

# git Good

Die wichtigsten Fragen und Antworten zum git-Einstieg

## Was ist git?

git ist ein Versionskontrollsystem. Wenn ihr mit mehreren Personen an demselben Projekt arbeiten wollt, behält es den Überblick über alle Veränderungen an allen Dateien und fügt diese, sofern möglich, zusammen.

Dadurch könnt ihr sehen:

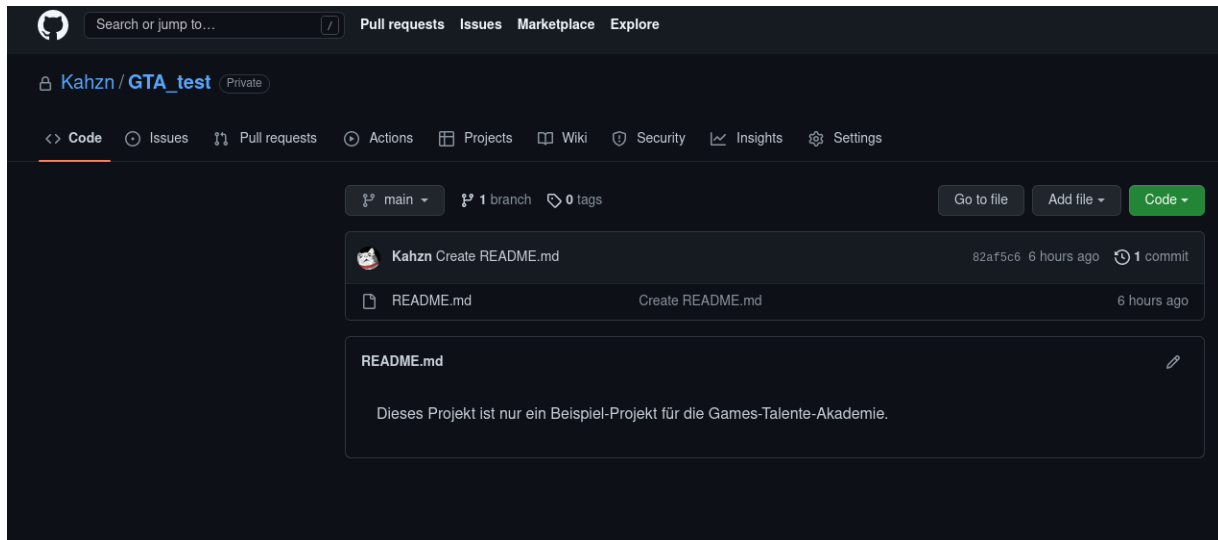
- Welche Änderungen vorgenommen worden sind.
- Wer die Änderungen vorgenommen hat.
- Wann die Änderungen vorgenommen worden sind.
- Warum die Änderungen notwendig waren.

Darüber hinaus gibt euch git die volle Kontrolle über euer Projekt. Das heißt, dass jede Änderung nicht nur nachverfolgt, sondern auch rückgängig gemacht (*reset / revert*) werden kann. git bietet eine Menge Funktionen, die ihr anfangs nicht unbedingt alle braucht. Deswegen wollen wir uns in dieser Hilfestellung auf die wichtigsten Grundfunktionen konzentrieren.

## Was ist GitHub?

GitHub ist eine Online-Plattform, auf der eure Projekte für alle Mitglieder erreichbar abgelegt werden können. Wenn ihr eure Änderungen “pusht”, werden diese auf die Projektversion auf dem GitHub-Server angewandt. Wenn ihr “pullt”, holt ihr euch den aktuellsten Stand des Projektes vom GitHub-Server. Dafür braucht ihr eine Internetverbindung, git auf eurem Rechner und einen GitHub-Account.

GitHub bietet euch außerdem weitere Funktionen, die ihr nutzen könnt, wie z.B. einen Wiki-Bereich zu eurem Projekt, Aufgabenmanagement mit GitHub Issues und Statistiken zu eurem Projekt über GitHub Insights.



## Muss ich für git Programmieren können?

**Nein.**

git erfordert keine Programmierkenntnisse, dafür aber etwas Geduld und den Willen, sich "hineinzudenken". Wenn man dies aber tut, ist git unheimlich hilfreich in der Projektarbeit.

Man kann git zwar problemlos über die Kommandozeile bedienen, aber es gibt auch viele grafische Oberflächen für git, die genauso gut funktionieren (z.B. GitHub Desktop, Tortoise Git, Fork, Git Extensions usw.). Oft wird Hilfestellung zu git im Internet über Kommandozeilenbefehle gegeben. Das ist aber kein Problem: Ihr könnt beides parallel benutzen.

## Wie bekomme ich ein git-Projekt auf meinen Rechner?

### Git installieren

Zu allererst braucht ihr dafür git.

