Begrifflichkeiten

Kommandozeile: Ort (-> Kommando*zeile*) der Eingabe von Befehlen.

CLI: Steht für "command line interface" und ist mit der Kommandozeile gleichzusetzen.

Shell: Interpreter-Programm der eingegebenen Befehle. Anwendungen werden im Kontext dieses Interpreters ausgeführt, befinden sich also in einer Hülle (-> engl. *shell*). Innerhalb der Shell kann man z.B. Umgebungsvariablen ändern und zwar ohne, dass andere Programme/Shells das merken. Die erste Shell gabs in UNIX und ist heute meist unter `/bin/sh` zu finden.

Bash: Weiterentwicklung der Ur-Shell. Alternativen sind z.B. ZSH, KSH, Dash, Fish aber auch die PowerShell unter Windows kann man als Shell bezeichnen (hat "Shell" ja schließlich auch im Namen).

Terminal: Gaaaanz früher ein elektronisches Gerät an dem jemand saß und Befehle eingegeben hat. Wird heute als Gerätedatei (z.B. `/dev/tty...`) dargestellt, welche eingehende Daten an die entsprechende Shell weiterleiten.

Terminal-Emulator: Programm, welches das Terminal darstellt. Es kümmert sich also um Schriftart, Farben, Styles, Effekte, etc. Meist sagt man einfach "Terminal" und meint damit einen Terminal Emulator.

Navigation

Die Shell befindet sich immer in *einem* Ordner (wie man es vom normalen Datei-Explorer kennt).

- `cd pfad` oder `cd pfad/zu/einem/ordner` wechselt in den eingegebenen Ordner
- `cd` oder `cd ~` wechselt in das eigene Home-Verzeichnis (quasi der "Eigene Dateien" Ordner unter Linux).
  - `cd ~/ordner` wechselt in den Ordner `ordner` im Home-Verzeichnis
- `cd /` wechselt in den Root-Ordner (den aller obersten Ordner; vergleichbar mit `C:\` unter Windows)
- `pwd` steht für "print working directory" und zeigt den Ordner an in dem man sich gerade befindet

Dateien und Ordner

Dateien

Erstellen

- `touch datei.txt` erstellt man eine leere Datei `datei.txt` mit den Rechnten `rw-r--r--` (s.o. Beispiel zu `ls`).

Lesen

- `cat datei.txt` : Zeigt den kompletten Inhalt von `datei.txt` auf einmal an. Will man vielleicht nicht immer.

- `less datei.txt` : Zeigt den Inhalt an aber man kann blättern. Bei großen Dateien empfohlen.

Schreiben

- Operator `>` : Mit `echo "foo" > datei.txt` schreibt man den string `foo` in die Datei `datei.txt` und löscht dabei alles bestehende in der Datei
- Operator `>>` : Mit `echo "foo" >> datei.txt` hängt man den string `foo` am Ende der Datei an

Man kann Befehle verbinden: `cat datei.txt >> andere-datei.txt` hängt den Inhalt von `datei.txt` am Ende von `andere-datei.txt` an.

Kopieren

- `cp foo.txt anderer/ordner/bar.txt` kopiert die Datei `foo.txt` in den Ordner `anderer/Ordner/` wo sie dann `bar.txt` heißt
- `cp foo/* bar` kopiert alle Dateien aus dem Ordner `foo` in den Ordner `bar`
- `cp foo/*.pdf bar` kopiert nur PDFs von `foo` nach `bar`

Löschen

- `rm datei.txt` löscht die Datei `datei.txt` , wenn der Nutzer das Schreibrecht (`w`) auf der Datei hat

Ordner

Erstellen

- `mkdir mein-ordner` erstellt man einen leeren Ordner namens `mein-ordner` mit den Rechten `drwxr-xr-x` .

Inhalt auflisten

- `ls` listet alle Dateien und Ordner auf
  - `ls -a` listet auch versteckte Dateien und Ordner auf (diese beginnen mit einem Punkt)
  - `ls -l` stellt das dann auch wirklich als Liste mit Metainfos dar
  - `ls -alh` verbindet beides und zeigt Größen in lesbarem Format an (z.B. `2,3G` statt `2368641604`)

Beispiele siehe oben.

Kopieren

- `cp -r foo anderer/ordner/` kopiert den Ordner `foo` samt Inhalt nach `anderer/ordner` . Der Pfad der Kopie ist also `anderer/ordner/foo` .

