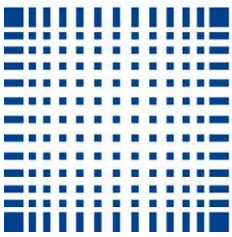

hochschule mannheim

Git - Anleitung

Eine Einarbeitung in den Umgang mit Git

von Andreas Roth

Inhaltsverzeichnis

Einführung in git

Was ist Git?

Git ist ein weit verbreitetes, verteiltes Versionskontrollsystem, das im Jahr 2005 von Linus Torvalds, einem finnischen US-amerikanischen Informatiker, entwickelt wurde.
Es dient vor allem den Entwicklern dazu, Änderungen an ihrem Code effizient zu verfolgen und zu verwalten.

Versionskontrolle

Hierzu ist die **Versionskontrolle** ein grundlegendes Konzept.
Git speichert alle Versionen eines Projektes und ermöglicht es, zwischen verschiedenen Versionen zu wechseln. Hiermit kann man besonders gut Änderungen verfolgen und in der Version zurückgehen, sollten Probleme auftreten.

Verteilte Entwicklung

Ein weiteres Konzept, welches von Git unterstützt wird, ist die **Verteilte Entwicklung.** Sie ermöglicht nicht nur, dass man als Nutzer von mehreren Geräten auf den Quellcode zugreifen kann, sondern es soll auch die Zusammenarbeit mit anderen erleichtern, da jeder Entwickler Kopien des gesamten Projektverlaufs auf seinem Rechner hat. Dies erleichtert die Zusammenarbeit in Projekten, auch wenn keine dauerhafte Internetverbindung besteht.

Branching und Merging

Durch das Erzeugen von **Branches** kann man den Quellcode, an welchem man gerade arbeitet, vom Hauptcode im Haupt Branch abkapseln und somit außerhalb programmieren, ohne dass das funktionierende Produkt bei einem Fehler beschädigt wird.
Wenn man mit seinem Teilabschnitt fertig ist und dieser geprüft wurde, kann man den dafür erzeugten Branch mit dem Haupt Branch **mergen**, das heißt, die Branches werden zusammengeführt und der neue Quellcode Abschnitt befindet sich im Anschluss mit auf dem Haupt Branch.

Funktionen von Git

Git-Repository

Um seine Daten im Git verwalten zu können, muss man als erstes ein **Repository** erstellen. Ein Git-Repository ist ein Ort, an dem sowohl das Projekt, als auch alle seine Versionen gespeichert sind. Der Vorteil hierbei ist, dass diese lokal auf dem eigenen Computer, als auch auf einem Remote-Server gehostet werden.

Pull-Request

Ein **Pull-Request** ist eine Anfrage, um die Daten auf dem lokalen Gerät mit dem Server zu synchronisieren. Hierbei werden die Neuerungen vom Server heruntergeladen.
Man muss darauf achten, dass man einen Pull-Request ausführt, bevor man mit der Arbeit auf dem lokalen Gerät anfängt. Ansonsten stimmt die Version auf dem Server nicht mehr mit der lokalen Version überein und es kommt zu einem Merge-Konflikt.

Commit

Eine **Commit-message** ist eine Nachricht, welche bei einem Push zu der Datei hinzugefügt wird. Diese Nachricht soll kurz und knapp beschreiben, was seit dem letzten Pull-Request geändert wurde.

Push

Sobald man etwas an seiner Datei geändert hat und die Arbeit daran abgeschlossen wurde, kann man einen **Push** ausführen. Hierbei wird die geänderte Datei auf den Server geladen und die Version wird mit der aktuellen abgeglichen. Ein Push ist im Regelfall nicht möglich, wenn eine Commit-message fehlt. Auch kommt es spätestens hier zu Problemen, wenn die lokale Version nicht mit der Version auf dem Server übereinstimmt.

Wo kann ich ein Repository erstellen?

Orte, an denen man ein Repository erstellen kann

Es gibt einige Orte, an denen man ein Repository erstellen kann. Im Regelfall nutzt man hierfür GitHub: https://github.com/
In unserem Falle steht auch ein Git der Hochschule zur Verfügung: https://gitty.informatik.hs-mannheim.de/
Im Falle von GitHub kann man sich einen freien Account anlegen und diesen nutzen. Bei unserem Hochschuleigenen Server kann man sich mit seiner hs-mail (matr.Nr.@stud.hs-mannheim.de) und dem dazugehörigen Passwort anmelden. Auch dieses Angebot kann frei genutzt werden.

Repository im Browser erstellen

GitHub

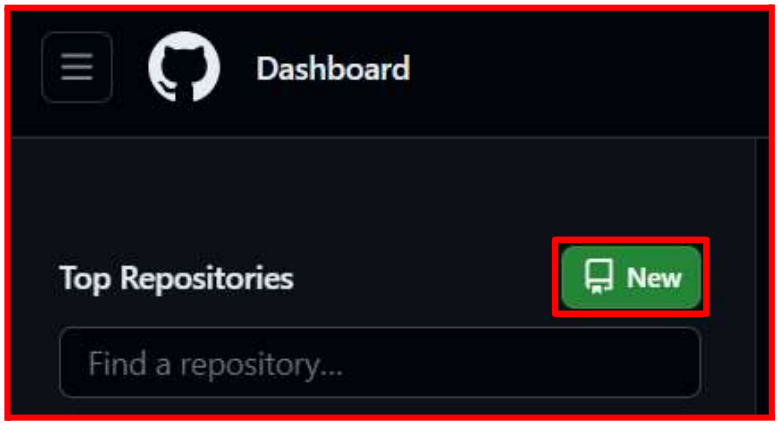

abb 1: neues repository auf GitHub erstellen

Im linken oberen Feld kann man einen grünen Button finden, der “new” sagt. Sollte man diesen antippen, kann man die Einstellungen für das neue Repository aufrufen.
Hier sucht man an der ersten Stelle den Owner, also den Admin heraus. Anschließend kann man dem Repository einen neuen Namen geben, in unserem Fall nennen wir es “neues_repo”.
Eine grundlegende Beschreibung, was sich in diesem Repository befindet, können wir auch angeben, müssen dies aber nicht. Anschließend wählt man aus, ob das Repository öffentlich oder privat zugänglich sein soll. Sollte man privat wählen, können trotzdem alle, die zur Kollaboration auf das Repository eingeladen werden, darauf zugreifen.
Die anschließenden Schritte sind optional auswählbar. Man kann hier eine README-Datei erstellen, in welcher häufig beschrieben wird, wie das Programm in diesem Repository installiert und genutzt wird. Für unseren Fall benötigen wir es fürs Erste nicht. Eine README-Datei kann man jederzeit ergänzen.
Man hat auch die Möglichkeit, eine .gitignore-Datei zu generieren, in welche Dateinamen geschrieben werden können, die bei einem Push-Befehl nicht berücksichtigt werden sollen.

Will man z.B. die Datei bild.jpg nie mit pushen, so schreibt man “bild.jpg” in die .gitignore. Will man hingegen alle Bilddateien bei einem push ignorieren, so schreibt man generell “*.jpg” in die .gitignore-Datei. Oft wird diese genutzt, um lokale .log Dateien nicht mit auf das Repository zu pushen. Auch .gitignore benötigen wir für den Anfang nicht. Anschließend kann man sich noch entscheiden, ob man sein Produkt lizenzieren will und welcher Lizenz dieses angehören soll. Auch dieser Teil interessiert uns in diesem Fall nicht.