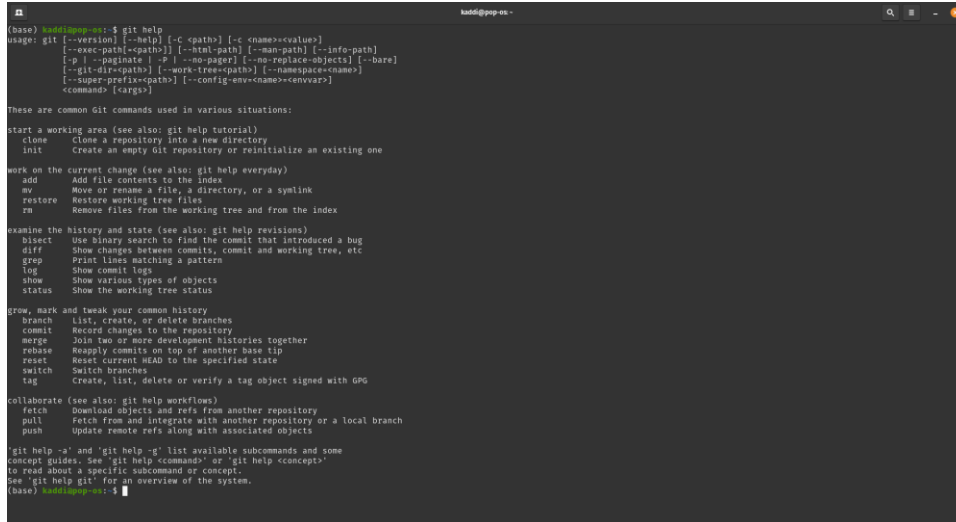
Wenn ihr das nicht schon habt, könnt ihr das hier herunterladen:

- Windows: <https://git-scm.com/download/win>
- Mac: <https://git-scm.com/download/mac>
- Linux: <https://git-scm.com/download/linux>

Verändert bei dieser ersten Installation am besten nichts an den Voreinstellungen und lasst diese einfach durchlaufen.

Beachtet bitte, dass in manchen Fällen nach der Installation ein Systemneustart erforderlich ist, damit alles richtig funktioniert.

Nach der Installation und dem Neustart sollte git auf eurem Rechner funktionieren. Ihr könnt das über die Kommandozeile prüfen, indem ihr z.B. “git help” eintippt. Wenn euer System ungefähr wie folgt antwortet, habt ihr git erfolgreich installiert:



```
(base) haddi@pop-os:~$ git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path<path>] [--html-path] [--man-path] [--info-path]
      [-p | --paginate] [-P | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir<path>] [--work-tree<path>] [--namespace<name>]
      [--super-prefix<path>] [--config-env<name>=<envvar>]
      <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
 clone      Clone a repository into a new directory
 init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
 add        Add file contents to the index
 mv         Move or rename a file, a directory, or a symlink
 restore    Restore working tree files
 rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
 bisect     Use binary search to find the commit that introduced a bug
 diff      Show changes between commits, commit and working tree, etc
 grep      Print lines matching a pattern
 log       Show commit logs
 show      Show various types of objects
 status    Show the working tree status

grow, mark and tweak your common history
 branch    List, create, or delete branches
 commit    Record changes to the repository
 merge     Join two or more development histories together
 rebase    Reapply commits on top of another base tip
 reset     Reset current HEAD to the specified state
 switch    Switch branches
 tag       Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
 fetch     Download objects and refs from another repository
 pull      Fetch from and integrate with another repository or a local branch
 push      Update remote refs along with associated objects

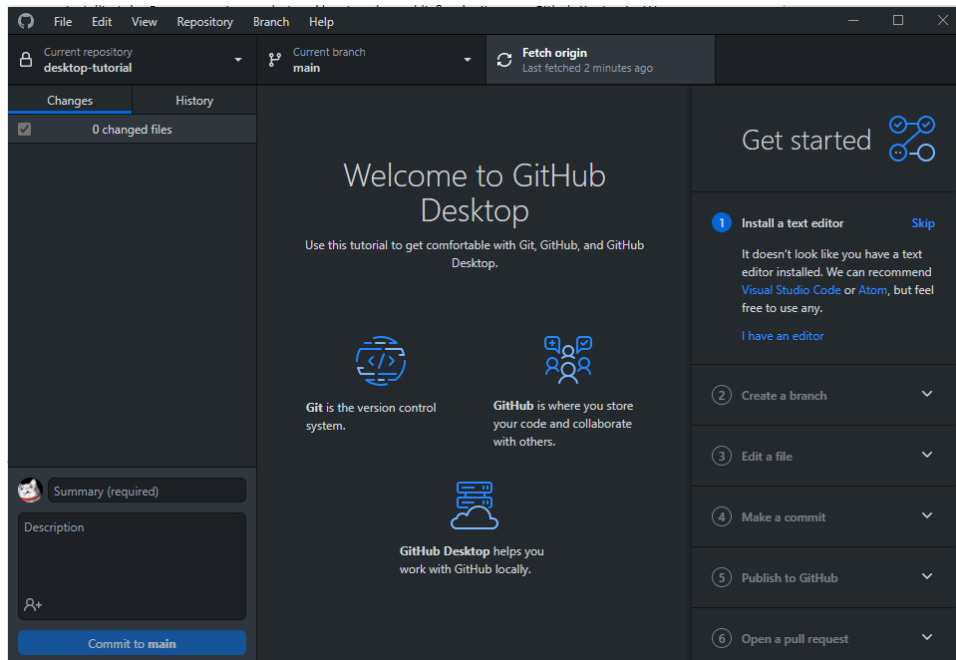
'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
(base) haddi@pop-os:~$
```

## GitHub Desktop installieren

Wenn ihr Mac oder Windows benutzt, könnt ihr darauf GitHub Desktop installieren. Das ist eine grafische Benutzeroberfläche für git, die ihr kostenlos nutzen könnt.