Löschen

- `rmdir ordner` und `rm -r ordner` löschen den Ordner `ordner` samt darin enthaltenen Dateien und

Unterordnern. Nutzer muss Schreibrecht auf `ordner` haben.

Berechtigungen

Linux kennt drei Berechtigungen: * Leserechte (`r`): Bestimmt, ob man Datei- oder Ordnerinhalt lesen darf * Schreibrechte (`w`): Bestimmt, ob man in Datei oder Ordner schreiben darf * Ausführungsrechte (`x` wie in e**x**ecute): * Bei Dateien: Bestimmt, ob diese Datei als Programm ausgeführt werden darf * Bei Ordnern: Bestimmt, ob man den Ordner betreten darf

Darstellung

Normalerweie schreibt man eine Berechtigung als `rwx` oder `r-x` oder ähnliches.

Manchmal sieht man auch Zahlenfolgen wie 604 oder ähnliches. Jede Ziffer steht für ein Dreierbündel aus `rwx` :

Zahl	`rwx` -Pendant
0	`---`
1	`--x`
2	`-w-`
3	`-wx`
4	`r--`
5	`r-x`
6	`rw-`
7	`rwx`

Also steht `754` für `rwxr-xr--` oder in Worten "Ich darf alles, Gruppenmitglieder dürfen lesen und ausführen und der Rest nur lesen".

Beispiele

```
$> ls -alh datei.txt
-rwxrw-r-- 1 foo bar 3,9K 13. Nov 13:58 datei.txt
```

- gehört Nutzer `foo` und Gruppe `bar`
- ist 3,9 KB groß
- wurde am 13.11.2021 um 13:58 das letzte mal geändert
- das `-` ganz am Anfang sagt "dies ist eine Datei" (bei Ordnern steht da `d` für **d**irectory)
- das `rwxrw-r--` sagt folgendes:
  - Der Nutzer `foo` hat Lese- (`r`), Schreib- (`w`) und Ausführungsrechte (`x` wie e**x**ecute)
  - Mitglieder der Gruppe `bar` haben Lese- und Schreibrechte (`rw-`)

- Alle anderen haben nur Leserechte (`r--`)

```
$> ls -alh ordner
insgesamt 12K
drwxr-xr-x  2 foo bar 4,0K 27. Nov 23:11 .
drwxr-xr-x 65 foo bar 4,0K 27. Nov 23:11 ..
-rwxrw-r--  1 foo bar 3,9K 13. Nov 13:58 datei.txt
```

Puh, das sieht komplizierter aus. Also gut:

- Insgesamt ist der Ordner (ohne Unterordner!) run 12 KB groß
- Der erste Eintrag gehört zum Ordner `.` . Dieser Ordner ist *immer* der Ordner in dem sich die Shell gerade befindet. Die Datei `./foo.txt` ist also die Datei `foo.txt` im aktuellen Ordner.
- Der zweite Eintrag gehört `..` , was der übergeordnete Ordner ist. Sagen wir der Ordner in diesem Beispiel hat den Pfad `/home/foo/ordner` , dann entspricht `..` dem Ordner `foo` .
- Der dritte Eintrag zeigt unsere Datei vom vorigen Beispiel.
- Das `d` ganz am Anfang der beiden ersten Zeilen heißt "dies ist ein Ordner" (weil Ordner im englischen = **d**irectory)
- Berechtigungen der Ordner (also `drwxr-xr-x`)
  - Benutzer `foo` darf Dateien im Ordner auflisten (`r`), Dateien erstellen (`w`) und den Ordner betreten (`x`)
  - Alle anderen dürfen Dateien auflisten und den Ordner betreten (`r-x`)

Berechtigungen ändern

- `chmod +w datei.txt` bzw. `chmod u+w datei.txt` (`u` = user) erteilt Schreibrechte für den aktuellen Nutzer auf der Datei `datei.txt`
- `chmod g+w datei.txt` (`g` = group) erteilt Schreibrechte für Gruppenmitglieder
- `chmod o+w datei.txt` (`o` = other) erteilt Schreibrechte für alle anderen
- `chmod -R ug+r ordner` erteilt Leserechte für mich und alle Gruppenmitglieder für alle Dateien im Ordner `ordner`
- `chmod 751 datei.txt` verteilt die Rechte `rwxr-xr--` auf der Datei `datei.txt`