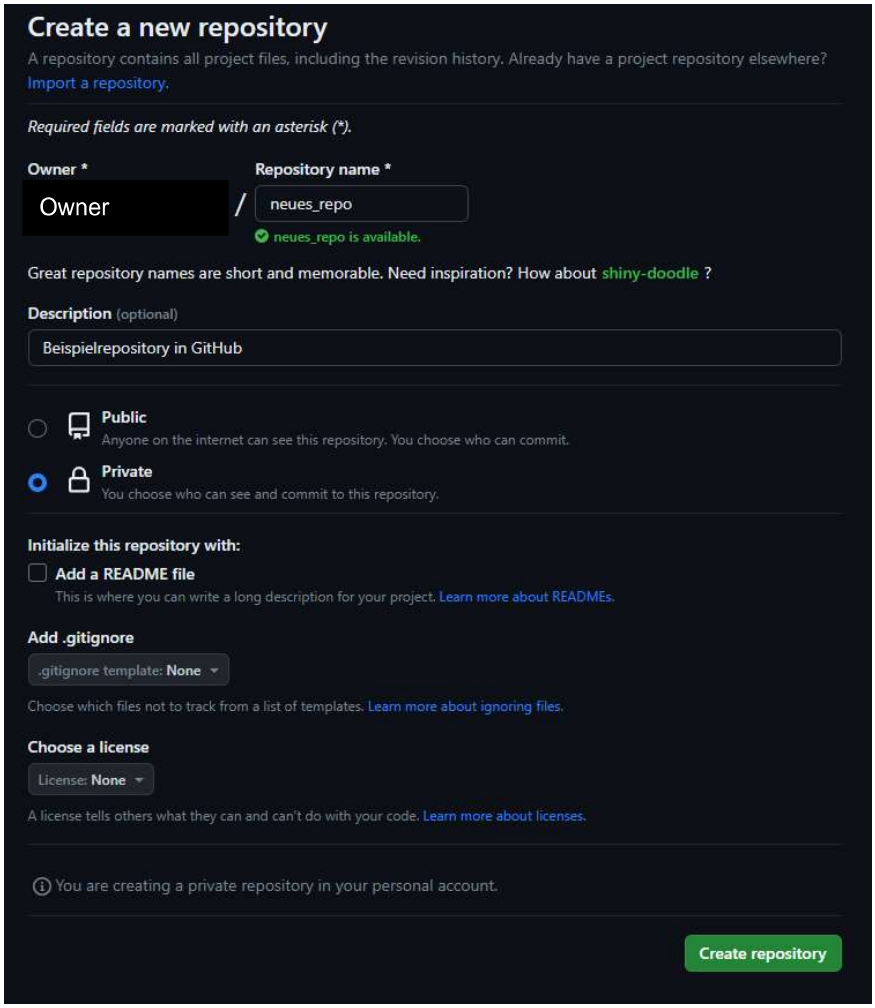

abb 2: einstellungen für neues Repository auf GitHub erstellen

Hiermit ist die Erstellung des Repositories abgeschlossen und man kann loslegen. In der rechten Ecke kann man nun mitwirkende einladen, darunter sieht man die URL, durch welche das Repository aufgerufen werden kann. Diese URL ist immer wie folgt aufgebaut:

https://github.com/NameErsteller/repositoryName.git

Mit dieser URL kann man nun seinen PC verbinden und loslegen.

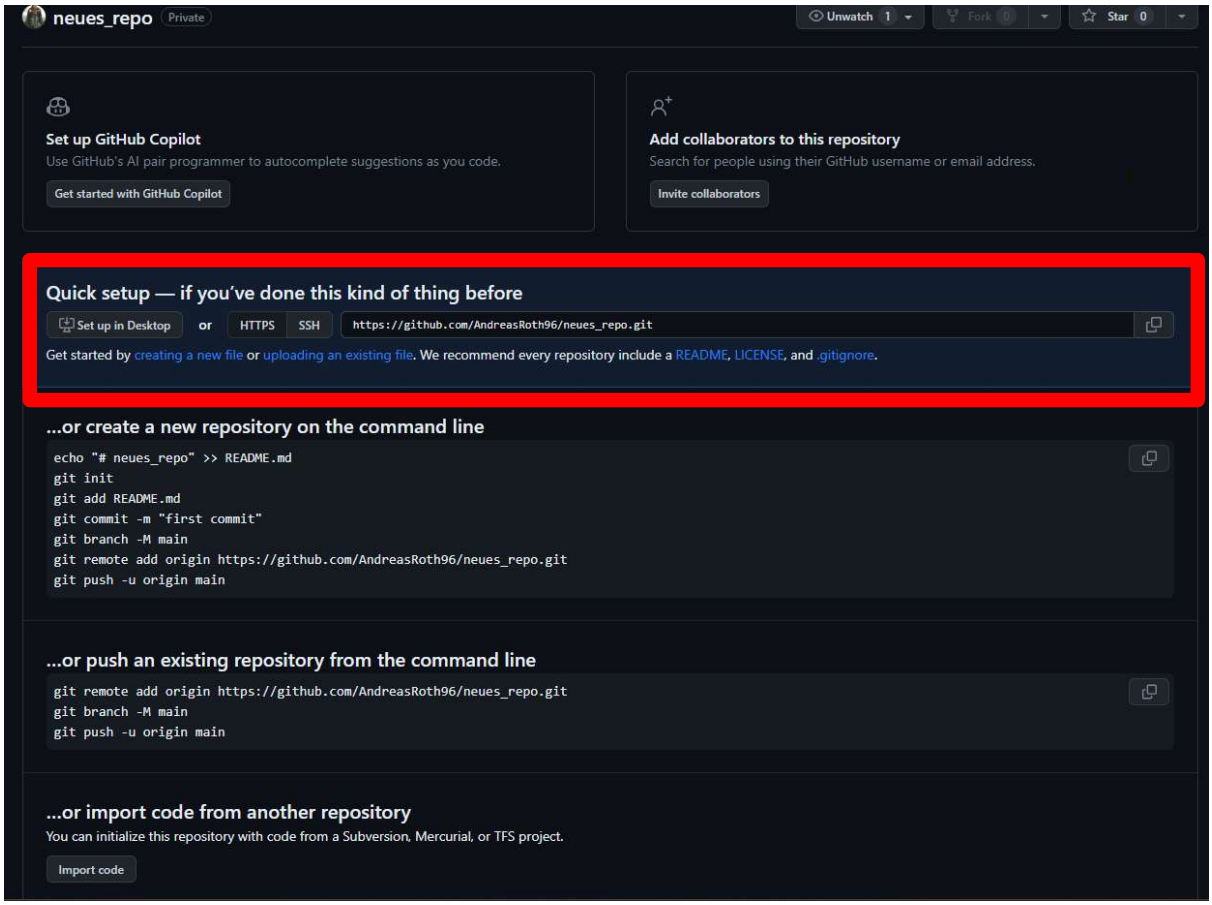

abb3: eigenes GitHub Repository URL

Sollte man das Repository in Zukunft im Browser suchen, wird es auf der linken Seite unterhalb unseres “new”-Buttons erscheinen. Wie man es mit dem eigenen PC verbindet, wird später näher erläutert. Man kann auch bereits im Browser anfangen, das Repository zu beschreiben. Hierzu kann man “creating a new file” im Satz “Get started by creating a new file or uploading an existing file.” auswählen.
Um den Quellcode im Anschluss nutzen zu können, muss man ihm erst einmal einen Namen geben. In unserem Fall heißt er “Hello”.

