Git Tutorial

Git installieren

Link zu Git Homepage

- installieren von git für Window
- macOS und Linux sollten bereits mit vorinstalliertem Git kommen

Git einrichten

- git config konfigurieren
- überprüfen ob config eingerichtet ist

```
git config --list
```

- mindestens E-mail und User-Name muss eingerichtet werden

```
# User Name erstellen
git config --global user.name "<Your Name>"

# E-Mail erstellen
git config --global user.email "<your email address>"
```

optional direkt immer `main` als Haupt-Branch festlegen

```
git config --global init.defaultBranch <name>
```

Repository initialisieren

```
git init
```

wenn ihr nur diesen Befehl eingebt und nicht vorher Git gesagt habt, dass der Hauptbranch main heißen soll wird der Hauptbranch master sein.

um das zu ändern müsst ihr NACH des init Befehls folgenden Befehl ausführen:

```
git branch -m main
```

um direkt bei der Initialisierung den Hauptbranch `main` zu nennen:

```
git init -b main
```

Status des neuen Repositories und überhaupt zu prüfen

```
git status
```

Dateien tracken bzw. zu Staging hinzufügen

- Datei im Arbeitsverzeichnis anlegen und mittels Git command tracken

```
git add <name_der_datei_mit_endung>
```

- wenn mehrere Dateien zu tracken bzw. zum Staging hinzugefügt werden sollen

```
git add .
```

- Datei wieder aus der Staging-Ebene entfernen

```
git rm --cached <name_der_datei>
```

Der erste Commit

- mit einem Commit werden alle Dateien, welche sich gerade im Staging befinden in das lokale Repository (Storage), mit dem aktuellen Staging Zustand, aufgenommen

```
git commit -m "<Deine Commit Message>"
```

Der zweite Commit

- lege eine neue Datei Deiner Wahl an und füge diese zum Staging hinzu mit `git add` und commite anschließen diese neuen Stand Deines Arbeitsverzeichnisses

`git log`

- git log gibt eine Übersicht über alle commits mit den dazugehörigen Daten
- optional nimmt `git log` die `--oneline` Flag um die Übersicht kürzer anzuzeigen

```
git log --oneline
```

Reise in die Vergangenheit

- mit dem Befehl `git checkout <hash>` kann ich den früheren Zustand eines Projektes mir anschauen im sog. 'detached HEAD Mode'
- um wieder zurückzukommen in die 'Gegenwart'
  - `git checkout -` bzw. `git switch -` oder
  - `git checkout bzw. switch main`

ACHTUNG

- manchmal möchtest Du Änderungen verwerfen und zu einem früheren Zeitpunkt Deines Projetes zurückkehren, nicht nur um zu schauen wie es zu dem Zeitpunkt aussah, sondern um von dem Zeitpunkt aus neu zu starten

```
git reset --hard <commit-hash>
```

- wenn die Änderungen, die danach gemacht wurden nicht gelöscht werden sollen, dann wird stattdessen aus dem vergangen commit ein neuer Branch abgeleitet und nicht resetet

Branching

um neue Features zu implementieren oder zu testen generieren wird ein neuer ‘Branch’ aus dem 'main' Branch heraus erstellt bzw. kopiert, welcher dann unabhängig von diesem bearbeitet wird

1. neuen branch erstellen:

   ```
   git branch <branch_name>
   ```

2. in den neuen branch wechseln

   ```
   git checkout <branch_name>
   ```

   oder besser und neuer

   ```
   git switch <branch_name>
   ```

oder

Branch anlegen und direkt in den neuen Branch wechseln

```
git switch -c <new_branch_name>

# oder mit checkout
git checkout -b <new_branch_name>
```

Merging

- wenn das neue Tool im neuen Branch zur Zufriedenheit erstellt wurde und alles funktioniert kann ich den neuen Branch wieder mit dem Hauptbranch zusammenführen und damit mein Projekt 'offiziell' aktualisieren

ACHTUNG

- erst in den Branch wechseln, in welchen hinein gemerged werden soll
- hier: ich möchte die Änderungen von 'weather' in 'main' übertragen, weswegen ich den merge aus dem main-Branch heraus machen muss

```
git merge <Feature-Branch>
```

- nun sind der Main-Branch, so wie der Feature-Branch, wieder auf dem selben Stand!

