Das erste Repo

Initialisieren

Wir wollen zunächst eine kleine statische Webseite entwickeln und unter Versionskontrolle stellen.

```
$ mkdir webseite
$ cd webseite
$ git init
Leeres Git-Repository in /home/frank/Downloads/webseite/.git/ initialisiert
```

Nun steht das Verzeichnis `webseite` unter Versionskontrolle. Das (lokale) Git-Repository befindet sich im Unterverzeichnis `.git`:

```
$ ls -la
insgesamt 132
drwxr-xr-x  3 frank frank   4096 24. Okt 13:32 .
drwxr-xr-x 21 frank frank 122880 24. Okt 13:32 ..
drwxr-xr-x  7 frank frank   4096 24. Okt 13:32 .git
```

Grundkonfiguration

Bevor man sinnvoll mit Git arbeiten kann, sollte man zunächst seinen Namen und seine Mailadresse korrekt einstellen:

```
git config user.email "meine@mail.adresse.hier"
git config user.name "John Doe"
```

Diese Befehle speichern die Einstellungen nur für das Repository, in dem wir gerade arbeiten, wenn man die Einstellungen für alle Repos auf einem Rechner vornehmen möchte, muss man das Flag `--global` ergänzen.

Gespeichert werden die für ein Repository geltenden lokalen Einstellungen in der Datei `.git/config`:

```
[frank@rita webseite]$ git config  user.name "John Doe"
[frank@rita webseite]$ git config  user.email "john.doe@nirgends.nix"
[frank@rita webseite]$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[user]
    name = John Doe
    email = john.doe@nirgends.nix
```

Repository Status anzeigen lassen

Mit dem Befehl `git status` kann man sich den aktuellen Status des Repos anzeigen lassen:

```
[frank@rita webseite]$ git status
Auf Branch main

Noch keine Commits

nichts zu committen (erstellen/kopieren Sie Dateien und benutzen
Sie "git add" zum Versionieren)
```

Ein erster Commit

Um den git-Workflow zu verstehen, muss man drei Begriffe unterscheiden: Das Arbeitsverzeichnis („Working Directory“) den Index („Staging Area“) und das eigentliche Repository.

- **Arbeitsverzeichnis (Working Directory):** Das ist Verzeichnis, welches man zuvor mit `git init` unter Versionskontrolle gestellt hat mit allen seinen Dateien und Unterverzeichnissen, so wie man es auf der Festplatte vorfindet. Das „spezielle“ Verzeichnis `.git` wird dabei ignoriert, es dient der internen Verwaltung der Abläufe durch git.
- **Index („Staging Area“)**: Im Index werden zunächst alle Dateien eingetragen, die in einem nächsten Schritt zu einem Snapshot zusammengefasst und im Repository gespeichert werden sollen. Der Sinn des Indes erschließt sich nicht unmittelbar, da man dazu neigt, sich vorzustellen, dass man nacheinander Änderungen in deinem Arbeitsverzeichnis vornimmt und dabei von Zeit zu Zeit einfach Snapshots des gesamten Arbeitsverzeichnisses anlegt - das trifft aber nicht zu. Es gibt zahlreiche Anwendungsfälle, bei denen man nicht alle Änderungen des Arbeitsverzeichnisses in einem Snapshot festhalten möchte, sondern z.B. auf mehrere Snapshots aufteilen will. Außerdem kommt es häufig vor, dass sich im Arbeitsverzeichnis Dateien befinden, die man gar nicht unter Versionskontrolle stellen möchte, beispielsweise Compilate von Java Programmen (class-Dateien).
- **Repository:** Wenn man im Index alle Dateien für den nächsten Snapshot zusammengestellt hat, kann man einen neuen Snapshot erstellen. Ein solcher Snapshot heißt **Commit** und wird durch eine Hashsumme identifiziert, außerdem werden Metainformationen wie Zeit und Name des Commiters festgehalten. Ein Commit wird mit dem Befehl `git commit` durchgeführt. Nach einem Commit ist der Index stets leer, da ja alle Änderungen, die dort vorgemerkt waren, in den Snapshot überführt wurden.