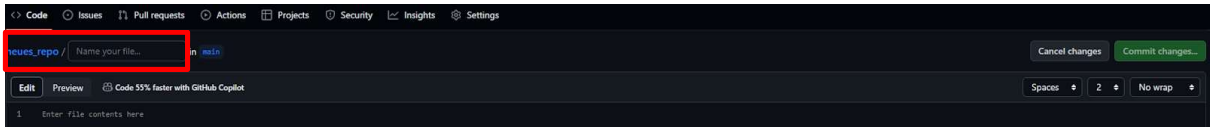

abb4: Name eines Quellcodes im Browser

Anschließend kann man mit dem Schreiben des Quellcodes beginnen:

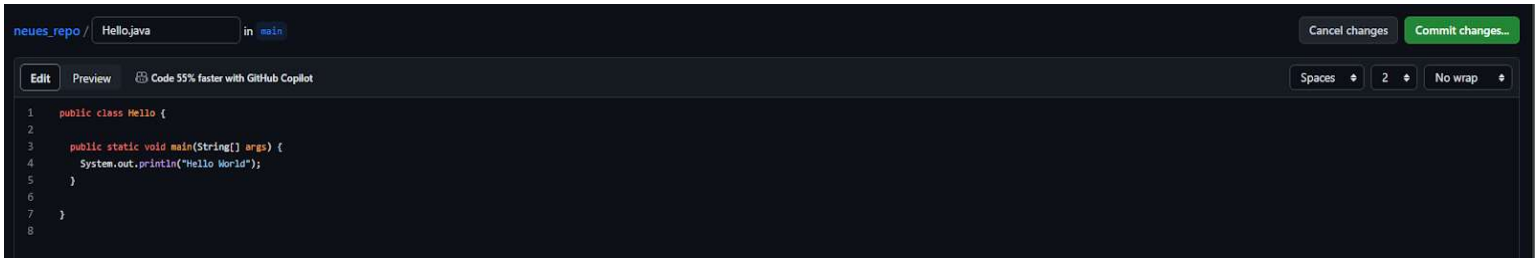

abb5: Quellcode direkt in GitHub

Sobald man auf “Commit Changes” im oberen rechten Bereich tippt, kann man in einem aufpoppenden Fenster die Commit-Message eingeben:

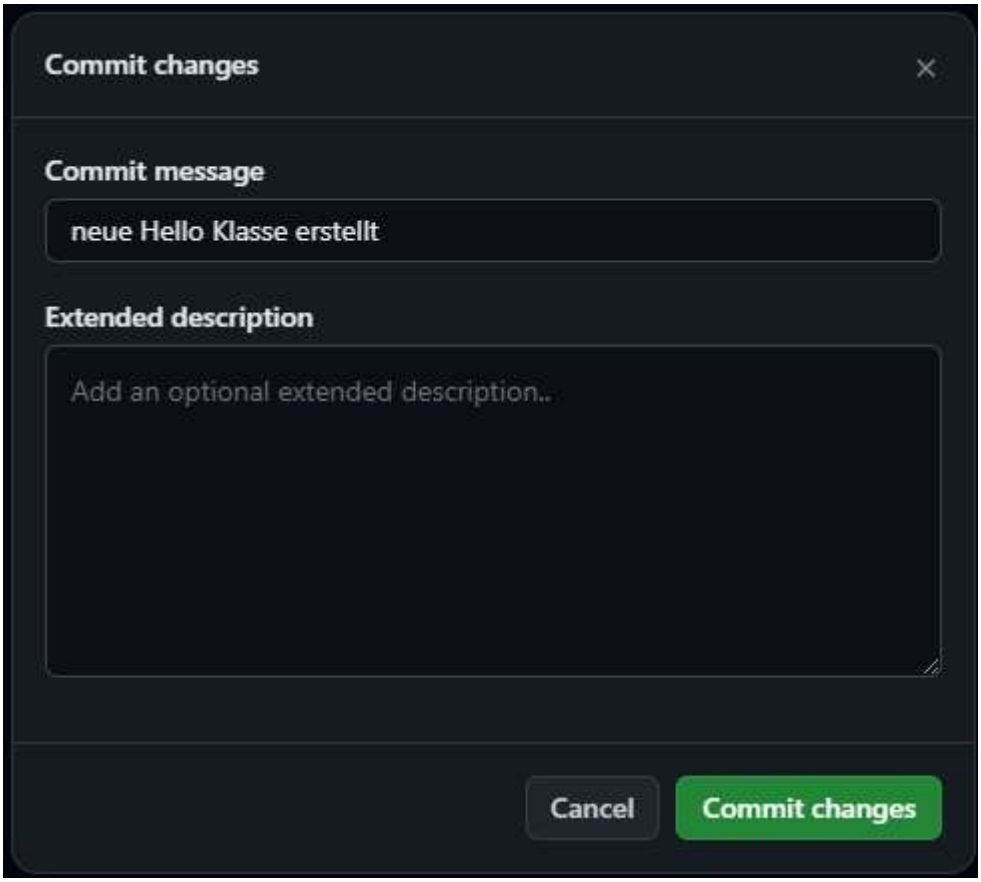

abb 6: Commit-Message im Browser

Sobald man eine sinnvolle Erklärung eingegeben hat, was man an dieser Datei geändert hat, kann man mit dem Button “Commit changes” die Änderungen zum Server ergänzen. Im neuen Repository müsste jetzt die Datei “Hello.java” zu sehen sein. Man kann hier entnehmen, wie viele Commits getätigt wurden, als auch die Zeit des letzten Commits, mitsamt der eingegebenen Nachricht entnehmen.

abb 7: Klasse, sichtbar im GitHub-Repository

Gitty

Bei Gitty ist die Funktion ähnlich zu GitHub. Lediglich in der Darstellung unterscheiden sich die beiden Plattformen. In Gitty wird der Repository Pool nicht auf der linken, sondern auf der rechten Seite angezeigt. Hier kann man sein Repository mit dem angezeigten + erstellen.

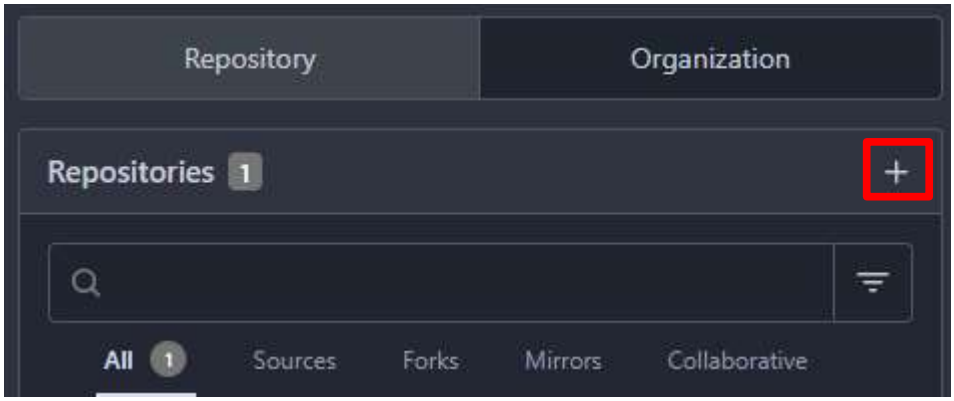

abb8: Repository in Gitty erstellen

Das Auswahlfenster in Gitty sieht etwas umständlicher aus, bietet aber genau dieselben Einstellungen wie GitHub. Oben kann man den Besitzer, in diesem Falle seine Matrikelnummer und den Namen des Repositorys eintragen. In unserem Fall nennen wir es gleich wie bei GitHub, “neues_repo”.
Mit dem Häkchen darunter kann man das Repository privat als privat deklarieren. Auch in das Beschreibungsfeld schreiben wir dieselbe Erklärung, sodass direkt bekannt ist, worum es sich bei dem Repository handelt.
Die weiteren Punkte können wir, genau wie bei GitHub, ignorieren.
Der Default Branch sollte auf Main bleiben, da der fertige Quellcode immer im Main gemerged wird. Zum Abschluss kann man entscheiden, ob man sein Repository zu einem Template umwandeln möchte, was sich dann rentiert, wenn man weitere Repositories mit denselben Einstellungen erstellen möchte.

