Git Basics

In diesem Cheat Sheet werden die notwendigen grundlegenden Konzepte hinter *git* anhand der CLI-Kommandos aufgeführt und erläutert.

think tecture

Repository initialisieren

Beim Erstellen eines neuen Projektes kann man direkt damit anfangen, ein *git*-Repository zu initialisieren und alle lokalen Features der Versionsverwaltung einsetzen.

```
$ git init
```

Um dieses Repository mit einem Remote zu verbinden und um die Files zu pushen, helfen folgende Befehle:

```
$ git remote add <name> <url>
$ git push -u <remote> <branch>
```

Ein existierendes Repository klonen

In einem existierenden Projekt, in dem schon git eingesetzt wird, oder im Falle, dass schon ein Repository besteht, kann dieses einfach geklont werden:

```
$ git clone <url> [<folder>]
```

Der optionale `folder`-Parameter nimmt den Zielpfad entgegen, in dem die Working-Copy angelegt werden soll.

Staging Files

Staging bezeichnet den Vorgang, in dem man die Änderungen, die man gerne in einem Commit festhalten möchte, markiert.

FILES HINZUFÜGEN

Um alle **existierenden** Files, die einem gegebenen `pattern` entsprechen, hinzuzufügen, bietet sich folgende Form an:

```
$ git add <pattern>
```

ALLE ÄNDERUNGEN HINZUFÜGEN

Um einfach alle Änderungen, sprich, ungetrackte Files, modifizierte Files sowie auch Löschungen zu stagen, gibt es den `--all` flag:

```
$ git add --all
```

Commit / Status / Diff

```
$ git status
```

Zeigt Informationen über den aktuellen Zustand des Repositories an.

```
$ git commit -m "<msg>"
```

Schreibt die gestageten Änderungen in die History mit der gegebenen `msg` als Nachricht.

```
$ git commit -am "<msg>"
```

Das `-a` Argument staged alle getrackten Files und committed.

```
$ git diff
$ git diff --staged
```

Zeigt die nicht gestageten Änderungen. Das `--staged`-Argument zeigt hingegen die gestageten Änderungen.

```
$ git diff --word-diff
```

Das `--word-diff`-Argument hebt mehrere Änderungen in der selben Zeile genauer hervor.

```
$ git diff <branch>
```

Zeigt die Änderungen zwischen dem aktuell ausgecheckten Commit (`HEAD`) und dem gegebenen `branch`.

```
$ git checkout <SHA|branch> -- <file>
```

Checkt ein `file` aus im Zustand des gegebenen `SHA`-Hash oder `branch`.

Branches

Ein Branch ist eine benamte Referenz auf ein Commit. Branches können sehr liberal verwendet werden und ermöglichen zwischen verschiedenen Zuständen des Quellcodes zu wechseln.

```
$ git branch <name>
```

Erstellt einen neuen Branch am aktuell ausgecheckten Commit (`HEAD`).

```
$ git checkout <branch>
```

Wechselt den Branch. Hierbei kann es zu Konflikten kommen, wenn Files mit Änderungen auch auf dem Ziel-Branch verwendet werden. Daher zuerst commiten (oder stashen, siehe weiter unten).

```
$ git checkout -b <name>
```

Erstellt einen neuen Branch am aktuell ausgecheckten Commit und wechselt auf den neu erstellten Branch mit dem gegebenen `name`.

```
$ git branch -d <branch>
```

Löscht einen lokalen Branch.

```
$ git merge --no-ff <branch>
```

Reintegriert `branch` in den aktuell ausgecheckten Branch. Das `--no-ff`-Argument unterbindet einen Fast-Forward-Merge und erzeugt einen Merge-Commit, sodass in der History die Existenz des Branches weiterhin nachvollzogen werden kann.

think tecture

RESET

```
$ git reset --hard <SHA|branch>
```

Bewegt den aktuellen Branch-Pointer auf den gegebenen `SHA`-Hash oder an die Stelle von `branch`. Das `--hard`-Argument verwirft dabei alle getrackten Änderungen.

```
$ git reset --soft <SHA|branch>
```