Optional kann ich nun den Feature Branch wieder löschen

- dazu muss ich mich aber in einem anderen, als den zu löschenden Branch, befinden

```
git branch -d <branch_name>
```

Verknüpfen mit einem remote Repository (Github)

Github Webseite

1. Auf Github ein leeres Repository anlegen
2. Den 'HTTPS' Link zu dem neuen Remote Repo kopieren
3. im lokalen Repository in der commandline folgende Befehle ausführen:

    1. `git remote add origin <Link zu Remote Repository>`
    2. `git branch -M main`
4. das lokale Repository auf das remote Repository pushen
    1. `git push origin main`

- `origin` ist der Name unserer Remote Verbindung

Remote Repository lokal clonen

Ein bestehendes Online Repository auf den lokalen Computer kopieren und dann dort Änderungen machen

1. Auf dem Github Repository den Repository Link kopieren
2. Auf dem lokalen Computer einen Ordner anlegen in welchen das neue Repository kopiert werden soll.
3. zum 'klonen' des online Repos folgenden Befehl ausführen: `git clone <Link zum Repo>`
4. damit wird lokal eine Kopie des Github Repos gedownloaded.
5. Dieses Repo ist auch schon direkt mit dem Github Repo verknüpft und enthält alle bereits getätigten Commits.
6. Mit `git push origin main` können dann lokale Änderungen wieder auf das Github Repo hochgeladen werden, SOFERN Ihr die Erlaubnis hierfür habt.

Änderungen eines Remote Repos auf Lokales Repo übertragen

```
git pull origin main
```

Merge Konflikte erzeugen und auflösen

Vorbereitung

1. Erstelle neues Git-Repository oder wechsel in ein vorhandes
2. Erselle eine neue Datei und füge Inhalte hinzu
3. Stage und commite die Datei

```
git add new_file.txt
git commit -m "neue Datei erstellt"
```

4. Erstelle neuen ‘feature’ Branch und wechsel in diesen

```
git checkout -b feature
```

Änderung in beiden Branches vornehmen

1. im ‘feature’-Branch ändere new_file.txt
2. Stage und commite die Änderungen.

```
# im feature Branch
git add new_file.txt
git commit -m "changes in feature branch"
```

3. Wechsle nun zurück zum ‘main’-Branch
4. Im ‘main’-Branch mache nun ebenfalls Änderungen (aber andere) in der Datei
5. Stage und commite die Änderungen dort ebenfalls

```
# im 'main'-Branch
git add new_file.txt
git commit -m "changes in main branch"
```

Merge Versuch

1. den merge Befehl ausführen

```
# im main branch
git merge feature
```

2. Git weist Dich nun darauf hin, dass es den Merge nicht ausführen kann, da es zu einem Konflikt in der ‘new_file.txt’ gekommen ist. Heißt. Git hat zwei Unterschiedliche Versionen von dieser Datei und weiß nicht wie es diese behandeln soll

```
Auto-merging new_file.txt
CONFLICT (content): Merge conflict in new_file.txt
```

Konflikt auflösen

1. Git wird Dir nun beide Versionen der new_file.txt in der Datei anzeigen. Wenn Du die Datei in einem Editor nun öffnest wird sie folgendermaßen ausschauen

```
Zeile 1
<<<<<<<<<< HEAD
Änderung auf main-Branch
==========
Änderung auf feature-Branch
>>>>>>>>> feature
```

2. Du musst die Datei manuell bearbeiten, um zu entscheiden, welche Änderungen beibehalten werden sollen. Du kannstentweder eine der beiden Versionen übernehmen oder die Änderungen zusammenführen.

```
Zeile 1
Änderung auf main-Branch
Änderung auf feature-Branch
```

Konflikt als gelöst markieren

1. Wenn Du die Datei(en) manuell nun geändert hast musst Du diese als gelöst für Git markieren.
2. Dies machst Du in dem Du die Datei(en) nochmal Staged und commitest

```
# immer noch im main branch
git add new_file.txt
git commit -m "Conflict resolved"
```