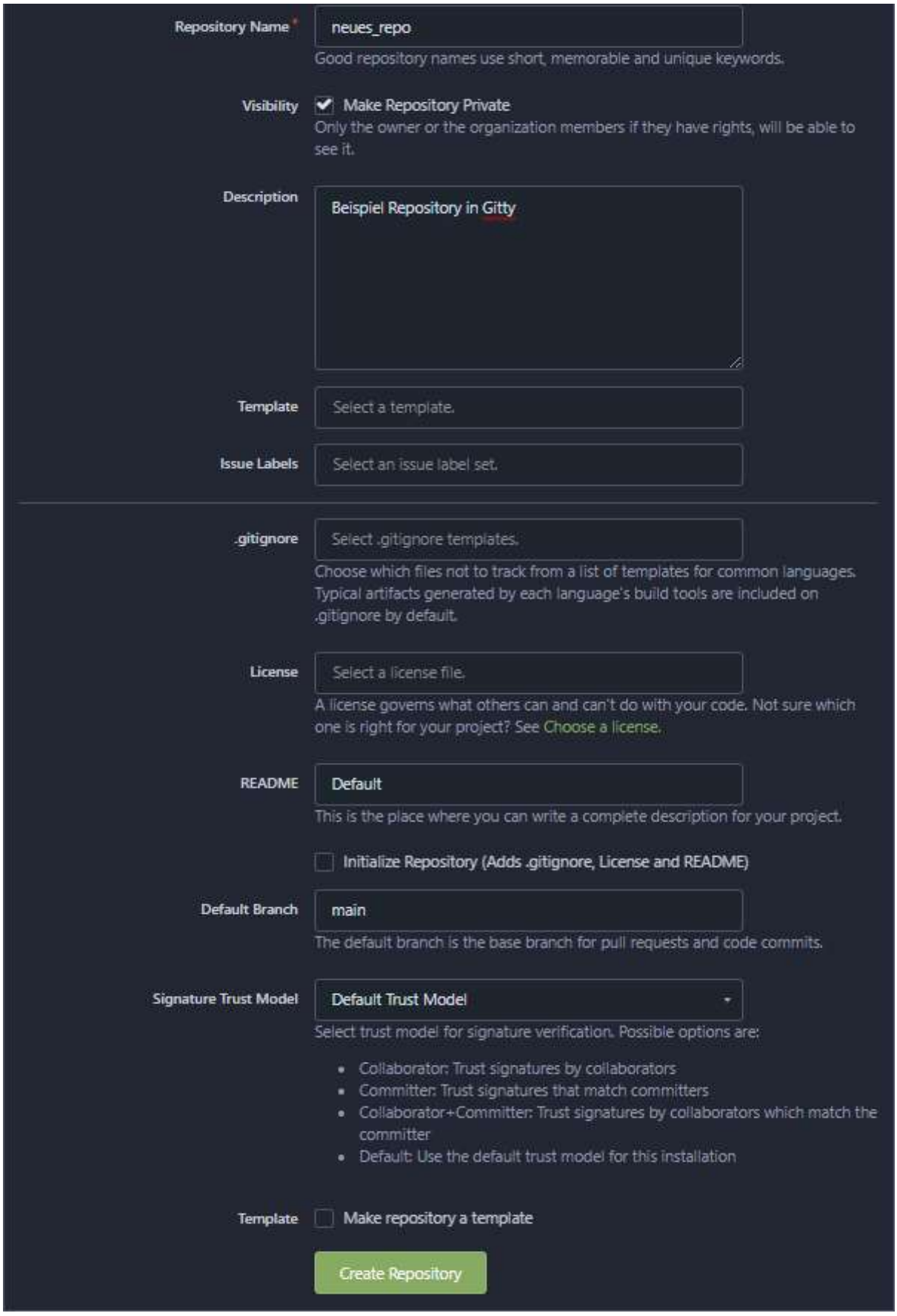

abb9: Repository Einstellungen in Gitty

Mit Bestätigung auf “Create Repository” erstellen wir ein neues Repository.
Auch die nachfolgende Seite zeigt uns ein ähnliches Bild, wie in GitHub. Auch hier haben wir einen Link, der unser lokales Gerät mit dem Server verbinden kann. Links davon haben wir einen Button mit dem Namen “New File”, mit welchem wir eine Datei direkt im Browser erstellen können.
Möchten wir nun Entwickler zum Kollaborieren einladen, so müssen wir die Einstellungen über “Settings” im rechten oberen Fenster aufrufen.

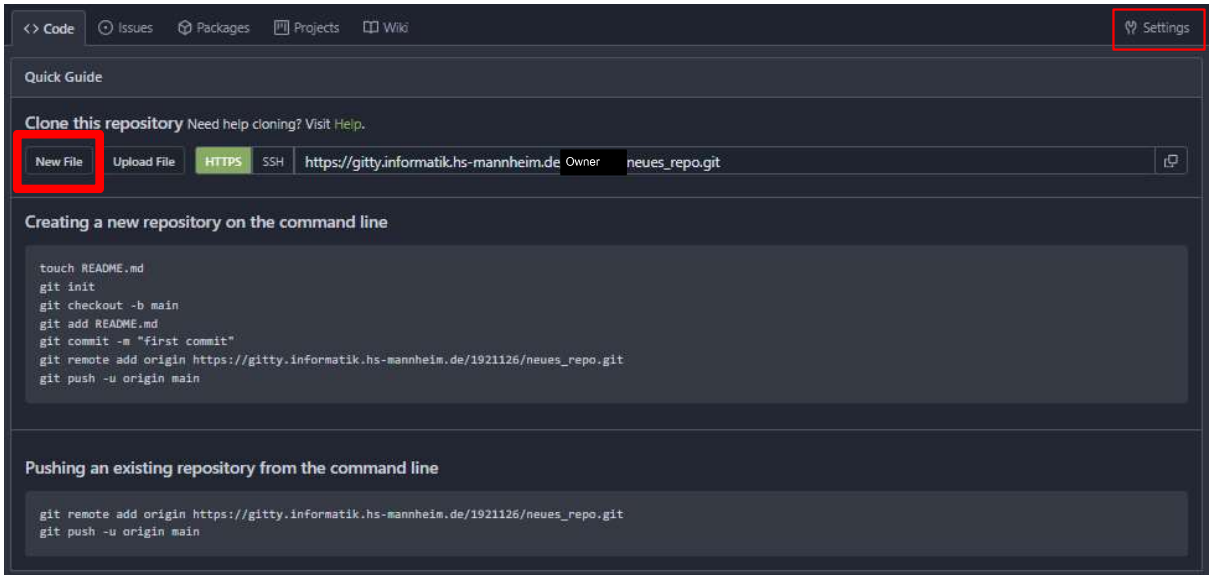

abb10: fertiges Repository in Gitty

Wenn man nun in den Einstellungen ist, kann man unter dem Reiter Collaborators neue User hinzufügen. Dies kann man ganz einfach über den Namen der Person oder über die Matrikelnummer vornehmen.

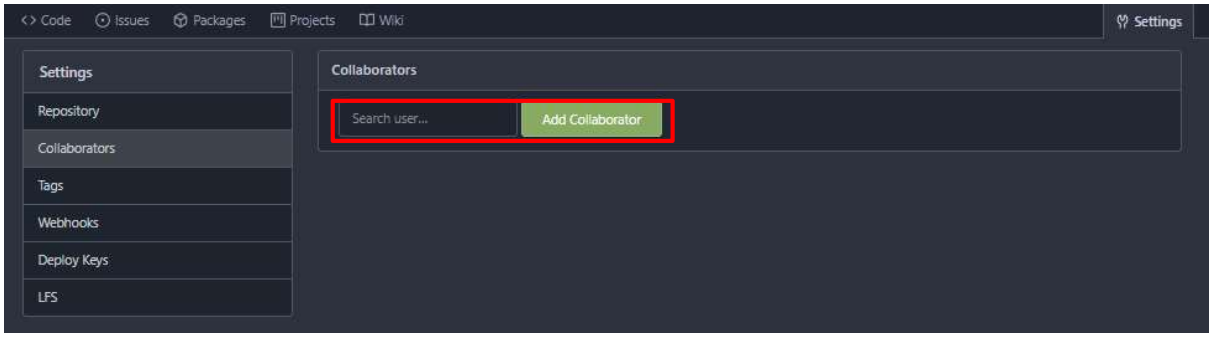

abb 11: Neuen Nutzer hinzufügen

Beim Erstellen einer Datei im Browser geht man ähnlich vor wie bei GitHub. Oben links muss man einen Namen für die Klasse eingeben, unten kann man anschließend den Quellcode schreiben.
Wenn man nun committen will, muss man nach unten scrollen und den Namen der Klasse, als auch die Commit-Message eingeben.