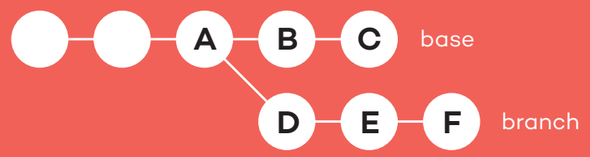
Bewegt den aktuellen Branch-Pointer auf den gegebenen `SHA`-Hash oder an die Stelle von `branch`. Das `--soft`-Argument behält dabei die Differenz in Änderungen gestaged bei.

REBASE

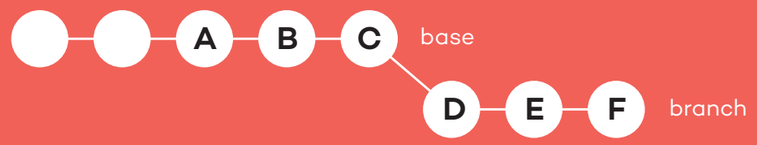
```
$ git rebase <base> <branch>
```

Rebase nimmt die Commits von `branch` und wendet sie auf der `base` wieder an, was darin resultiert, dass der letzte Commit von `base` die neue Basis von `branch` wird.

VOR DEM REBASE



NACH DEM REBASE



Log

Der `log`-Befehl, stellt die Möglichkeit bereit, die *git*-History in verschiedenen Formen darzustellen.

```
$ git log
```

Zeigt die History des aktuellen Branches in einer scrollbaren Ansicht an.

```
$ git log --oneline
```

Das `--oneline`-Argument reduziert die Informationen eines Commits auf den `SHA`-Hash und die Commit-Nachricht.

```
$ git log --graph --oneline
```

Das `--graph`-Argument zeigt die History als Baum an.

```
$ git log --all --decorate --oneline --graph
```

Die Argumente `--all` und `--decorate` zeigen den Graph für alle Branches an und markieren alle Referenzen, wie z.B. Branch-Namen der sichtbaren Commits.

Remote-Interaktionen

Remote-Repositories sind Versionen eines Projekts, die im Internet oder im Netzwerk gehostet werden (bspw. bei *GitHub*).

```
$ git remote add <name> <url>
```

Fügt dem Repository ein Remote unter der gegebenen `url` mit dem Namen `name` zu.

```
$ git push -u <remote> <branch>
```

Pusht die Commits des aktuellen Branches auf das gegebene `remote` und stellt eine Tracking-Verbindung zwischen dem lokalen und dem Remote-Branch her.

```
$ git push
```

Pusht die Commits des aktuellen Branches auf den getrackten Remote-Branch.

```
$ git fetch
```

Lädt den aktuellen Zustand des Remotes herunter.

```
$ git push <remote> :<branch>
$ git push <remote> --delete <branch>
```

Löscht den gegebenen `branch` auf dem `remote`.

Stash

Stashen ist ein *git*-Feature und entspricht einem Commit auf einem temporären Branch, mit bestimmtem Namen, sodass der *git*-Client diese identifizieren kann. Stashen eignet sich, um Änderungen kurz auf Seite zu legen und sie später wiederzuholen, wie z.B. vor einem konfliktbehafteten Branch-Wechsel.

```
$ git stash
```

Legt die aktuellen Änderungen auf den Stash.

```
$ git stash list
```

Zeigt eine Liste aller gestashten Commits an.

```
$ git stash pop
```

Nimmt den letzten Commit vom Stash herunter und wendet ihn auf den aktuell ausgecheckten Zustand an.

```
$ git stash apply
```

Wendet den letzten Commit vom Stash auf den aktuell ausgecheckten Zustand an. Der Commit bleibt auf dem Stash erhalten.

```
$ git stash drop
```

Entfernt den letzten Commit vom Stash ohne ihn anzuwenden.

Help

```
$ git help <command>
```

Zeigt die *git*-Hilfe zu dem gegebenen `command` an.