Git Cheatsheet

Git ist eine Versions-Verwaltung für Dateien und Software-Dokumente und wird von Entwicklern genutzt, bei der Software Entwicklung den Datei-Verlauf zu verwalten. Auch in vielen `static sites` wird git eingesetzt.

git installieren

Ubuntu

```
sudo add-apt-repository ppa:git-core/ppa
sudo apt update && sudo apt install -y git
```

Windows

1. Git von der Webseite herunterladen
2. Installation durchführen

Ein Repository erstellen

Will man ein Repository mit git erstellen hat man dabei primär 2 Optionen:

- man erstellt ein neues Repo
- man nutzt ein vorhandenes aus einer Remote Verwaltung

git init - ein neues repo erstellen

Erstellt ein neues Repo in einem Ordner.

```
$ mkdir my-project
$ cd my-project
$ git init
```

Hat man bereits einen Ordner mit Dateien, kann man diesen ebenfalls nutzen:

```
$ cd /path/to/my-project
$ git init
```

In dem Ordner wurde ein Verzeichnis `.git` erstellt.

zufügen eines remote Host für die git Verwaltung

Um das Git Repo über mehrere PCs zu verwenden kann man eine Remote Verwaltung zufügen. Das kann zum Beispiel ein Drittanbieter sein wie Bitbucket oder Github, man kann aber auch einen eigenen Git-Server

betreiben mit Gitolite oder (wenn man mehr als nur Repos hosten will) Gitea.

```
$ git remote add origin git@host.de:path/to/ui.git
```

Die URL am Ende des Commands ist dabei die URL zum Git Repo. Man kann hier sowohl `SSH` einsetzen (wie im Beispiel) als auch `https`

```
$ git remote add origin https://bitbucket.org/path/to/ui.git
```

bare

Ein "nacktes" Repository, das mit `git init --bare` erstellt wurde, ist Beispielsweise für das Teilen von Dateien. Wenn man mit einem Team von Entwicklern zusammenarbeitet und einen Ort benötigt, an dem man Änderungen an einem Repository freigeben kann, sollte man ein nacktes Repository an einem zentralen Ort erstellen, an dem alle Benutzer ihre Änderungen übertragen können (oft ist die einfache Wahl github).

Da git ein verteiltes Versionskontrollsystem ist, wird niemand direkt Dateien im freigegebenen zentralen Repository bearbeiten. Stattdessen klonen Entwickler das freigegebene `bare` Repository, nehmen Änderungen lokal in ihren Arbeitskopien des Repository vor und schieben die geänderten Dateien dann zurück zum freigegebenen blanken Repository, um ihre Änderungen anderen Benutzern zur Verfügung zu stellen.

Bare repos werden damit überwiegend auf Git Servern eingesetzt.

```
$ mkdir my-bare-project
$ cd my-bare-project
$ git init --bare
```

git clone - ein vorhandenes Repo nutzen

Nicht immer will man ein neues Repo erstellen, sondern auch vorhandene holen. Dafür stellt git den Command `clone` bereit.

```
$ git clone https://bitbucket.org/path/to/ui.git ui
```

Dabei wird das Remote Repo in den Ordner ui geklont. Will man den aktuellen Ordner verwenden, kann man das `ui` am Ende auch weglassen.

Git clone ist im Prinzip der schnellere Ersatz für

```
$ git init
$ git remote add origin https://bitbucket.org/path/to/ui.git
$ git pull origin master
```

git add

Fügt eine einzelne Datei, ein Verzeichnis, mehrere Dateien/Verzeichnisse oder alle Änderungen hinzu.

Hinzufügen aller Dateien

```
git add .
```

Hinzufügen einer Datei

```
git add pfad/zu/file.php
```

Hinzufügen eines Verzeichnisses

```
git add pfad/zu/verzeichnis
```

Optionen

Testen ob das Hinzufügen funktioniert

Einen Probelauf um auf eventuelle Fehler für add zu testen kann man mit der Option `--dry-run` oder kurz `-n` ausführen. Durch den Probelauf wird vorerst nichts geändert/hinzugefügt. Sollten Fehlermeldungen ausgegeben werden, hat man die Möglichkeit, diese erst zu beheben, bevor man im Nachhinein die Fehler mühselig Rückgängig machen muss.

```
git add . --dry-run
```

Entfernen von Dateien/Verzeichnissen aus dem Repo

Entfernen einer Datei aus dem Git Repo

Um eine Datei aus dem Git Repo zu löschen kann man den Command

```
git rm path/to/file.ext
```

verwenden. Der Pfad kann dabei relativ oder absolut sein. Will man die Datei selbst lokal behalten, hilft einem die Option `--cached`.

```
git rm --cached path/to/file
```

Damit wird die Datei aus dem index entfernt, selbst aber nicht gelöscht.