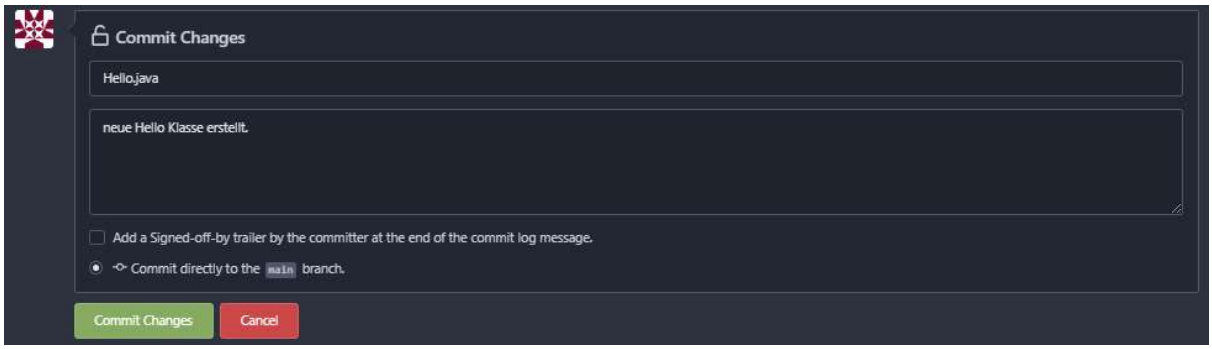

abb 12: commit im gitty-Browser

Anschließend kann man den Button “Commit Changes” betätigen und die neue Datei pushen.

Repository lokal erstellen

Um lokal mit Git zu arbeiten, gibt es viele verschiedene Möglichkeiten. Es gibt die git Konsole, über welche man mit git kommunizieren kann: https://git-scm.com/downloads
Man kann auch über ein grafisches User Interface arbeiten, sollte man sich mit Konsolenbefehlen nicht wohl fühlen: https://desktop.github.com/
Diese Anwendung wird direkt von GitHub angeboten.
Darüber hinaus bietet so gut wie jede Entwicklungsumgebung einen eigenen Weg, mit GitHub zu kommunizieren, unter anderem auch Oracles Eclipse mit eGit.
Der Einfachheit halber werden wir im Folgenden lediglich über die Konsolenbefehle und über Eclipse reden. Des Weiteren gehen wir nur auf die Verbindung mit GitHub ein, nicht auf die mit Gitty, da die Kommunikation mit beiden Servern gleich verläuft.

Repository in der Konsole erstellen

Als erstes bewegt man sich in der Konsole an den Ort, welchen man mit einem Repository verknüpfen will. Hierzu gibt man den Befehl “cd”, gefolgt von dem Pfad des Ordners ein.
Als Beispiel kann ich mein Verzeichnis zeigen:

```
cd C:/Users/Roth/PR1_Beispiel
```

Hierbei muss man beachten, dass man nicht Windows-typisch das Backslash(\) verwendet, sondern mit Slash(/) arbeitet.
Bevor man jetzt loslegt, sollte man sich mit seinem GitHub-Account verbinden. Dies nimmt man wie folgt vor:

```
git config --global user.name “Dein Name”
git config --global user.email “Deine-email@example.com”
```

Hierbei muss man beachten, dass es sich vor “global” nicht um einen, sondern zwei Striche handelt.
Anschließend gibt man den Befehl:

```
git init
```

an. Dieser Befehl erstellt ein .git Unterverzeichnis, welches alle relevanten Daten für Git enthält.
Zu guter Letzt muss man ein Repository in GitHub mit dem lokalen Repository verbinden. Hierzu gibt man den Befehl:

```
git remote add origin https://github.com/linkZuRepository
```

an.
Hiermit kann man loslegen und alle Dateien innerhalb dieses Ordners in die Cloud pushen. Git kann lediglich auf diesen Ordner zugreifen, weswegen alle Dateien, welche gepusht werden sollten, auch innerhalb dieses Ordners liegen müssen.

Repository in Eclipse erstellen

In Eclipse kann man mit Rechtsklick auf sein Projekt gehen und den Reiter “Team” auswählen. In der nun angezeigten Liste kann man die Option “Share Project…” wählen. Hier wird nun automatisch ein Repository erstellt.

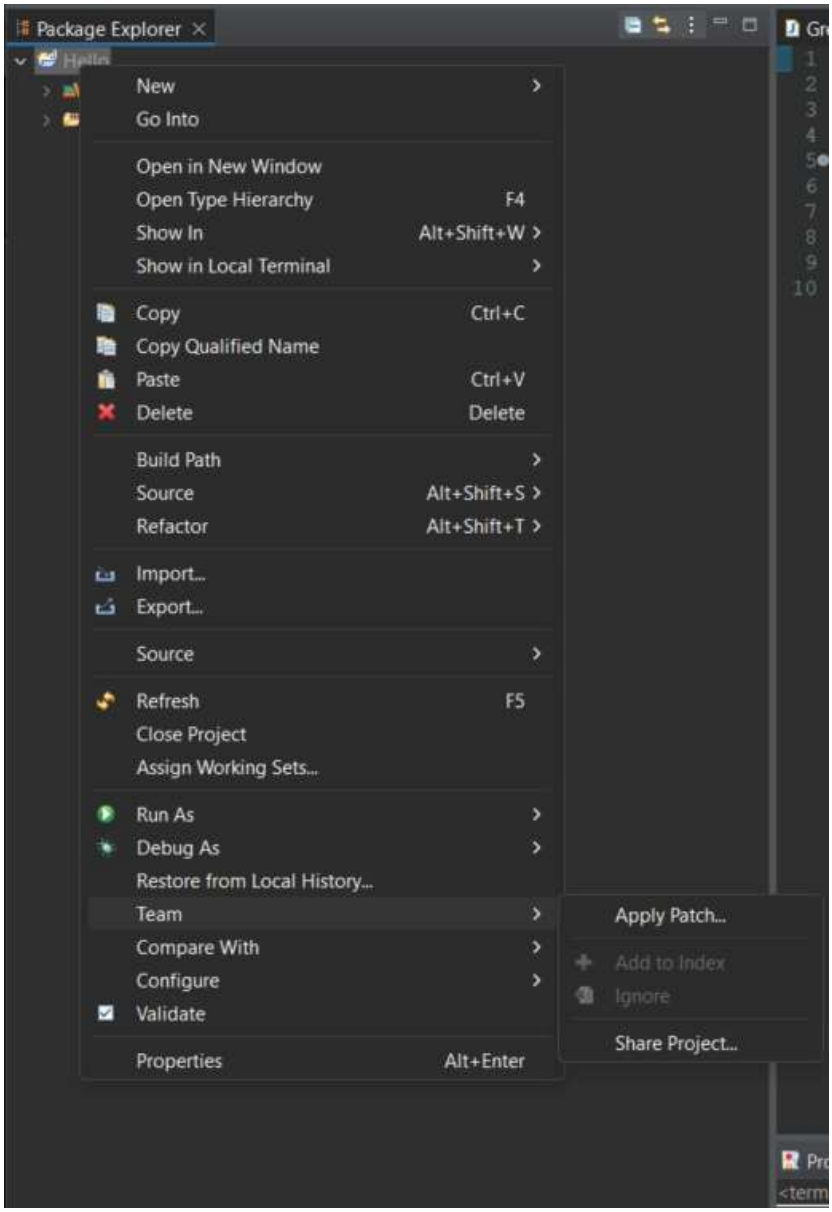

abb 13: Repository in Eclipse erstellen

In dem anschließenden Fenster wählt man den Pfad aus, wo sich das Projekt befindet, welches man teilen möchte. Nach der Bestätigung durch den Button “finish” ist alles für die Arbeit vorbereitet.
Sollte man anschließend noch einmal auf den Reiter “Team” gehen, kann man alle Möglichkeiten sehen, die man mit Git hat.
Bevor man etwas machen kann, muss man noch den Pfad des Repositories einfügen, als auch seine Anmeldedaten.
Um sich bei Eclipse anmelden zu können, muss man einen Access Token generieren. Diesen setzt man anschließend als Passwort ein.

Wenn man rechts oben auf sein Profil klickt, erscheint ein Menu, in dem man unterhalb “Settings” aufrufen kann.