Die Installationsdateien bekommt ihr hier: <https://desktop.github.com/>

Installiert das Programm wie gewohnt und loggt euch anschließend mit eurem Github-Konto ein. Wenn ihr GitHub-Desktop dann startet, sollte die Oberfläche in etwa wie folgt aussehen:

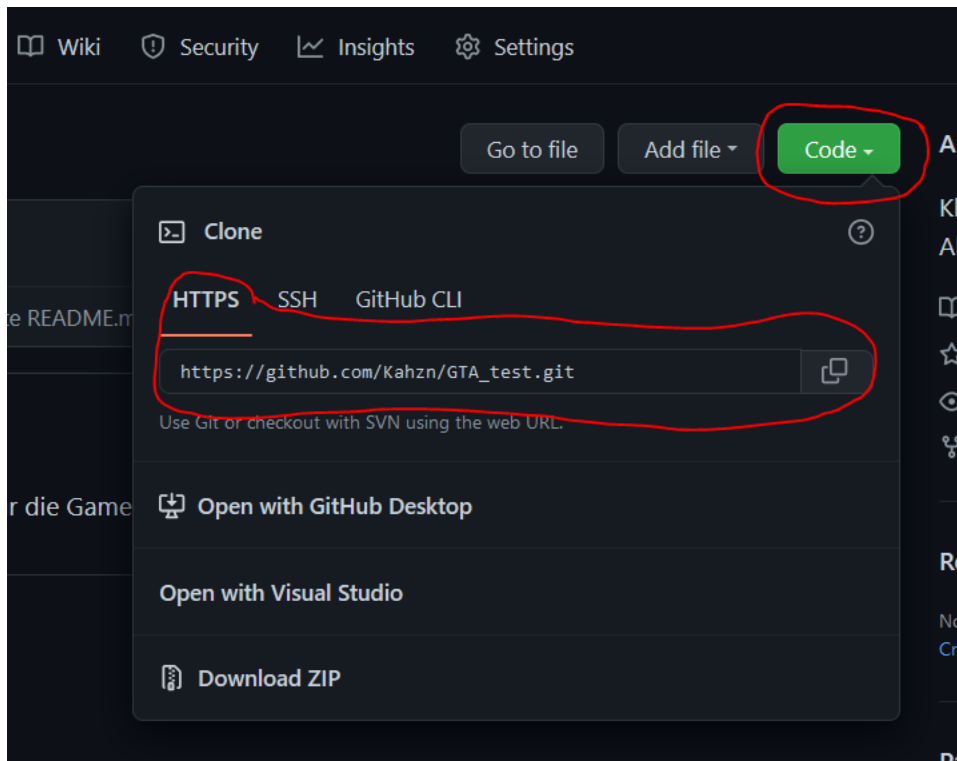


[Hinweis: Ihr braucht nicht unbedingt GitHub Desktop, um auf ein GitHub-Projekt zuzugreifen. Das funktioniert auch mit jeder anderen git-Oberfläche, auch mit der Kommandozeile. Wenn ihr also schon eine andere git-Oberfläche installiert habt, wie z.B. TortoiseGit oder Fork oder ähnliches, dann müsst ihr GitHub Desktop nicht unbedingt zusätzlich installieren.]

## Ein Repo klonen

Zu Beginn des Projekts bekommt ihr ein sogenanntes "Repository" (kurz *repo*) von euren Teamern. Ein Repository ist ein git-Projekt, das auf einem Server für euch zur Verfügung gestellt wird. Um es anfangs herunterzuladen, müsst ihr es "klonen". Hierfür gibt es mehrere Herangehensweisen, die alle gleichermaßen funktionieren.

Zu allererst braucht ihr den **repo-Link**. Entweder bekommt ihr diesen direkt von uns oder ihr könnt ihn auf Github im repo finden:



In der Kommandozeile:

Das erste Setup ist in der Kommandozeile relativ simpel. Im Grunde ist es nur ein Befehl:

- **git clone [url des repos] [optional: Neuer Verzeichnisname]**
- Beispiel: `git clone https://github.com/libgit2/libgit2 meineLibGitKopie`

Wenn ihr diesen Befehl eingibt, werdet ihr dazu aufgefordert, euch bei GitHub zu authentifizieren. Da GitHub seit August 2021 seine Sicherheitseinstellungen geändert hat, reicht es nicht mehr, seinen Nutzernamen und Passwort anzugeben. Stattdessen braucht ihr einen *Personal Access Token (PAT)*, der an euren Rechner gebunden ist. Diesen könnt ihr dann anstelle des Passwortes eingeben.