Schritt für Schritt

Neue Dateien befinden sich zunächst „nur“ im Arbeitsverzeichnis und werden von git ignoriert. Mit `git status` kann man das überprüfen, solche Dateien tauchen dort in der Liste der „Unversionierten Dateien“ auf:

```
[frank@rita webseite]$ git status
Auf Branch main

Noch keine Commits

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit
vorzumerken)
    index.html
    style.css

nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutzen Sie "git add" zum Versionieren)
```

Mit dem Befehl `git add` wird eine Datei im Index vorgemerkt - das kann man sich vorstellen wie ein Einkaufswagen, in dem neue Dateien und Änderungen gesammelt werden, bis man zu einem Punkt kommt, den man sich „merken“ möchte. Im Folgenden habe ich die Datei `index.html` zum Index hinzugefügt, `style.css` jedoch nicht.

```
[frank@rita webseite]$ git add index.html
[frank@rita webseite]$ git status
Auf Branch main

Noch keine Commits

Zum Commit vorgemerkte Änderungen:
  (benutzen Sie "git rm --cached <Datei>..." zum Entfernen aus der Staging-
Area)
    neue Datei:     index.html

Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit
vorzumerken)
    style.css
```

Wenn man mit den im Index vorgemerkten Änderungen zufrieden ist, macht man einen „Commit“. Mit dem Befehl `git commit -m „Erster Commit“` legt man einen Commit mit einer Commit-Message an (Paramter `-m`). Wenn man die Commit-Message nicht mit `-m` angibt, öffnet sich ein Editor, in dem man diese bearbeiten kann.

```
[frank@rita webseite]$ git commit -m "Erster commit"
[main (Root-Commit) bb0d027] Erster commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.html
[frank@rita webseite]$ git status
Auf Branch main
Unversionierte Dateien:
  (benutzen Sie "git add <Datei>...", um die Änderungen zum Commit
vorzumerken)
    style.css
```

```
nichts zum Commit vorgemerkt, aber es gibt unversionierte Dateien
(benutzen Sie "git add" zum Versionieren)
```

Man erkennt, dass der Index wieder leer ist („nichts zum Commit vorgemerkt“) und die Datei `style.css` noch immer unversoniert ist.

Die Liste deiner Commits kann man mit `git log` ansehen:

```
[frank@rita webseite]$ git log
commit bb0d027bd6376da3d67c46bbeeb14e5fd1623581 (HEAD -> main)
Author: John Doe <john.doe@nirgends.nix>
Date:   Wed Apr 28 15:35:30 2021 +0200

    Erster commit
```

Aufgaben

(A1)

Erkläre, was man machen muss, um von der derzeitigen Situation ausgehende, die Datei `style.css` ebenfalls unter Versionskontrolle zu stellen. Welche git Befehle würdest du verwenden?

(A2)

Lege ein Verzeichnis `webseite` an, erstelle dort die Dateien `index.html` Datei ein sowie zwei weitere Verzeichnisse - `css` und `img`:

```
sbel@r107-ws15:~/git$ mkdir webseite
sbel@r107-ws15:~/git$ cd webseite
sbel@r107-ws15:~/git/webseite$ touch index.html
sbel@r107-ws15:~/git/webseite$ mkdir css
sbel@r107-ws15:~/git/webseite$ mkdir img
sbel@r107-ws15:~/git/webseite$ ls
css  img  index.html
```

- Initialisisere das Verzeichnise `webseite` als git-Repository.
- Lasse dir den Status des Repos anzeigen

- Füge die Datei und die beiden Verzeichnisse dem Index hinzu und erstelle einen ersten Commit. Untersuche den Status deines Repos. Welche Beobachtung machst du hinsichtlich der beiden Verzeichnisse?

Erstelle nun im Verzeichnis `css` eine Datei `style.css` mit dem folgenden Inhalt:

```
body {
  color: #666;
}

h1 {
 color: green;
 text-decoration: underline;
}
```

Füge außerdem in die Datei `index.html` den folgenden Inhalt ein:

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="css/style.css"
media="screen" />
    <title>Superwebseite!</title>
  </head>
  <body>
   <h1>Meine erste Webseite!</h1>
  </body>
</html>
```

Untersuche jetzt den Zustand deines Repos.

Erstelle weiteren Commit, der die letzten Änderungen enthält. Was ist hierfür der Reihe nach zu tun?

Ändere weitere Teile deiner Webseite. Erstelle jeweils an sinnvollen Stellen weitere Commits mit entsprechenden Commit-Messages.

Betrachte die Ausgabe des Befehls `git log`.