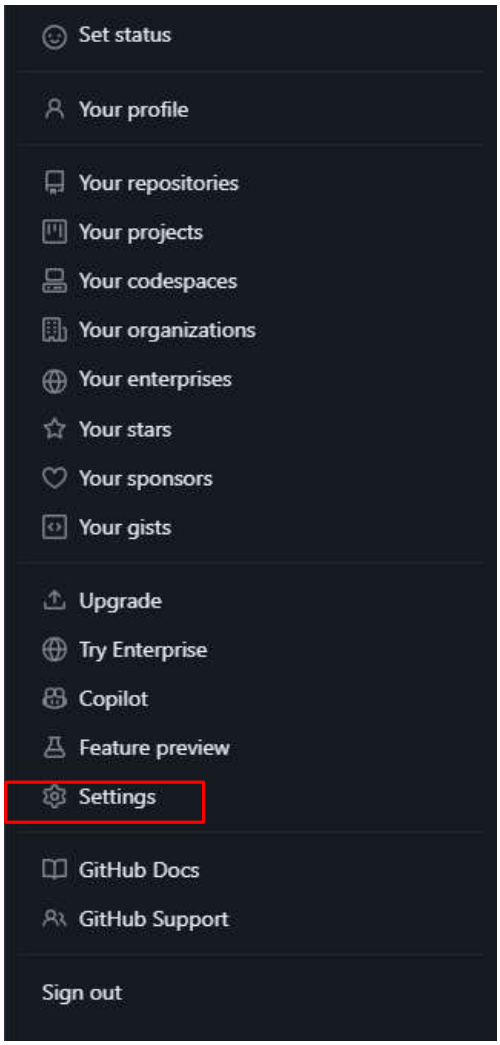

abb 14: Github Einstellungen

Hier springt man auf den letzten Punkt, “Developer settings” ab.

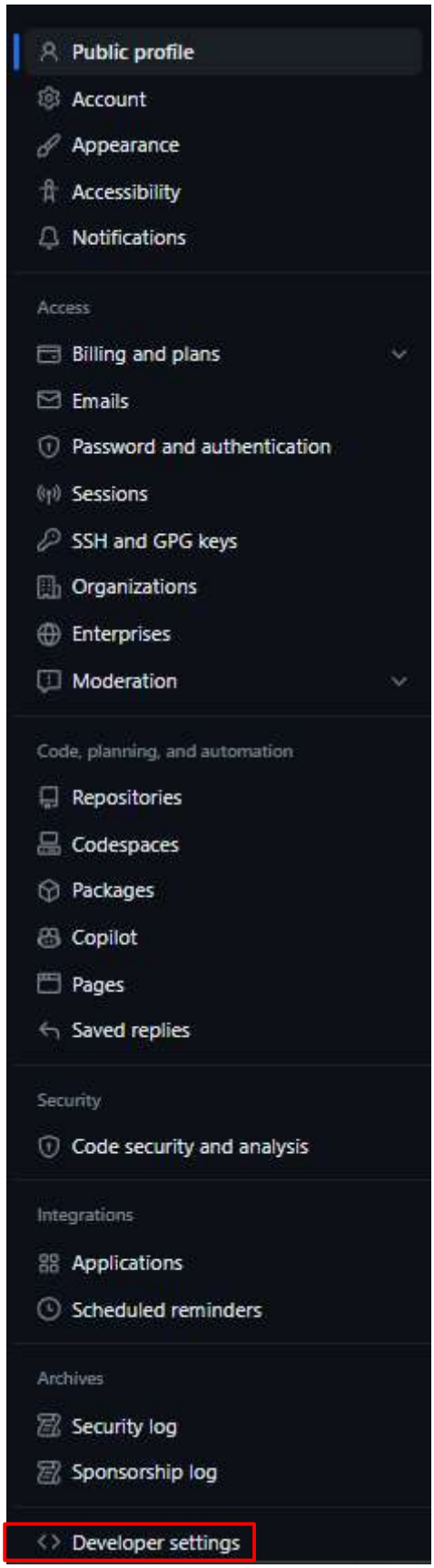

abb 15: GitHub Developer Settings

Hier kann man unter “Personal Access Tokens" -> “Tokens (classic) einen neuen Token generieren.

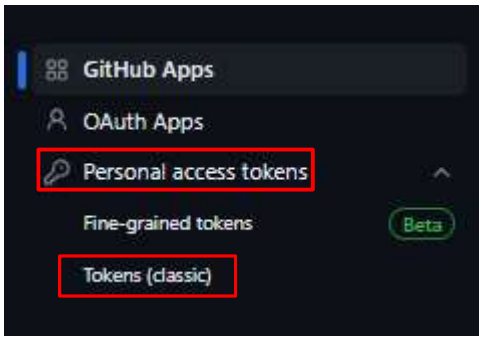

abb 16: Generate Access Token

Anschließend kann man unter dem Reiter “Generate new Token” “Generate new Token (classic)” einen neuen Token generieren.

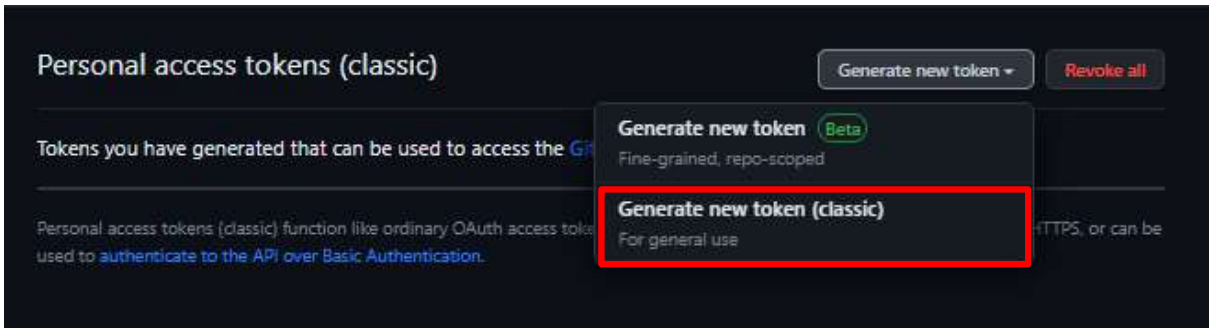

abb 17: Generate Access Token

Bei dem Token muss man angeben, wofür dieser Token genutzt wird. Wir können in unserem Fall etwas wie “Verbindung” wählen.
Anschließend wählen wir das Kästchen repo aus und bestätigen mit “Generate token”.

