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

- sollte ich von einer früheren Version neu beginnen wollen und meine Änderungen bis dahin verwerfen, kann ich mit einem hard Reset von einem alten Zeitpunkt neu beginnen

```
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 wir einen neuen ‘Branch’ der erst einmal unabhängig vom ‘main’ Branch ist

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 löschen Branch, befinden

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