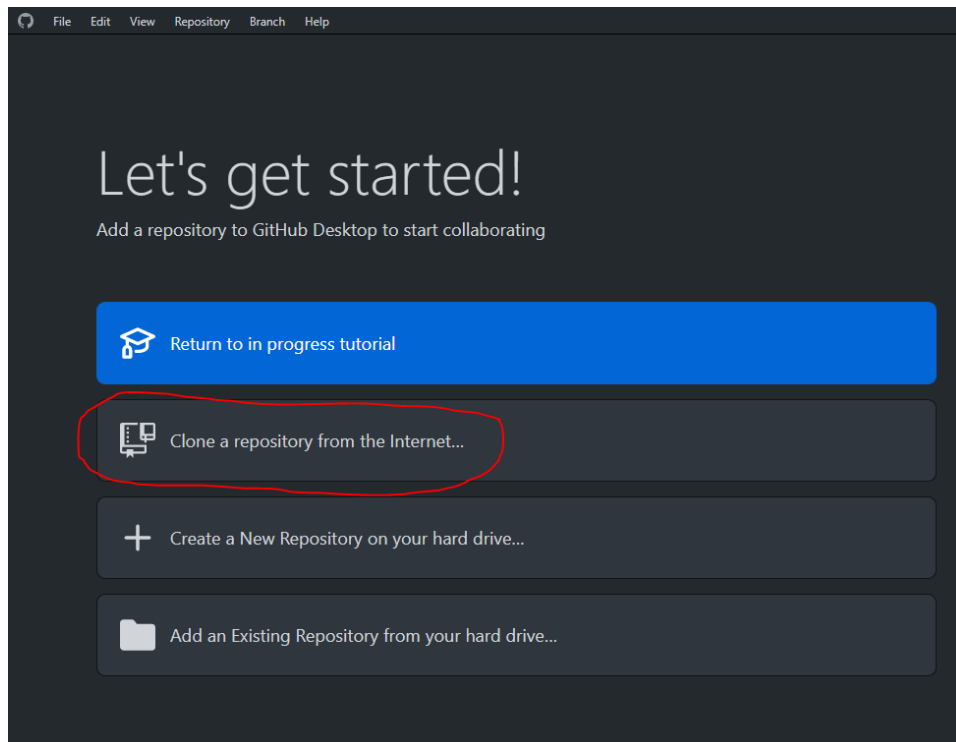
Eine Anleitung, wie ihr den PAT erzeugt, findet ihr hier:

<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>

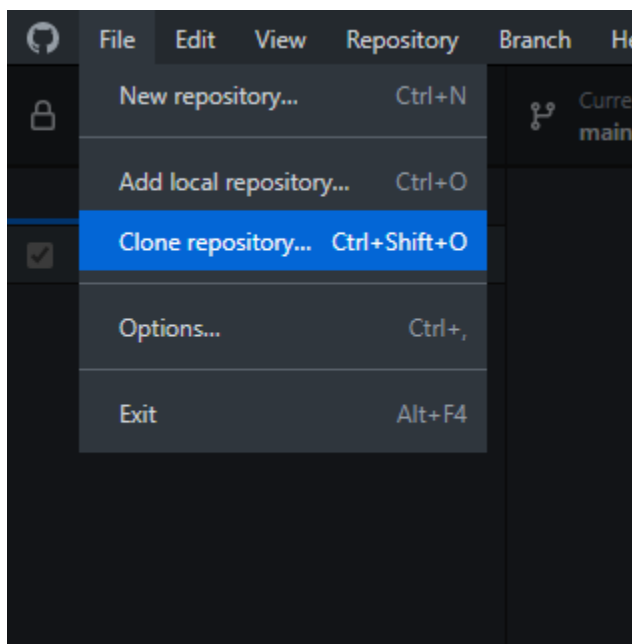
Sobald ihr euch erfolgreich authentifiziert habt, legt git ein entsprechendes Verzeichnis an, initialisiert ein .git-Verzeichnis, lädt alle Daten des Repositories herunter und checkt eine Arbeitskopie der aktuellsten Version aus.