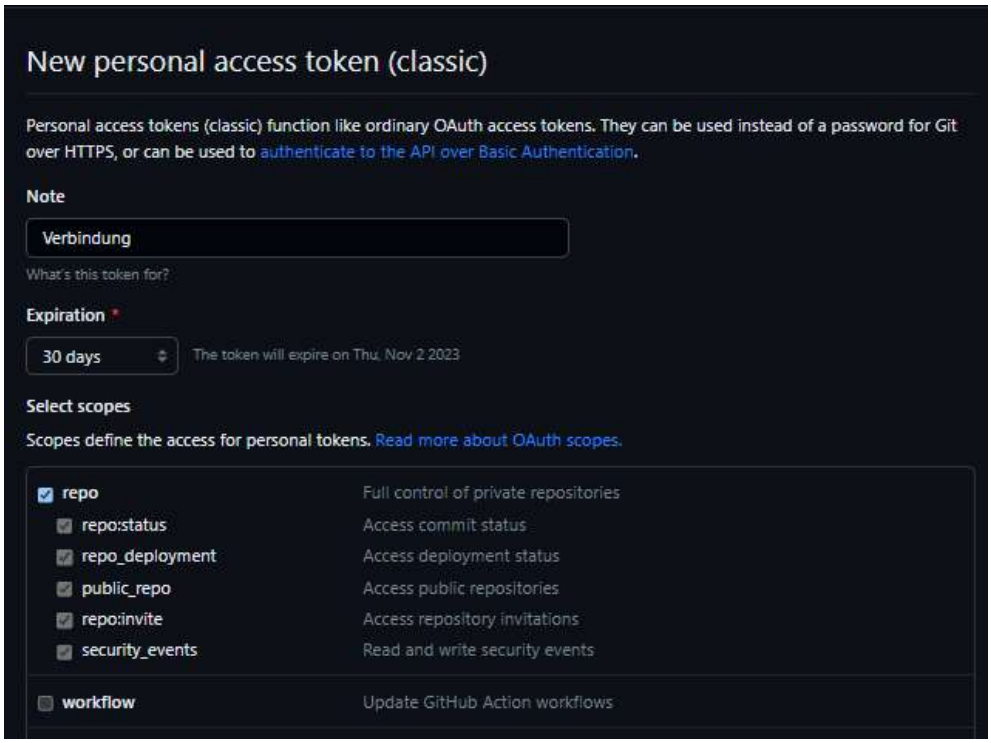

abb 18: Token erstellen

Den anschließend angezeigten Token kann man nun kopieren und als Passwort in Eclipse eingeben. Hierbei ist zu beachten, dass der Token trotz angegebener Frist nur einmal nutzbar ist.
Wenn man nun bei seinem Eclipse Projekt das erste Mal versucht zu pullen, muss man sich mit Name und Passwort anmelden. Wie bereits oben erwähnt, gibt man seinen Namen an, anschließend gibt man den Access Token ein, nicht das Account Passwort.

Repository klonen

Was macht man, wenn man bereits einige Dateien hat und genau dieses Repository auch auf seinem Rechner haben will?
Dafür erstellt man nicht mehr ein neues Repository, sondern kann das bereits existierende klonen.
Um ein Repository klonen zu können, müssen wir die URL des Repositorys kopieren.
Dafür hat man allerdings nicht den Link, der im Browser angegeben ist, zur Verfügung, sondern muss einen separaten Link anfordern.
Diesen erhält man, wenn man in das Repository geht und den Reiter “Code” ausfährt.
Hier entscheiden wir uns im Regelfall entweder für den HTTPS-Reiter oder den Reiter namens SSL. Für uns reicht der Link, der unter https angegeben ist. Diesen kopieren wir.

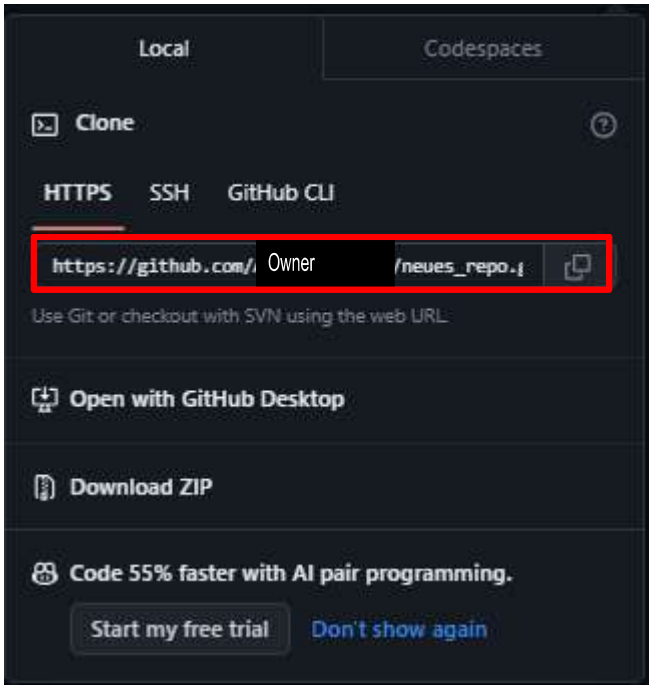

abb 19: Repository Link kopieren

Repository in der Git Bash klonen

Mit dem Kopierten Link können wir das Repository schnell klonen. Wir geben hierzu folgenden Befehl in der Bash ein:

git clone https://github.com/kopierterLink

Anschließend befolgen wir die Anmeldeschritte, die angegeben werden. Im Regelfall reicht es, im Browser angemeldet zu sein und dies zu bestätigen.

Repository in Eclipse klonen

Auch in Eclipse ist das Klonen sehr einfach. Mit Rechtsklick im Package Explorer kann man “Import…” auswählen.

abb 20: Importieren in Eclipse

Anschließend wählt man Git und Projects from Git aus und tippt auf weiter.

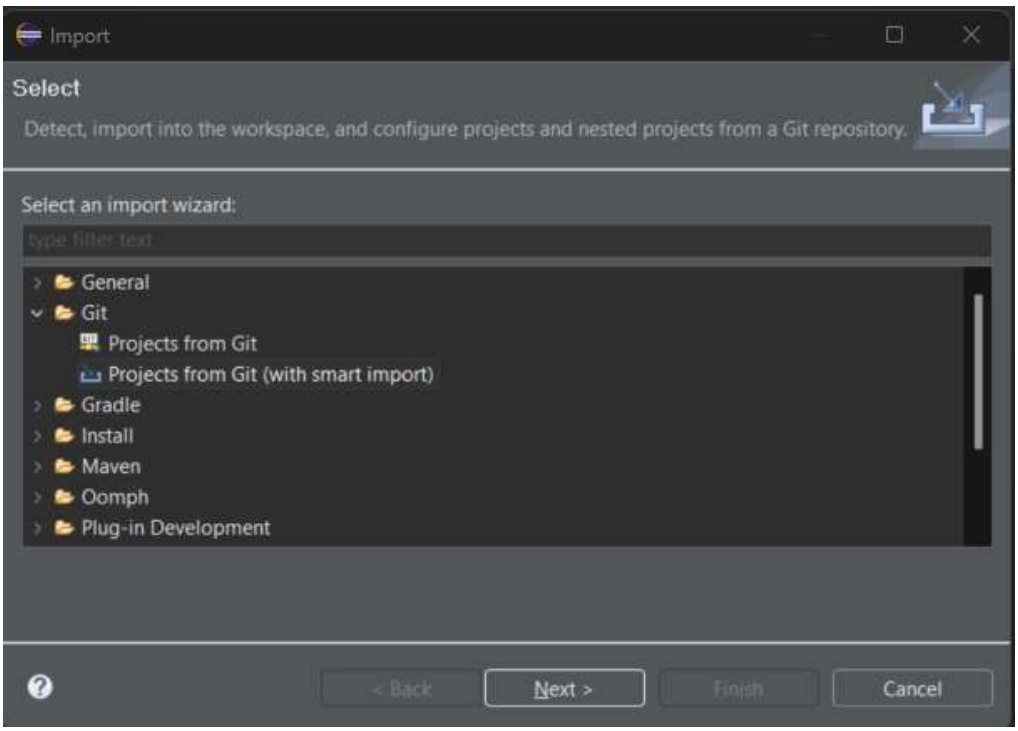

Abb 21: Git für den Import auswählen

Im anschließenden Fenster wählt man Clone URI aus und tippt abermals auf weiter.
Im Feld URI wird nun die in GitHub kopierte URL eingetragen. Im Reiter Authentication trägt man anschließend wieder seinen Namen und den generierten Token ein.
Anschließend müsste ein neues Projekt im Package Explorer erscheinen, welches den Namen des Repositorys trägt.

Operationen in Git

Pull

Pull in der Git Bash

Wenn die Bash mit dem Server verbunden ist, reicht es, den Befehl:

```
git pull
```

zu verwenden und der lokale Speicher synchronisiert sich mit dem Server.

Pull in Eclipse

Man springt wieder über “Team” ab und wählt “Pull” aus.

