|  |  |  |
| --- | --- | --- |
| ls  ls -l  ls  -lh  ls -lah ls <Ordner> | Alle Dateien & Ordner im aktuellen Ordner anzeigen  Alle Dateien & Ordner im aktuellen Ordner anzeigen (Detail)  Alle Dateien & Ordner im aktuellen Ordner anzeigen (Detail mit Grössenangabe)  Alle Dateien & Ordner im aktuellen Ordner anzeigen (Detail mit Grössenangabe) - auch versteckte  Alle Dateien & Ordner im <Ordner> anzeigen (kann natürlich mit den Optionen kombiniert werden | FS |
| ls -al | grep <Suchstring> | Listet alle Dateien & Ordner im aktuellen Verzeichnis die den <Suchstring> enthalten | FS |
| ls -R | grep '^IMG\_[36][0-9]\*.jpg' | Listet alle Dateien im aktuellen Verzeichnis (ls) oder darin enthaltenen Unterverzeichnissen (-R), deren Dateinamen mit der Zeichenkette IMG\_ anfgangen(^), dann eine 3 oder eine 6 enthalten ([36]), gefolgt von beliebig vielen (\*) Zahlen ([0-9]) und die mit .jpg enden. | FS |
| mkdir | Ordner erstellen | CREATE |
| rmdir | Ordner löschen | DELETE |
| pwd | Aktuellen Pfad anzeigen | FS |
| cp ~/Documents/Artikel.txt ~/Desktop/  cp <filename.ext> ./<subfoldername>  cp -R <folder> ./<subfolder>  cp -R <folder> ../ | Datei aus den Dokumenten auf den Desktop kopieren  Datei aus aktuellem Verzeichnis in Unterverzeichnis kopieren  Ordner in Subfolder kopieren  Ordner in Parent Folder kopieren | COPY |
| mv | Datei verschieben (handling analog copy nur das -R nicht mehr benötigt wird aktuell) | MOVE |
| mv \*.jpg ./<Ordnername> | Verschiebt alle Dateien im aktuellen Ordner die die Endung jpg haben in das angegebene Unterverzeichnis | MOVE |
| mv ?.jpg ./<Ordnername> | Verschiebt alle Dateien mit exakt einem Zeichen vor dem Punkt in den angegebenen Unterordner  z.B.: 'a.jpg', '1.jpg' aber **nicht** 'ErstesFoto.jpg' etc. | MOVE |
| mv [2-5].jpg ./<Ordnername> | Verschiebt die Dateien 2.jpg, 3.jpg, 4.jpg, 5.jpg in das angegebene Unterverzeichnis nicht jedoch 1.jpg oder 6.jpg etc. | MOVE |
| cd | Ordner wechseln | FS |
| touch <dateiname.ext> | Erstellt neue Datei des Typs | FILE |
| vi <dateiname.ext> | Öffnet Datei im entsprechenden Editor | FILE |
| rm <filename.ext> | Löscht Datei | DELETE |
| rm -rf <foldername> | Löscht Ordner und alle Inhalte | DELETE |
| Open <filename.ext> | Öffnet die Datei mit registrierter Applikation | FILE |
| xdg-open (Linux) | Öffnet die Datei mit registrierter Applikation | FILE |
| find . -name "\*.txt" | Findet alle Dateien des Typs txt im aktuellen Verzeichnis und Unterverzeichnissen | FS |
| <filename> -delete | Löscht angegebene Datei(en) unwiderruflich - kann mit Suche angewendet werden: | DELETE |
| find . -name "\*.txt" -delete | Löscht alle Dateien des Typs txt in diesem und aller Unterordner - zuerst ohne Delete nutzen für check | FS DELETE |
| cd "Mein Ordner mit Spaces"  cd Mein\ Ordner\ mit\ Spaces | Ordnername in Anführungszeichen damit Leerzeichen im Ordnernamen gelesen werden können Leerzeichen werden mit vorangestelltem \ geschrieben: "\ " damit der Pfad korrekte identifiziert werden kann | FS |
| cd | Direkt zum User Heim Verzeichnis | FS |
| cd / | Direkt zum Root | FS |
| cd - | Wechsel zwischen den beiden zuletzt besuchten Verzeichnissen hin und her | FS |
| <arrow-up>  ALT + <arrow links / rechts> | Durch die zuletzt verwendeten befehle scrollen  Durch die Wörter springen | HANDLING |
| CTRL+a  CTRL+e | An den Anfang der Zeile Springen  Ans Ende der Zeile Springen | HANDLING |
| CTRL+r | Suche nach / in den zuletzt verwendeten befehlen | HANDLING |
| <Teilbefehl>+TAB  TAB | Zeigt Auswahl verfügbarer Befehle oder setzt direkt einzig gültige Option Circle durch verfügbare Befehle (wenn mehrere zur Verfügung stehen) | HANDLING |

