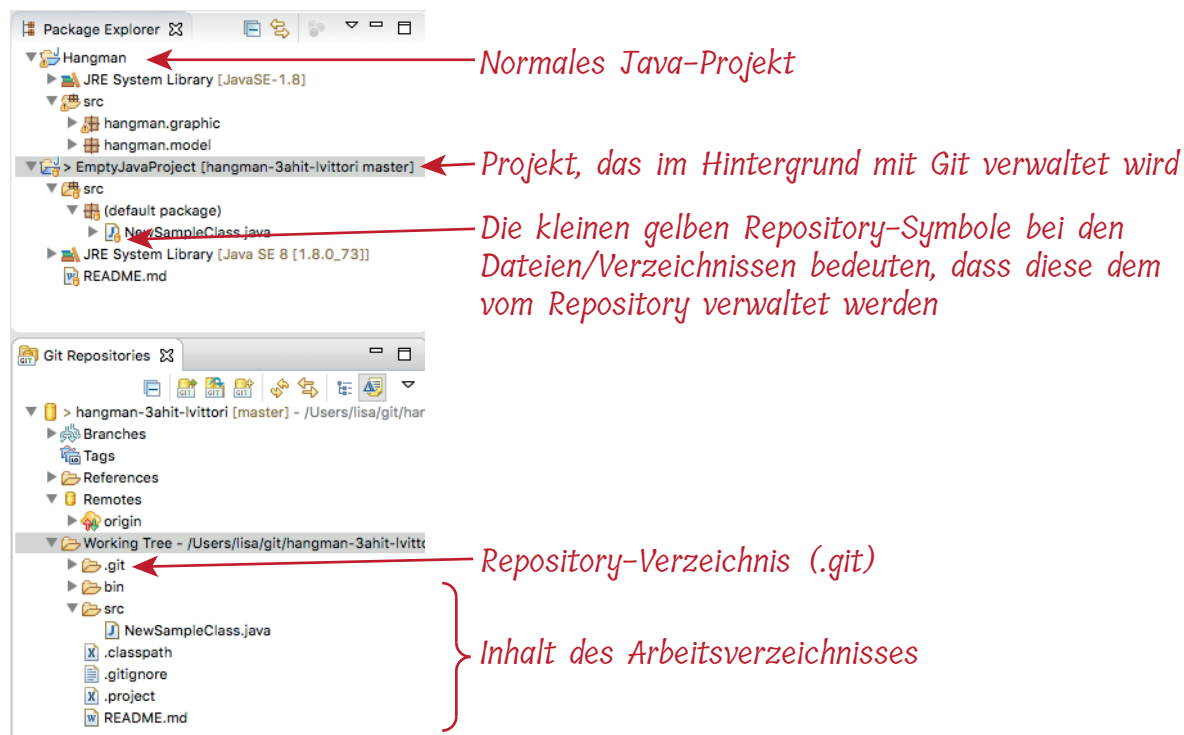
Diese URL wird im ersten Fenster, nachdem "Clone Git Repository" ausgewählt wurde, eingefügt. Die weiteren Daten bis auf den Benutzernamen und das Passwort sollten dann richtig voreingestellt sein. Danach muss man noch auswählen, welchen Entwicklungszweig man klonen möchte. Da das Projekt nur über einen Entwicklungszweig verfügt, sollte auch nur einer angezeigt werden (master).



Mit Next kommt man noch auf das Fenster, wo die Optionen für das Speichern im Eclipse-Workspace ausgewählt werden können. Dabei können die meisten Voreinstellungen übernommen werden. Es ist allerdings wichtig den Punkt "Import all existing Eclipse projects after clone finishes" anzuhaken. Sonst muss man anschließend noch manuell das Projekt aus dem Arbeitsverzeichnis in den Eclipse Workspace importieren.



Wenn der Klon-Vorgang abgeschlossen wurde, sollten in Eclipse die folgenden Bereiche sichtbar sein.