In GitHub Desktop:

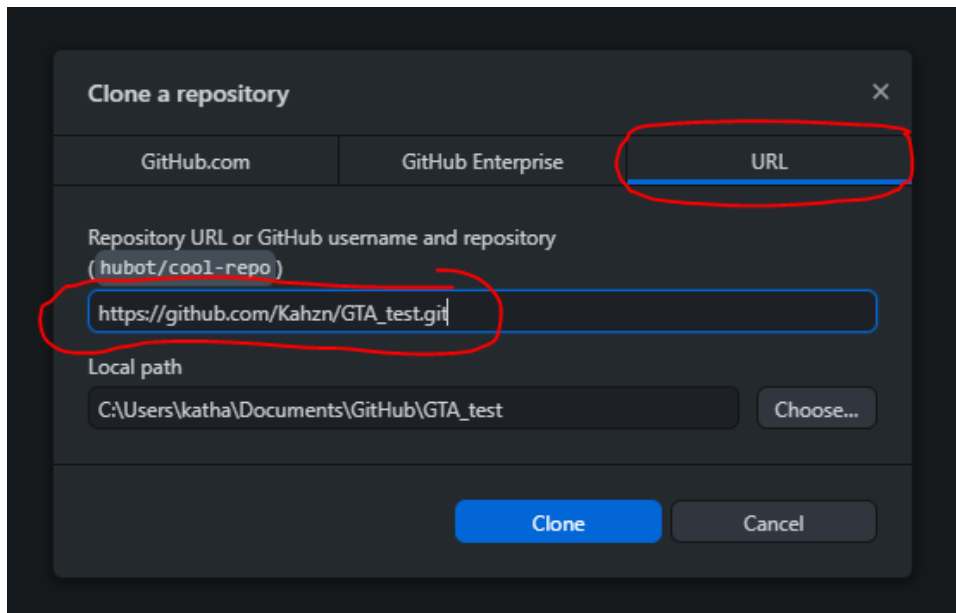
In GitHub Desktop findet ihr die *clone*-Funktion hier...



Oder im Kontextmenü unter “File / Datei”:



Dort könnt ihr das Repo entweder aus der Auswahl eurer hinzugefügten Repos in Github auswählen, oder ihr gebt den repo-Link im Bereich “URL” an:

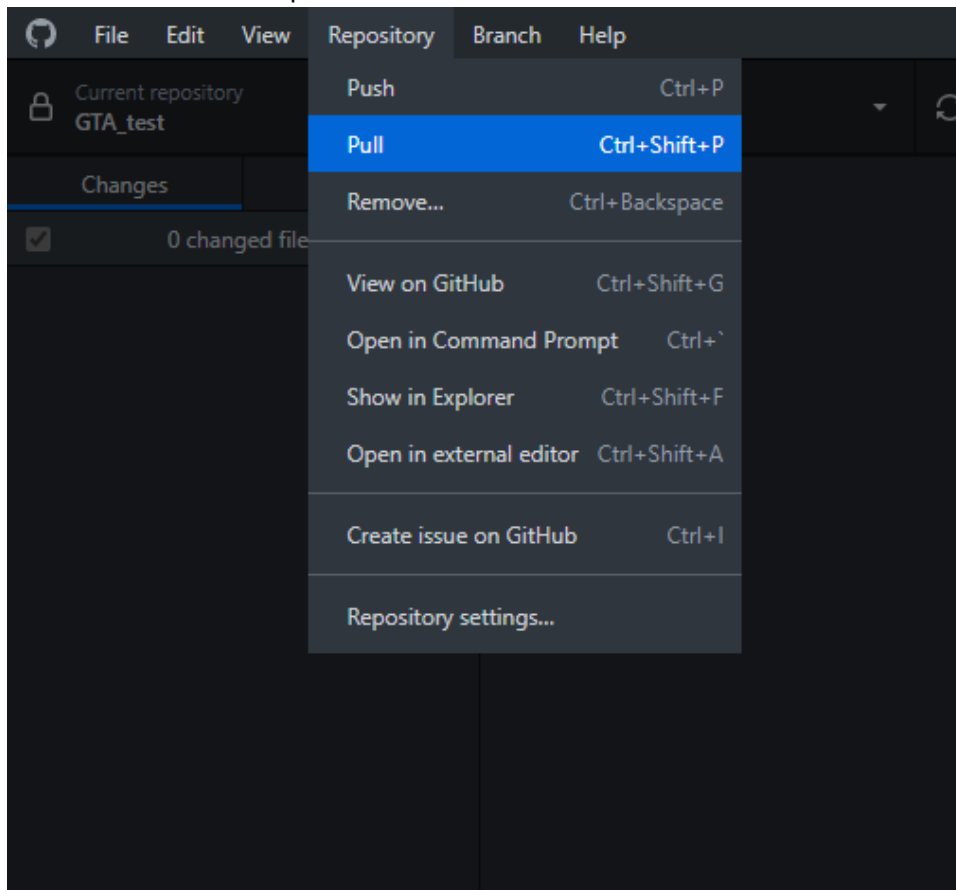


Wenn ihr dann auf “clone” klickt, legt GitHub Desktop einen Ordner für euch an und initialisiert das repo auf eurem PC.

