ERSTELLEN

Neues Repository erstellen

```
$ git init
```

Erstellt ein neues lokales Git Repository im aktuellen Ordner.

Vorhandenes Repository klonen

```
$ git clone *URL*
```

Klont den Inhalt eines Remote Repositories in das aktuelle Verzeichnis.

LOKALE ÄNDERUNGEN

Aktuellen Status anzeigen

```
$ git status
```

Zeigt alle lokalen Änderungen im aktuellen Repository an (entspricht *Pending Changes* im TFVC). Weiterhin werden z.B. bei Merge-Konflikten nützliche Hilfe-Hinweise für mögliche weitere Kommandos angezeigt.

Dateidifferenzen anzeigen

```
$ git diff
```

Zeigt alle Änderungen bei bereits getrackten Dateien an (Dateiinhalt)

```
$ git difftool
```

Öffnet ein grafisches Tool um Änderungen anzuzeigen

Änderungen stashen

```
$ git stash
```

Alle geänderten Dateien werden gestasht und das Arbeitsverzeichnis bereinigt (Arbeitsstand wechselt zum letzten Commit)

```
$ git stash save -u *Stash-Name*
```

Alle geänderten Dateien werden gestasht unter dem Namen *Stash-Name* und das Arbeitsverzeichnis bereinigt (Arbeitsstand wechselt zum letzten Commit)

```
$ git stash list
```

Listet alle Stash-Einträge auf

```
$ git stash apply stash@{1}
```

Gewählter Arbeitsstand aus dem Stash wird in das Arbeitsverzeichnis übernommen und bleibt im Stash erhalten.

```
$ git stash pop
```

Zuletzt abgelegter Arbeitsstand wird in das Arbeitsverzeichnis übernommen und aus dem Stash gelöscht. Bei Konflikten bleibt der Eintrag im Stash erhalten.

Dateien für den Commit vorbereiten (Staging)

```
$ git add *Pfad/zu/Datei*
```

Datei zum Committen markieren.

```
$ git add .
```

Alle Dateien ab dem aktuellen Verzeichnis (inkl. aller Unterordner) zum Committen markieren.

Dateien aus dem nächsten Commit nehmen (Unstaging)

```
$ git reset
```

Dateien vom Status "Bereit für Commit" entfernen (Änderungen bleiben dabei erhalten)

Committen

```
$ git commit [-m *Kommentar*]
```

Alle zum Committen markierte Dateien Committen

Ohne den Parameter **-m** geht der konfigurierte Editor auf, der den Kommentar entgegennimmt und nach dem Schließen des Editors wird committed.

Mit dem Parameter **-m** gibt man den Kommentar direkt auf der Kommandozeile an (bei vorhandenen Leerzeichen mit Anführungsstrichen umschließen)

Zusätzliche Optionen:

```
--amend
```

Aktuelle Änderungen in den letzten Commit mit integrieren

NIEMALS BEREITS VERÖFFENTLICHTE COMMITS AMENDEN

HISTORY ANZEIGEN

```
$ git log
```

History des gesamten Repository anzeigen

```
$ git log *Pfad/zu/Datei*
```

History der Datei anzeigen

History einer Datei anzeigen

```
$ git blame *Pfad/zu/Datei*
```

Alle Änderungen in einer Datei anzeigen, dabei wird pro Zeile angezeigt, welche Änderung von wem committed wurde (wie *Annotate* im TFVC)

BRANCHES & TAGS

Branch erstellen

```
$ git branch *Branchname*
```

Branch mit dem Namen *Branchname* anlegen

Branch wechseln

```
$ git checkout *Branchname*
```

Zum Branch *Branchname* wechseln

Tag erstellen

```
$ git tag -a *Name*
```

Tag mit Namen und Beschreibung am aktuellen Commit anlegen

UPDATE & PUBLISH

Änderungen von einem Remote Repository holen

```
$ git fetch
```

Alle Änderungen des Remote Repository in das lokale Repository laden, dabei wird aber das Arbeitsverzeichnis nicht geändert

Änderungen von einem Remote Repository holen und gleichzeitig mergen

```
$ git pull
```

Alle Änderungen des Remote Repository in das lokale Repository laden und gleichzeitig die Arbeitskopie aktualisieren (Erzeugt ggf. eine Merge-Commit)

Möglichst vermeiden. Bitte stattdessen den Workflow "*Änderungen vom zentralen Repository integrieren*" verwenden.

Änderungen an ein Remote Repository übertragen

```
$ git push
```

Tags an ein Remote Repository übertragen

```
$ git push --tags
```

MERGE & REBASE

Mergen

```
$ git merge *Branch*
```

Den angegebenen *Branch* in den aktuellen Branch mergen, dabei entscheidet Git selbst, ob es rekursiv oder mittels fast-forward geschieht

Zusätzliche Optionen:

```
--ff-only
```

Nur fast-forward zulassen

```
--no-ff
```

Erzeugt immer einen Mergecommit

Mergekonflikte lösen

```
$ git mergetool
```

Öffnet das konfigurierte Standard Mergetool, um die Merge Konflikte aufzulösen.

```
$ git mergetool --tool=kdiff3
```

Öffnet das angegebene Mergetool, um die Merge Konflikte aufzulösen.

```
$ git checkout --conflict=merge
```

Bei fehlerhaftem Merge-Versuch, werden die Merge-Marker wieder in die konfliktbehafteten Dateien eingefügt und Merge-Konflikte können erneut aufgelöst werden.

Rebase

```
$ git rebase
```

Commit History neu schreiben, v.a. hilfreich, wenn beim fetch & merge eines Remote Branches kein Fast Forward möglich ist

UNDO

Änderungen rückgängig machen

```
$ git checkout .
```

Alle Änderungen an den Dateien rückgängig machen, die **nicht gestaged** sind (betrifft keine neu erstellten Dateien)

WORKFLOWS

Änderungen vom zentralen Repository integrieren

1. `git fetch`
2. `git rebase`
3. Merge-Konflikte?
    1. `git mergetool`
    2. `git rebase --continue`

(Abbruch des rebase-Vorganges jederzeit möglich mit `git rebase --abort`)

WEITERFÜHRENDE LINKS

https://www.ralfebert.de/git/
http://tklepzig.github.io/git-vs-pres/
http://tklepzig.github.io/git-branching-pres/