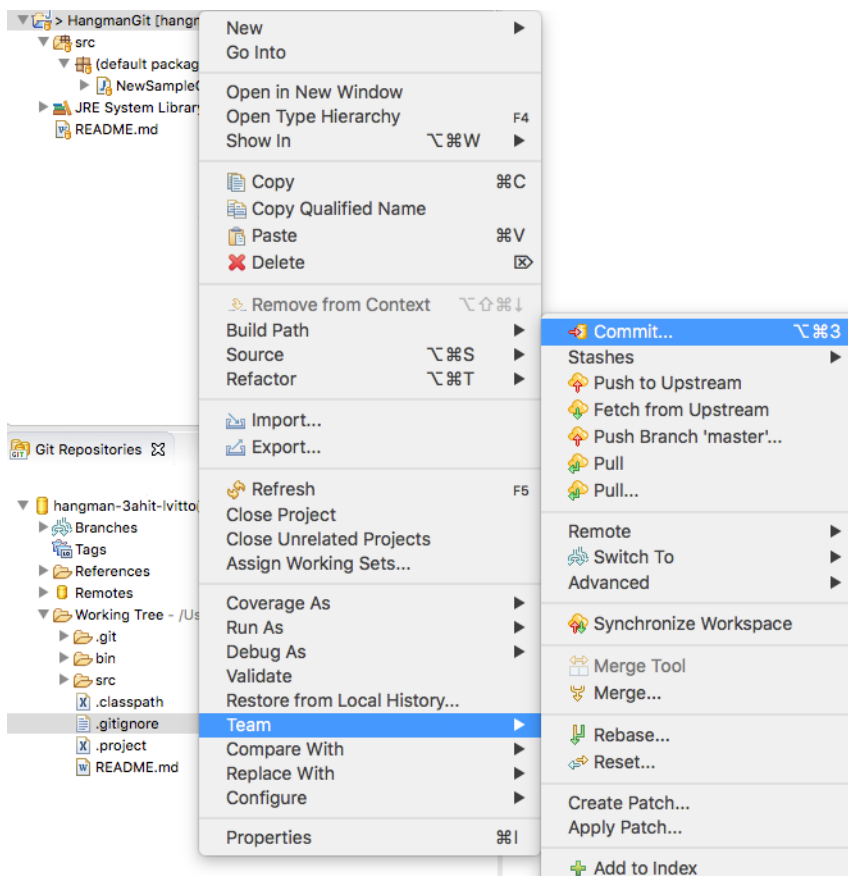
### Eine Änderung an den Server übertragen

Nach dem Klonen ist der Projektname u.U. nicht passend. Dieser kann mittels Refactor - Rename geändert werden. Die Änderung wird zwar vom System erkannt, doch noch wird sie nicht als Versionspunkt gespeichert (manchmal wird das Symbol erst nach dem nächsten Schritt aktualisiert).





Damit die Änderung auch als Versionspunkt gespeichert wird, ist ein "Commit" notwendig. Dieses kann über das Kontextmenü (rechte Maustaste auf den Projektnamen) ausgelöst werden:



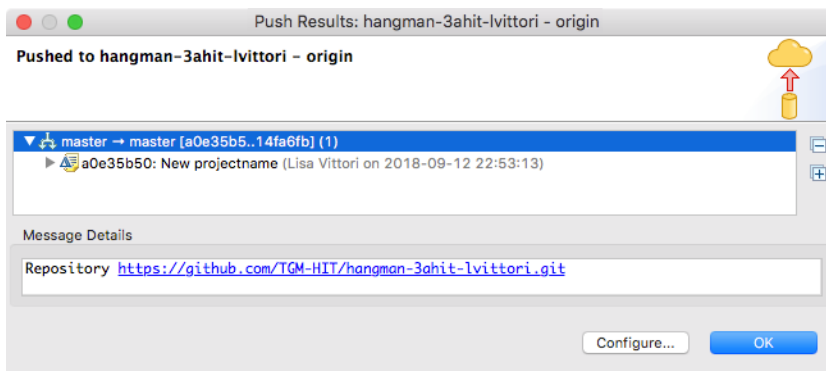
Dadurch öffnet sich die "Git Staging" Ansicht



*Einen "Versionspunkt" erstellen und ihn auch gleich am Remote speichern*

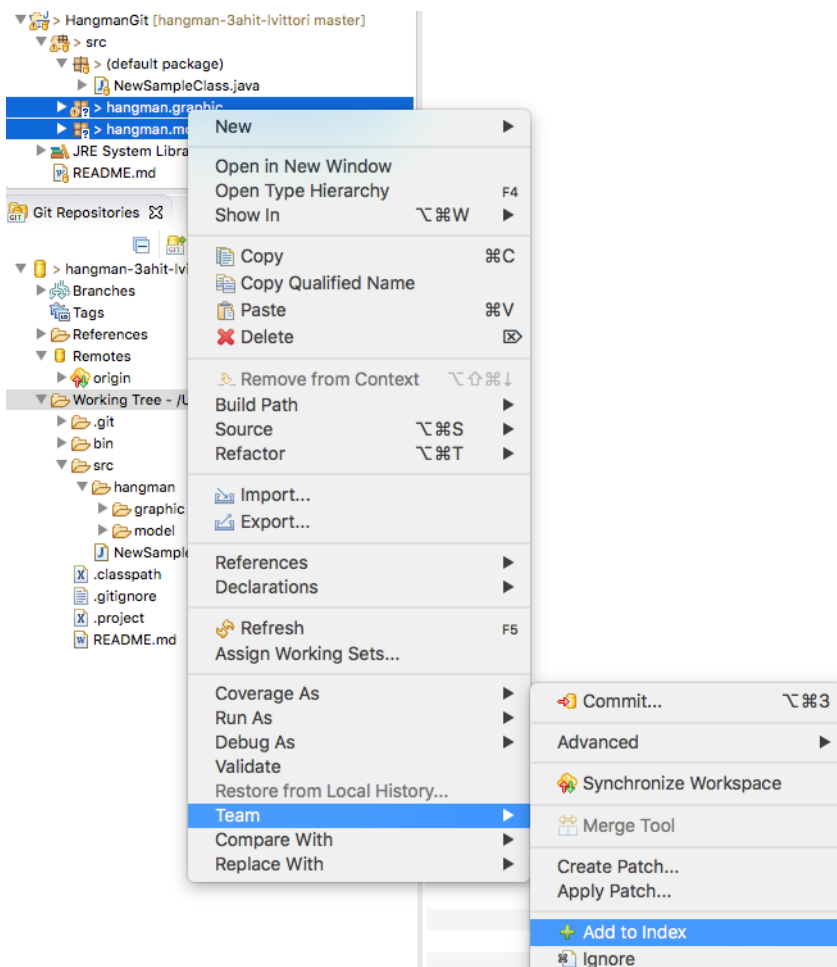
*Nur einen lokalen "Versionspunkt" erstellen*