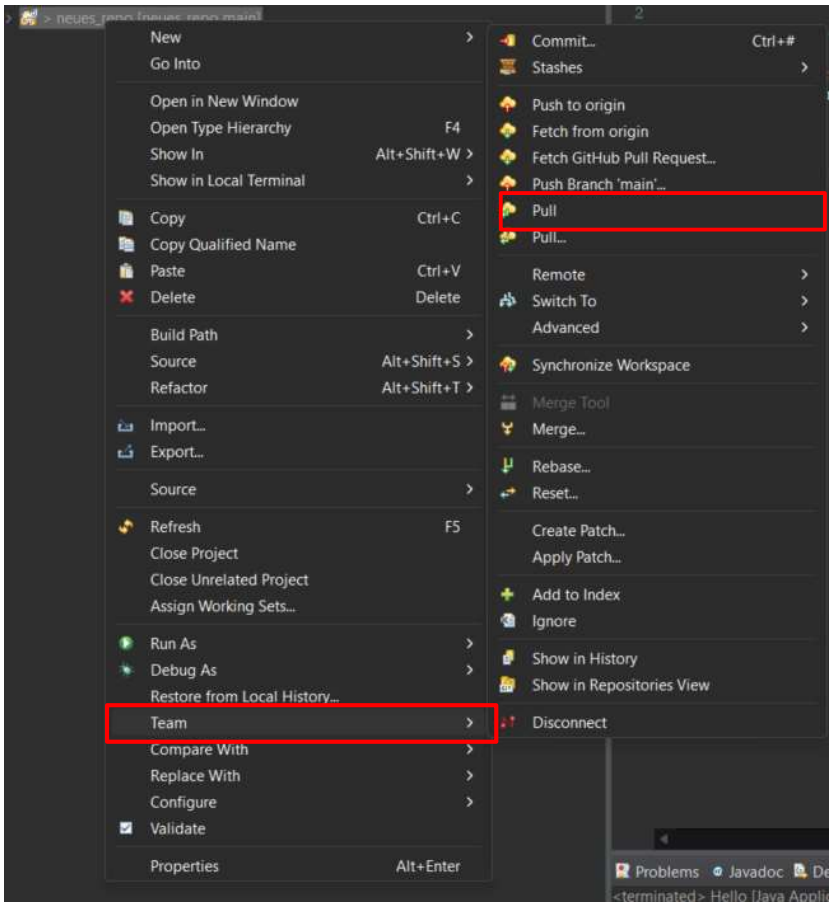

Abb 22: Pull in Eclipse

Anschließend wurde das Projekt, auf welches der Pull-Request ausgeführt wurde, um die Dateien, die auf dem Server abgelegt waren, ergänzt.

Add

Add in der Git Bash

In der Git Bash gibt es die Besonderheit, dass man die ergänzten Daten erst zu den zu pushenden Daten ergänzen muss. Dies geschieht mit dem Befehl

```
git add nameDerDatei.txt
```

Sollte man alle Änderungen pushen wollen, verwendet man den Befehl

```
git add --all
```

Auch hier gilt wieder, dass das mit zwei Bindestrichen, nicht mit einem geschrieben wird.

Commit

Inzwischen ist es kaum noch möglich, dass man Dateien in ein Repository pusht, ohne dass eine Commit-Message mitgegeben wird. Hier sollte man darauf achten, dass in dieser Nachricht eine sinnvolle Erklärung geschrieben wird, welche dem nachfolgenden Entwickler aufzeigt, was genau geändert wurde.

Commit im Git Bash

Nach dem Add-Befehl muss man den Commit-Befehl eingeben, ansonsten kann man in der Bash nicht pushen. Der Befehl lautet:

```
git commit -m “deine Nachricht in Anführungszeichen”
```

Sobald die Nachricht eingegeben wurde, kann man die Dateien pushen.

Commit in Eclipse

In Eclipse geht man wie üblich über den Reiter “Team” und wählt “Commit” aus.
Nach der Auswahl öffnet sich ein Fenster am unteren Rand, welches zum Git Staging gehört. Hier werden in einem Fenster die nicht hinzugefügten Dateien angezeigt, im Fenster darunter kann man die Dateien sehen, die zum Push hinzugefügt wurden, aber nur lokal abgelegt sind.

Abb 23: Zwei Dateien, die nicht hinzugefügt wurden

Die nicht hinzugefügten Dateien kann man jetzt entweder per Drag-and-Drop in das untere Fenster ziehen, oder man wählt die Dateien aus und bestätigt das Verschieben über das grüne Plus. Über das doppelte Plus schiebt man alles in die Ablage “Staged Changes”.

Abb 24: Auswahl von ein- oder mehreren Dateien für den Push

Nachdem man alle Dateien ausgewählt hat, die man pushen möchte, kann man im rechten oberen Rand die Commit-Message angeben. Auch hier sollte eine sinnvolle Nachricht eingegeben werden, sodass der Nachfolger schnell verstehen kann, was genau geändert wurde.
Anschließend kann man mit dem Commit and Push Button bestätigen. Hiermit werden die lokalen Daten mitsamt der Commit Message auf den Server gepusht.
Wenn man nur über den Button Commit abgeht, werden die Dateien als Push bereit gekennzeichnet, aber nicht in das Repository auf GitHub gepusht. Auch dies kann nützlich sein, wenn man mit einigen Dateien bereits fertig ist, diese aber noch nicht pushen möchte, da man noch unfertige Dateien hinzufügen möchte.

Push

Ein Push wird ausgeführt, um die lokalen Daten auf den Server zu laden. Man muss vorher eine Commit Message verfasst haben, als auch die gewünschten Dateien hinzugefügt haben.

Push mit Git Bash

Um in der Git Bash zu pushen, verwendet man den Befehl:

```
git push
```

Da alles weitere in den vorherigen Schritten erledigt wurde, wird hier nur noch die Verbindung zum Server hergestellt und die Daten werden übermittelt.

Merge Konflikte

Merge Konflikte entstehen, wenn die lokale Datei nicht mehr mit der Datei auf dem Server übereinstimmt. Es kann dadurch passieren, dass einer der anderen Nutzer etwas an der selben Datei geändert hat, an der man lokal gearbeitet hat, weswegen zwei verschiedene Versionen derselben Datei vorhanden sind. Diese Konflikte lassen sich sehr einfach beheben, indem man die Versionen der Datei miteinander vergleicht und entscheidet, was davon verwendet wird und was gelöscht wird.
Generell sollte man Merge Konflikte vermeiden, da diese sehr viel Zeit und Nerven kosten können. Um diese zu vermeiden, sollte man immer, bevor man anfängt zu arbeiten, einen Pull-Request verschicken. Dadurch werden Merge Konflikte zwar nicht gänzlich behoben, aber das Risiko, dass es zu einem Konflikt kommen kann, ist deutlich verringert.

Merge Konflikte in der Git-Bash beheben

Sollte es hier zu einem Merge Konflikt kommen, so wird der Push-Befehl abgebrochen und man bekommt eine Fehlermeldung ausgeworfen:

```
$ git push
To https://github.com/AndreasRoth96/neues_repo.git
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/AndreasRoth96/neues_repo.
git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. If you want to integrate the remote changes,
hint: use 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Abb 25: Fehlermeldung bei einem Merge Konflikt

Um hier sehen zu können, welcher Fehler hier vorliegt, muss man die lokale Datei mit der Datei im Repository abgleichen. Dafür muss man die Datei mit folgenden Befehlen holen:

```
git merge
git pull
```

Jetzt kann man die Dateien miteinander abgleichen. Dies tut man mit dem Befehl:

```
git mergetool
```

Hier sieht man jetzt folgenden Abschnitt:

```
<<<<<<< HEAD
// Änderungen aus dem aktuellen Branch
=======
// Änderungen aus dem zusammenzuführenden Branch
>>>>>>> branch-name
```

Hierbei kann man seine eigene Version sehen, als auch die bereits gepushte. Man kann hier die Daten, die man herausnehmen will, löschen und anschließend wieder mit add, commit und push die bereinigte Datei hochladen.

Merge Konflikte in Eclipse beheben

Wenn ein Merge Konflikt in Eclipse auftritt, kann man die Datei mit dem Pull-Befehl herunterladen. Anschließend erhält man eine Datei namens Dateiname.orig.
In dieser Datei kann man sehen, was sich genau in der lokalen Version zur Version im Repository unterscheidet.

Abb 26: Merge Konflikt in Eclipse

Hier kann man die Zeilen löschen, die im Konflikt zueinander stehen. Anschließend kann man per Commit die angepasste Datei wieder pushen.