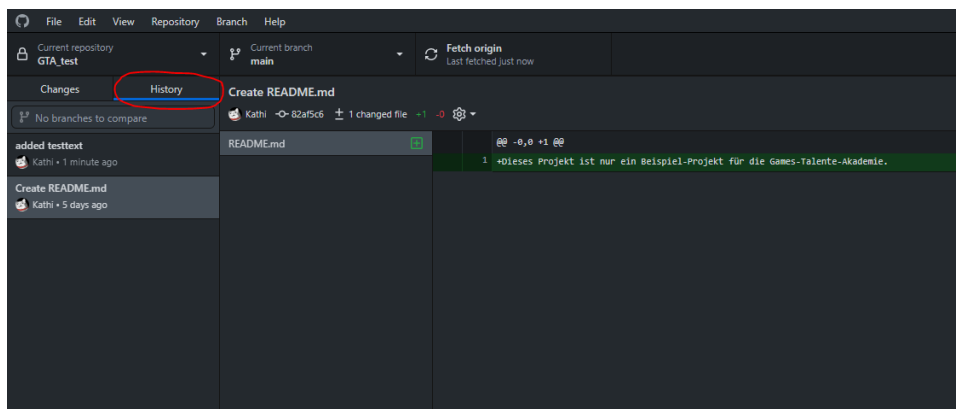
Wie sieht ein typischer Arbeitsablauf in Git aus?

## Git pull

Mit dem *pull*-Befehl ruft ihr immer den aktuellsten Stand des Repositories vom Server ab und fügt diesen in eure lokale Kopie ein.



Zu Beginn jeder Arbeitssession und kurz bevor ihr etwas auf den Server pusht solltet ihr immer den neuesten Stand des Projektes “pullen”. Neueste Änderungen in eurem Projekt seht ihr in der Spalte “History”.

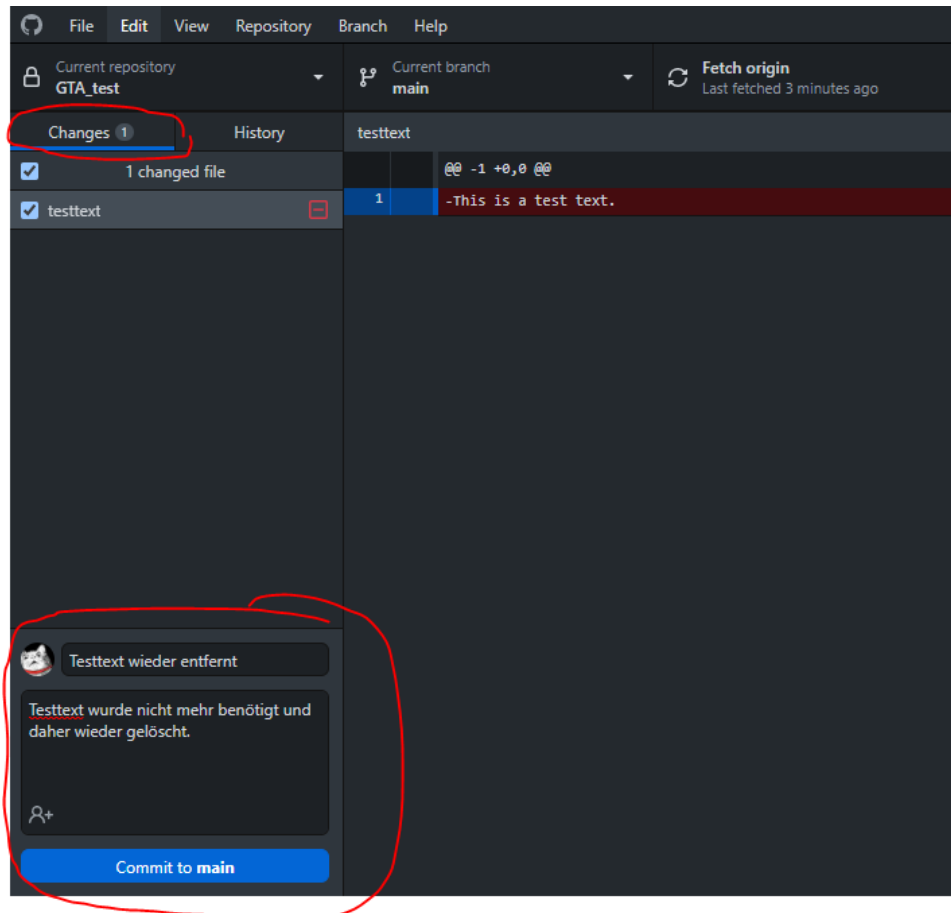




## Git commit

Der *commit*-Befehl speichert die letzten Änderungen am Projekt und versieht diese mit Metadaten, damit ihr später nachvollziehen könnt, wer diese Änderungen wann vorgenommen hat. Jeder Commit muss eine Commit-Nachricht enthalten, in der die Änderungen beschrieben werden.