Entfernen eines Verzeichnisses

```
git rm -r --cached path/to/dir-to-delete
```

git commit

Übermitteln von Änderungen an die Versions-Kontrolle. Ein Commit erfordert, dass zuvor die geänderten Dateien des commit mit `git add` oder `git rm` hinzugefügt/zum löschen Markiert wurden.

```
git commit -m "Commit Nachricht kommt hier..."
```

Optionen für die Commit Beschreibung

-m *[message]*
--message *[message]*

Setzen der Commit Beschreibung. Wenn mehrere -m Optionen angegeben wurden, werden deren Werte als separate Paragraphen kombiniert.

Beispiel:

```
git commit -m"Commit Beschreibung hier..."
```

-F *[filename]*
--file *[filename]*

Lesen der Commit Beschreibung aus der angegebenen Datei.

Beispiel:

```
git commit -F /home/me/current-commit.txt
```

--author *[author]*

Überschreibt den Autor eines Commits auf einen bestimmten Namen

Beispiel:

```
git commit  -m"Commit Beschreibung hier..."
            --author "Karl Gustav"
```

--date *[datum]*

Überschreibt das Datum eines Commits auf einen bestimmten Zeitpunkt. Mehr über die unterstützten Formate findest du hier.

Beispiel:

```
git commit  -m "Commit Beschreibung hier..."
            --date "2017.12.27"
```

git reset

Tool zum Zurücksetzen von Änderungen im working tree oder auch ganzer Commits.

Zurücksetzen aller lokalen Änderungen

```
# Zurücksetzen aller veränderten Dateien
git reset --hard
# Entfernen aller nicht versionierten Dateien und Verzeichnisse,
# die noch nicht via git add zugefügt wurden
git clean -fd
# Pull um aktuelle Commits aus dem Remote zu holen
git pull origin master
```

Zurücksetzen eines Commit, der noch nicht via push übertragen wurde

```
# Zurücksetzen aller verändeten Dateien im Commit auf den vorigen Status
git reset --hard HEAD~1
# Entfernen aller nicht versionierten Dateien und Verzeichnisse,
# die noch nicht via git add zugefügt wurden
git clean -fd
# Pull um aktuelle Commits aus dem Remote zu holen
git pull origin master
```

Zurücksetzen mehrere Commits

Um mehrere Commits rückgängig zu machen, kann man die gleiche Prozedur durchziehen wie im vorigen Punkt. Einzige Änderung wäre, dass man anstelle von `HEAD-1` den Commit angibt:

```
git reset --hard 4564150952
```

`4564150952` entspricht dabei den ersten Ziffern des Commit.

Zur Erinnerung: die Commits kann man sich mit Hilfe von

```
git log --pretty=oneline
```

auflisten lassen.

git tag

> Git Tags sind im Prinzip Versions-Punkte in der Entwicklung, die z.B. als Release gepackt werden. Tags können direkt ausgecheckt werden.

einen Tag erstellen

```
# Erstellt den Tag v1.1.0 mit entsprechender Commit Message
git tag -a v1.1.0 -m "Tagging Version v1.1.0"
# Übermittelt den Tag
git push origin v1.1.0
```

Auf einem anderen PC kann man dann die Tags holen mit

```
git pull origin --tags
```

einen Tag basierend auf einem Commit erstellen

Zuerst kann man sich die Commit Nachrichten über

```
git log --pretty=oneline
```

tag checkout

Tags kann man via checkout vom Remote Host holen:

```
git checkout v1.1.0
```

Auflisten vorhandener Tags

Um alle Tags aufzulisten, kann man einfach

```
git tag
```

aufrufen. Will man jedoch alle Tags zu bestimmten Sub-Releases erhalten, kann man die durch

```
git tag -l "v1.1.*"
```

bekommen.

Zu einem Major Release könnte der Command wie folgt aussehen:

```
git tag -l "v1.*"
```

git submodule

Hier habe ich ein separates Cheatsheet für Submodule.

git subtree

Hier habe ich ein separates Cheatsheet für Subtree.

Zusatz-Informationen

Git Datum Format

Die `GIT_AUTHOR_DATE`, `GIT_COMMITTER_DATE` Umgebungs-Variablen sowie die `--date` Option unterstützen die folgenden Datum Formate:

1. Git internal format
   Das interne Format setzt sich aus `<unix timestamp> <time zone offset>` zusammen. Dabei ist `<unix timestamp>` die Anzahl an Sekunden seit der UNIX-Epoche. `<time zone offset>` ist ein positives oder negatives Offset von der `UTC` Zeitzone. z.B. `MESZ` (diese hat 2 Stunden mehr als `UTC`) is `+0200`.
2. RFC 2822
   Das Standard E-Mail Format wie es in der **RFC 2822** beschrieben wird, z.B. `Thu, 07 Apr 2020 22:13:13 +0200`.
3. ISO 8601 Zeit und Datum wie es in der **ISO 8601** festgelegt ist, z.B. 2020-04-07T22:13:13. Der Parser akzeptiert auch ein Leerzeichen (space) anstelle des `T` Character.

> **Beachte**:
>
> Zusätzlich wird das Datum in den folgenden Formaten akzeptiert:
>
> - `YYYY.MM.DD`
> - `MM/DD/YYYY`
> - `DD.MM.YYYY`