In dieser kann – nachdem eine sinnvolle Beschreibung der Änderung als "Commit message" eingetragen wurde – der neue Projektname mittels "Commit and Push..." auf Github übertragen werden. Nach Eingabe des Benutzernamens und des Passworts wird die Übertragung durch eine Erfolgsmeldung bestätigt

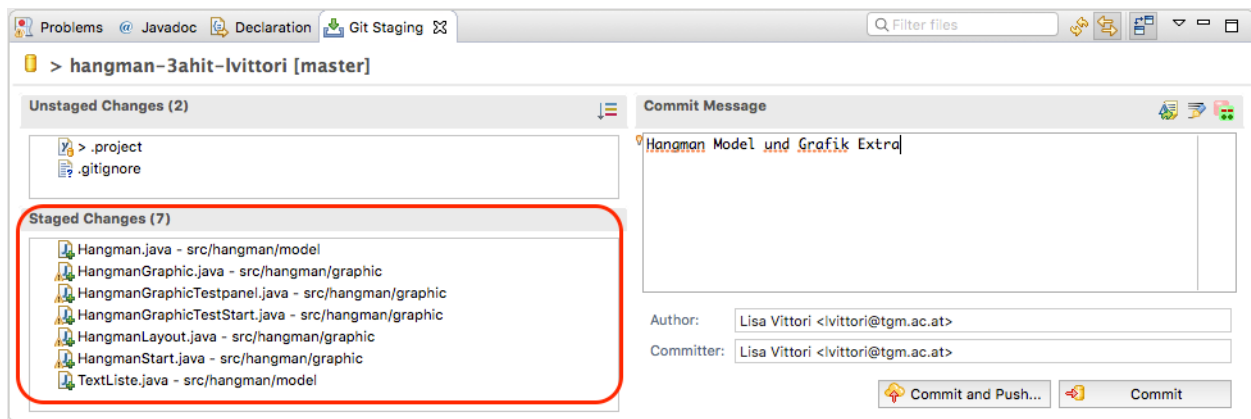


### Neue Dateien zum Index bzw. Staging Bereich hinzufügen

Damit man mit dem Projekt arbeiten kann muss man neue Dateien hinzufügen bzw. bestehende Klassen ändern. Diese werden nicht automatisch zum Staging Bereich hinzugefügt. Man muss sie vor dem Commit in den Staging Bereich mittels "Add to Index" verschieben.



Erst wenn die neuen Dateien im Bereich "Staged Changes" aufgelistet sind, werden sie bei einem "Commit" auch als Versionspunkte gespeichert bzw. bei einem "Commit and Push ..." an Github übertragen.



### Habe ich es verstanden?

- Ich kenne die Begriffe Versionsverwaltungssystem bzw. Verteiltes Versionsverwaltungssystem und die zugehörigen Abkürzungen VCS bzw. DVCS
- Ich kenne die Github-Plattform und kann mittels Github Classroom dort Repositories anlegen.
- Ich kenne die folgenden Begriffe und kann sie erklären
  - Working tree, working directory, working copy
  - Repository
  - Index oder staging area
  - Commit
  - HEAD
  - Remote (repository)
- Ich kann erklären, wie der Ablauf bei der Versionierung einer Datei mit git aussieht.
- Ich kann ein repository von Github mit Hilfe von Eclipse auf meinen lokalen PC klonen
- Ich kann Versionspunkte erstellen
- Ich kann Änderungen auf das Repository auf Github übertragen
- Ich kann neue Dateien zum Repository auf Github hinzufügen.