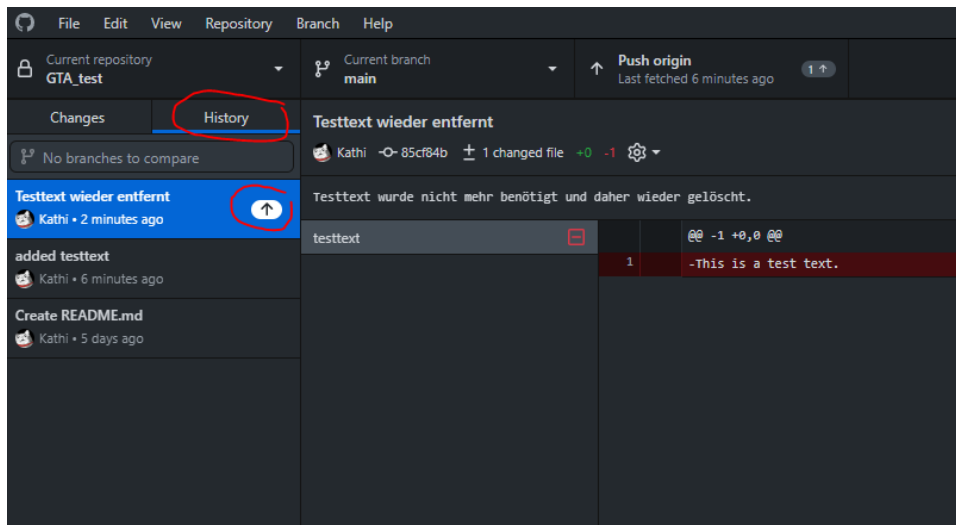
Ein Beispiel für eine Commit-Nachricht wäre "Testtext wieder entfernt."



Da Commits bei vielen Funktionen in git eine große Rolle spielen, macht es Sinn, möglichst kleinschrittig zu committen. Wenn ihr also z.B. einen Bug fixen, ein neues Skript erstellen und die Grafiken in einer Szene überarbeiten wollt, macht es wahrscheinlich Sinn, dies auf verschiedene Commits aufzuteilen, sofern diese drei Aktionen nicht direkt aneinandergekoppelt sind.

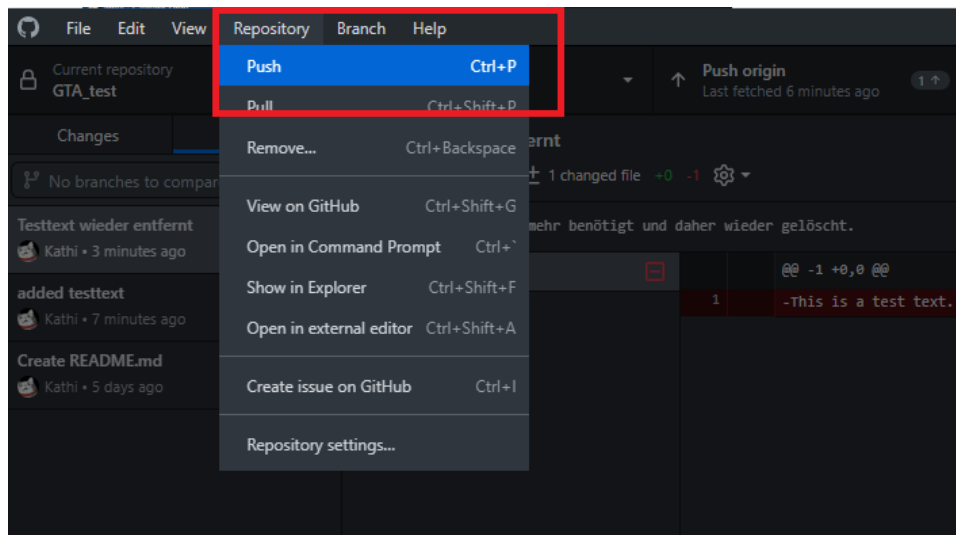
Wichtig zu verstehen ist, dass ein Commit noch nicht mit den anderen in eurem Team geteilt wird. Um diesen auf den Server zu bringen und damit für alle verfügbar zu machen, müsst ihr ihn "pushen".

So lange ihr einen Commit noch nicht gepusht habt, erscheint er bei GitHub Desktop unter "History", aber mit einem Pfeil nach oben versehen.



## Git push

Mit git *push* lädst du deine letzten Commits auf den Server hoch.



**Wichtig:** Stelle vorher bitte sicher, dass du den neuesten Stand auf deinem Rechner hast, indem du vorher “pullst”! Wenn dabei Konflikte in Dateien entstehen, musst du diese zunächst auflösen und neu committen, bevor du wieder pushen kannst.

Ein “typischer” Ablauf wäre demnach: Pull --> Änderungen vornehmen --> Commit(s) --> Pull --> Push

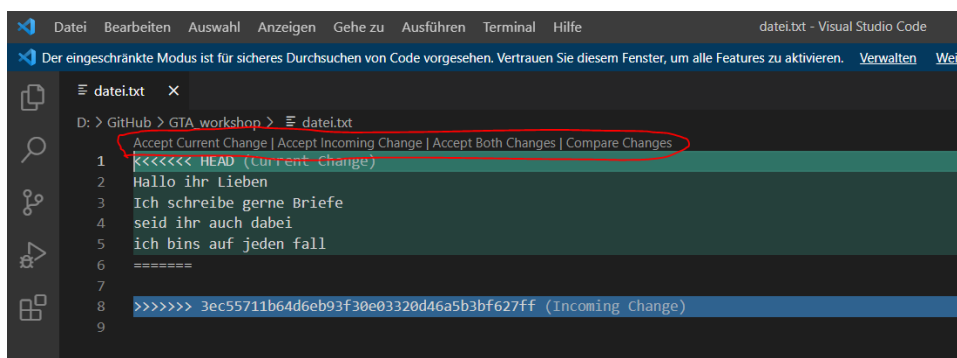
## Was tun bei Konflikten?

