

Grundbefehle

| | |
|--|--|
| <code>git clone <url></code> | Erzeugt eine lokale Kopie eines Git Repositories (es werden keine Schreibrechte benötigt) |
| <code>git status</code> | gibt den aktuellen Zustand der Working Copy aus |
| <code>git commit -a</code> | Fügt alle lokalen Änderungen dem Repository hinzu (ohne Übertragung an den Remote Host) |
| <code>git reset --hard origin/<branch-name></code> | Setzt local auf den HEAD Stand zurück. Lokale commits werden verworfen. |
| <code>git clean -f -d</code> | Löscht alle temporären Dateien und Folder z.B. Files die nicht unter git Kontroller stehen |
| <code>git push</code> | Veröffentlichung der lokalen Commits auf dem Remote Repository |
| <code>git branch</code> | Liste alle lokal bekannten Branches auf |
| <code>git branch <branch-name></code> | Erstellt lokal einen neuen Branch, bleibt aber auf dem Alten |
| <code>git branch -D <branch-name></code> | Löscht einen Branch unabhängig davon ob er im upstream existiert |
| <code>git push origin --delete <branch></code> | Löscht auch den Remote Branch erfolgreich |
| <code>git checkout <branch-name></code> | Wechselt zu einem anderen Branch |
| <code>git tag --delete <tagname></code> | Löscht den Tag lokal |
| <code>git push --delete origin <tagname></code> | Löscht den Tag im Remote Repo |
| <code>git stash</code> | Lokale Änderungen als Backup auf einen internen Stack legen. |
| <code>git stash pop</code> | Lokale Änderungen aus dem Stack wiederholen |
| <code>git remote add upstream <url></code> | Alias <i>upstream</i> zum master eines Remote Repos definieren |
| <code>git remote add --track <branch-name> upstream <url></code> | Alias <i>upstream</i> zum Branch mit Repo URL definieren |
| <code>git fetch upstream</code> | Aktuellen Stand vom Alias <i>upstream</i> herunterladen |
| <code>git merge <branch> - -no-commit - -no-ff</code> | <branch> wird in den aktuell ausgecheckten branch gemerged (ohne commit) |
| <code>git merge upstream/master</code> | Merge der Änderungen vom Alias <i>upstream</i> branch "master" in den lokalen branch wobei für konfliktfreie Änderungen ein Autocommit erfolgt. |
| <code>git merge upstream/master - -no-commit - -no-ff</code> | Merge der Änderungen vom Alias <i>upstream</i> branch "master" in den lokalen branch wobei für konfliktfreie Änderungen KEIN Autocommit erfolgt. |
| <code>git clone --mirror <reporurl></code> | Spiegelt ein Repository und kann zur Erstellung eines lokalen Backups verwendet werden. |
| <code>git remote update</code> | In einem Backup die remote Updates einspielen |



By **Huluvu424242**
(FunThomas424242)

Published 6th April, 2015.
Last updated 29th April, 2019.
Page 1 of 3.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish
Yours!
<https://apollopad.com>

cheatography.com/funthomas424242/
github.com/Huluvu424242

Grundbefehle (cont)

| | |
|--|---|
| <code>git push <remotename> <commit SHA>:<remotebranchname></code> | Push eines ausgewählten lokalen commits. z.B. <code>git push origin 712acff81033eddc90bb2-b45e1e4cd031fetc50f:master</code> |
| <code>git branch -m new-name</code> | Aktuellen lokalen Branch in new-name umbenennen |
| <code>git branch -m old-name new-name</code> | Einen lokalen Branch old-name in new-name umbenennen |
| <code>git push origin :old-name new-name</code> | Erst Prüfen!: Alten remote Branch löschen und den neuen lokalen Branch pushen |
| <code>git push origin -u new-name</code> | Aktuellen Branch im Remote umbenennen |

Alle lokalen Änderungen müssen um im Remote Repository sichtbar zu werden über den Mechanismus commit und push in das Remote Repository eingetragen werden (auch neu angelegte Branches).

Github - Pull Requests aktuell halten

| | |
|---|--|
| <code>git clone <url></code> | Lokale Arbeitskopie erstellen |
| <code>git checkout <branch-name></code> | Auf den Pull Request wechseln |
| <code>git remote add upstream git://github.com/owner/projectname.git</code> | Alias <i>upstream</i> für den Zugriff auf den Master definieren |
| <code>git fetch upstream</code> | Die Änderungen vom <i>master</i> herunterladen |
| <code>git merge upstream/master</code> | Den <i>master</i> in den lokalen Arbeitsstand mergen |
| <code>git commit -a</code> | Alles commiten wenn keine Konflikte mehr bestehen |
| <code>git push</code> | Nach dem erfolgreichen lokalen bauen, den aktuellen Stand veröffentlichen. |

Generell sollte für jedes neue zu realisierende Feature ein eigener Feature Branch vom *master* gezogen werden.

Hat man das Feature erfolgreich implementiert kann man auf github.com einen pull Request stellen.

Wenn dieser Pull Request vom Owner nicht gemerged werden kann weil es Änderungen auf dem *master* gab dann hilft das oben beschriebene Vorgehen.

Neuen Branch mit lokalen Änderungen erstellen

| | |
|---|--|
| <code>git status</code> | Gibt den aktuellen Branch aus und zeigt welche lokalen Änderungen vorliegen |
| <code>git stash</code> | Lokale Änderungen auf den Stack legen |
| <code>git branch <neuer-branch-name></code> | Neuen Branch anlegen, Abspaltung vom Stand des aktuellen Branches |
| <code>git checkout <neuer-branch-name></code> | Wechseln auf den neuen Branch |
| <code>git stash pop</code> | Lokale Änderungen vom Stack in den neuen Branch laden |
| <code>git commit -a</code> | Alle Änderungen auf den neuen Branch commiten |
| <code>git push</code> | Nach erfolgreichem lokalen Bauen die Änderungen und den neuen Branch an das Remote Repository publizieren. |

Es gibt Tage das will man schnell mal was ausprobieren und dann funktioniert es gleich - unglaublich. Aber natürlich ist man mit den lokalen Änderungen gerade auf dem frisch ausgecheckten *master* und will die Änderungen aber als neuen Feature Branch commiten - was tun? Das was oben steht ;)



By [Huluvu424242](#)
(FunThomas424242)

Published 6th April, 2015.
Last updated 29th April, 2019.
Page 2 of 3.

Sponsored by [ApolloPad.com](#)
Everyone has a novel in them. Finish Yours!

<https://apollopod.com>

Git für Präsentationen

Wer in einer Präsentation zum nächsten oder vorherigen Stand des Programmes springen möchte ohne das Risiko von Schreibfehlern zu riskieren der sollte sich die Git Aliase prev und next anlegen. Damit lässt sich zum vorhergehenden oder nächsten Commit wechseln.

(Quelle: <https://coderwall.com/p/ok-iyg/git-prev-next>)

Die Aliase werden in der ~/.gitconfig eingetragen

```
[alias]
```

```
prev = checkout HEAD^1
```

```
next = "!sh -c 'git log --reverse --pretty=%H master | awk \"/>${git rev-parse HEAD}/{getline;print}\" | xargs git checkout"
```



By **Huluvu424242**
(FunThomas424242)

Published 6th April, 2015.

Last updated 29th April, 2019.

Page 3 of 3.

Sponsored by **ApolloPad.com**

Everyone has a novel in them. Finish
Yours!

<https://apollopad.com>

cheatography.com/funthomas424242/

github.com/Huluvu424242