Dateien & Ordner suchen

Tool der Wahl um Dateien und Ordner zu suchen/finden: `find`

- Wie finde ist Datei `foo.txt` ? → `find -name foo.txt`
- Wie finde ich Datei `<irgendwas>wodafon-rechnung<irgendwas>` ? → `find -name "*wodafon-rechnung*"`
- Wie finde ich alle PDF-Dateien im Ordner `foo` ? → `find foo -name "*.pdf*`
- Wie finde ich Ordner `bar` ? → `find -type d -name bar`

Text in Dateien suchen

- In welchen Dateien kommt der Text `wodafon ist doof` vor? → `grep -Hirn "wodafon ist doof"`
- In welchen Dateien im Ordner `foo` kommt der Text `wodafon ist doof` vor? → `grep -Hirn "wodafon ist doof" foo`
- In welchen Dateien kommt das Muster `<irgendwas>bar<vier-Ziffern>` also vor? → `grep -HirnP "bar[0-9]{4}"`

Prozesse

Prozesse auflisten und System-Auslastung anschauen

`htop`: Quasi wie der Task-Manager in Windows. Bei den größeren Distributionen (z.B. Ubuntu und Ubuntu-Verwandten) direkt mit dabei, ansonsten muss man `htop` nachinstallieren.

Prozess beenden

- `kill 12345` beendet Prozess mit der ID (PID) 12345
- `killall firefox` beendet alle Prozesse des Programms `firefox`

Eventuell mit man dem Befehl ein `sudo` voranstellen (also `sudo kill ...`), je nachdem ob der eigene Nutzer oder jemand anderes (z.B. das Betriebssystem) den Prozess gestartet hat.

Texteditoren

`nano`

Mit `nano foo.txt` barbeitet man Datei `foo.txt`. Shortcuts stehen am unteren Bildschirmrand (z.B. `^X` heißt STRG-x).

`vim`

Uff .... ja ne, das erkläre ich jetzt nur grob. Also `vim` arbeitet mit verschiedenen Modi/Zuständen und bedarf daher etwas Übung.

Startet man `vim` ist man im "normal mode", VIM nimmt in diesem Modus nur Befehle entgegen, man kann aber nicht normal in der Datei schreiben. Befehle fangen mit `:` an (also Doppelpunkt, also halt die Shift- und Punkt-Taste drücken) und nützliche sind z.B. folgende: * `:w` schreibt (**w**rite) Änderungen der gerade geöffneten Datei * `:q` beendet VIM und kann nur benutzt werden, wenn keine ungespeicherten Änderungen vorliegen * `:q!` beendet VIM und verwirft ungespeicherte Änderungen * `:wq` speichert alles und beendet VIM * `:e foo.txt` öffnet (**e**dit) die Datei `foo.txt` im aktuellen Fenster.

Drückt man im "normal mode" `i` (also die normale i-Taste) ist man im "insert mode" und kann normale schreiben und mit den Pfeiltasten navigieren. Drückt man dann "ESC" (also die Escape-Taste), wechselt man zurück in den "normal mode".

Vim kann noch den ganzen geilen Scheiß (Makros, Syntax-Highlighting, Autovervollständigung, File-Browser, Tabs, Split-Screen, abgefahren mächtige Standard-Shortcuts und Befehle die so kompliziert sind, dass einem der Kopf explodiert, etc. pp.). Wusstest du, dass man mit `:%s/foo/bar/gc` den string `foo` durch `bar` ersetzen kann und VIM dich bei jeder Ersetzung fragt? Oder, dass man mit `:%s/^\(.*\)\n\1$/\1/` easy peasy alle doppelten Zeilen löschen kann?! ODER, dass man sich mit `map <C-g> :! bash -c "set -e xtrace; gcc -M % \| tr '\\\\ ' '\n' \| sed -e '/^$/d' -e '/\.o:[\t]*$/d' \| ctags -L - --c++-kinds=+p --fields=+iaS --extras=+q"<CR><CR>` einen Shortcut auf STRG-g definieren kann, der ... der äh ... ähm ... EGAL! Man kann zudem ne MILLIARDE Plugins installieren und VIM damit noch umfangreicher und komplexer machen solange bis es mächtiger ist als ALLE IntelliJ-IDEs ZUSAMMEN!!1! *bösewicht-lache*

Konfiguration der Bash

Die Konfiguration der Bash findet in der Datei `.bashrc` im Home-Verzeichnis statt. Bei Alternativen ist es ähnlich (z.B: bei ZSH ist es `.zshrc`).