Konflikte in git bedeuten immer, dass dieselbe Datei an derselben Stelle verändert worden ist und git nicht wissen kann, welche Änderung jetzt übernommen werden soll. In diesem Fall müsst ihr das entscheiden und entsprechend markieren. Hierfür gibt es verschiedene Möglichkeiten:

- Eine der beiden Änderungen vollständig übernehmen.
- Die Änderungen in einem Diff/Merge-Tool miteinander vergleichen und entsprechende Zeilen auswählen, die übernommen werden sollen
- Die Änderungen in einem Editor bearbeiten und entsprechende Zeilen auswählen, die übernommen werden sollen bzw. Die Zeilen löschen, die nicht übernommen werden sollen

Alle diese Möglichkeiten sind gleichwertig, das heißt sie funktionieren gleich "gut" für git. Sobald ihr eure Änderungen vorgenommen und den Konflikt aufgelöst habt, müsst ihr die Konfliktdatei speichern und neu committen. Wenn ihr Visual Studio Code auf eurem PC habt, könnt ihr das Konfliktmanagement darüber lösen. Visual Studio hebt euch die Änderungen, die im Konflikt miteinander sind, dann farblich hervor und bietet darüber verschiedene Optionen an:

- Accept Current Change:
  - Übernimmt die lokalen Änderungen
  - Überschreibt die Änderungen vom Server
- Accept Incoming Change:
  - Übernimmt Server-Änderungen
  - Überschreibt lokale Änderungen
- Accept Both:
  - Übernimmt beide nacheinander
- Compare Changes:
  - Zeigt euch die Änderungen nebeneinander an statt übereinander

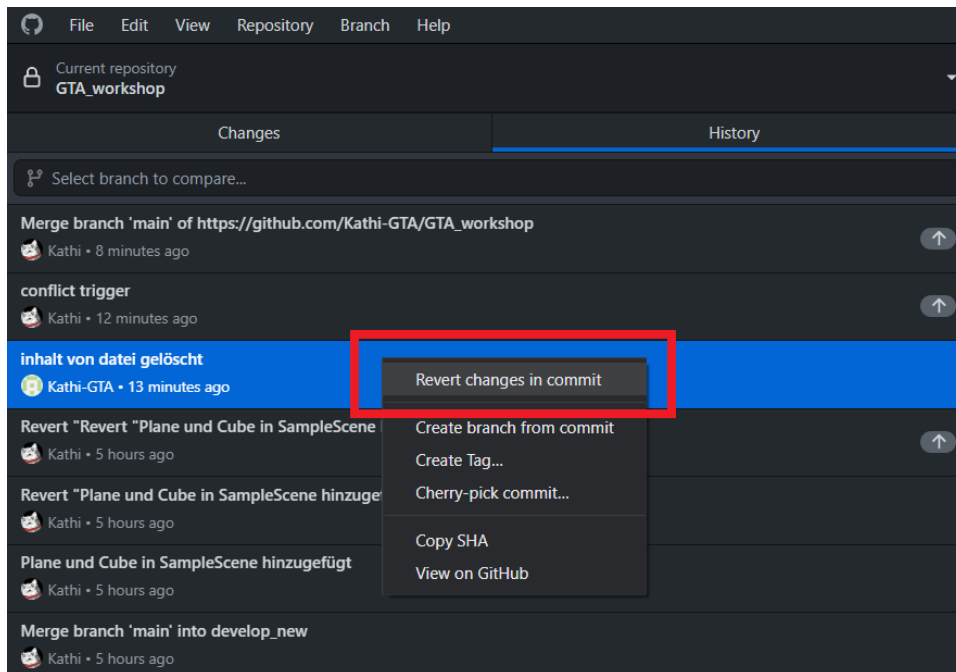


Wenn ihr auf eine der Optionen geklickt habt, wird diese angewandt. Danach müsst ihr die Datei speichern und VS Code wieder schließen. GitHub Desktop erzeugt dann automatisch einen neuen

Commit für die Konfliktauflösung.

## Wie mache ich etwas rückgängig in GitHub Desktop?

Rechtsklickt auf den Commit, den ihr rückgängig machen wollt, in der History. Wählt “Revert Changes in Commit” aus und bestätigt, dann wird die Änderungen aus diesem Commit rückgängig gemacht.



## Hilfreiche Links

Git Dokumentation

- <https://git-scm.com/book/de/v2>

Youtube-Tutorials

- <https://youtu.be/N-vFTYEgguU>

Hilfe! Ich habe ein Problem, das hier nicht hineinpasst!

Wende dich an unser Technik-Team, das steht dir gerne mit Rat und Tat zur Seite 😊.