**git-bash Befehle**

|  |  |  |
| --- | --- | --- |
| git init | auf dem lokalen Ordner git initiieren | initial |
| git clone <repo> <dir> | Kopie eines Repos in ein lokales Verzeichnis erstellen | initial |
| git pull <remote> or <repo> | fetch-merge Befehl für ein Repo | Updates ziehen |
| git commit -m « xxx » | Veränderte Files bestätigen und mit einer Nachricht versehen | Staging |
| git add <filename>  git add . | Datei in den Staging Bereich geben  alle markierten Files/Folder in den Staging geben | Staging |
| git reset <filename>  git reset –soft HEAD~xy | Die Datei aus dem Staging rausnehmen  Anpassen einer Commit Message |  |
| git stash  git stash pop | Die Arbeit absichern  Die Arbeit an der Sicherung fortsetzen |  |
| git push <repo> | die lokalen Changes an das Repo senden | Change in das Repo senden |
| git log  git log –oneline | Das Log von git aufrufen und die Changes sehen | Log |
| git status | View für die geänderten Files welche gestaged oder noch nicht gestaged wurden | Log |
| git branch  git branch <name>  git branch -b <name> | Status der Branches sehen  Erstellen des Branches mit <name>  Branch mit <name> erstellen und einchecken | Verzweigung |
| git checkout <branch> | Auf einen Branch <branch> einchecken | Verzweigung |
| git merge | Zusammenführen der Branches | Verzweigung |
| git rebase <branch> | Die Basis des Branches versetzen auf einen anderen Master commit | Verzweigung |
| git remote  git remote -v  git remote add <name> <url>  git remote set-url <name> <url> | die verschiedenen Anker für URL’s abrufen  Die Anker mit URL ausgeben  Remote mit Name und URL erstellen  Remote <name> mit URL ändern | Remote |
| Working Directory | Das Arbeitsverzeichnis (Lokal) |  |
| Index (Staging Area) | Die Changes welche beim nächsten Commit erfasst werden |  |
| Repository | Der Object Store Datenbank von git |  |
| Commit | Verweis auf einen einzigen Zustand des Object Stores |  |
| git fetch | Download von Objekten und Referenzen von anderen Repos |  |
| git revert  git revert HEAD~xy | Rückgänging machen eines commits |  |
| git cherry-pick | Einen commit aus einem anderen Branch auf dem eingenen Branch anwenden |  |

## git Workflow Zentrales Repo

* Nur ein Branch

Vorteil:

* keine Branches

Nachteile:

* bei jedem push muss potentiel zuerst gemerged werden

## git Workflow – Feature Branches I

* Neue Entwicklungen pasieren auf Feature Branches
* master wird lokal integriert
* Erst wenn alles funktioniert, wird auf master gemerged
* Gleichzeitiges, unabhängiges entwickeln

Vorteile

* master Branch ist immer sauber
* push ohne vorher zu mergen
* Feature kann vor dem merge getestet werden

Nachteil

* umg mit Branches unerlässlich

## git Workflow – git Flow (erweiterter Feature Branch Workflow)

* für langlebige Projekte mit Releaszyklen

Vorteile

* Bugs in ausgelieferten Versionen fixen (Hotfixes, Backports)

Nachteile

* höchste Disziplin im Umgang mit Branches & Tags

## git Tags

Ein tag ist ein definierter Name für einen bestimmten Commit

* git tag v1.3
  + erstellt einen tag v1.3, der auf en aktuellen commit zeigt
* git tag -a v1.3 -m «version 1.3» 83ah7
  + erstellt einen tag v1.3, der auf den commit 83ah7 zeigt
  + zusätzlich wird eine Nachricht gesetzt und er Zeitpunkt wie auch Ersteller des tags wird gespeichert
  + genutzt für Release Markierungen
* git push origin –tags
* git pull origin –tags
* git checkout -b hotfix1 v1.2

## git flow

* master --> der Hauptbranch der nur die veröffentlichte Applikation beinhaltet
* hotfix --> ein hotfix/bugfix Branch der fürs bugfixing zu einem bestimmten commit/tag/release gemacht wird
* release --> der release Branch in dem die Entwicklungen für die Produktion freigegeben werden
* develop --> der develop Branch ist für die Entwicklung genutzte Branch, ab hier werden die Features abgebrancht
* feature --> der feature Branch wird für Testings und Weiterentwicklungen genutzt