

Kurzanleitung zu Linux

1. Nutzer- und Gruppenmanagement

1.1. Nutzermanagement

```
sudo adduser nutzername  
sudo userdel nutzername  
sudo passwd nutzername
```

sudo vim /etc/passwd -> gibt alle Nutzerkonten aus bzw lässt alle Konten konfigurieren

1.2. Gruppenmanagement

```
sudo groupadd gruppenname  
sudo groupdel gruppenname  
sudo adduser nutzername gruppenname -> fügt Nutzer zur Gruppe hinzu  
sudo deluser nutzername gruppenname -> löscht Nutzer aus Gruppe  
pkill -kill -u nutzername -> loggt nutzer aus dem System aus
```

sudo vim /etc/group -> Konfigurationsdatei aller Gruppen

2. Systemessentials

echo "text" -> gibt text in der Terminalausgabe aus

Inhalt zur Datei anfügen

echo "hi" << hi //fügt hi der datei an

cat file //gibt file ins Terminal aus

cat file file2 //(con)CATenates Files!

< file //gibts auf die POSIX-Art aus!

> file //scheiße, meine Datei ist leer!!!

clear -> löscht Terminalausgabenverlauf

mkdir Ordner -> erstellt Ordner

cd Ordner -> geht in Ordner

. -> dieser Ordner

~ -> Heimverzeichnis

cd .. -> geht einen Ordner im Pfad hoch

cd -> cd ~

rm -rf Ordner -> löscht Ordner rekursiv (also mit Dateien drin)

ls -a -> gibt alle versteckten Dateien aus

ls -al -> gibt alle versteckten Dateien aus mit allen Dateieinformationen aus

time cat file -> gibt Zeit für Ausgabe von file aus

touch datei -> erstellt Datei

mv dateiname dateiname2 -> benennt Datei um

mv dateiname Verzeichnis/ -> verschiebt Datei ins Verzeichnis

cp datei Verzeichnis/ -> kopiert Datei ins Verzeichnis

less datei -> man kann bequem die Datei lesen

file Datei -> gibt Dateiarart aus

ip addr -> gibt ip-adresse aus

pidof prozess -> gibt prozessnummer an

ps -e | grep -> listet alle Prozesse aus und pipet in grep, um zu filtern

kill prozessnummer -> tötet Prozess
 pkill prozessname -> tötet Prozess, aber diesmal durch Patternmatching
 date -> gibt Datum aus
 cal -> gibt Kalender aus
 ping seite.de -> pingt eine Seite an
 df -h -> gibt Laufwerkinformationen aus
 du -h -> gibt Größe von Dateien aus
 sleep 5 -> wartet fünf Sekunden
 alias rm=rm -rf -> setzt alias für rm -rf
 setxkbmap -option caps:escape -> macht capslock zu einem weiteren Escape
 pacman -Suy programmname // -u=Update, S=install, y=packageupdate; installiert packages
 seq 5 oder seq 200 // zählt bis 5 bzw 200
 initx bzw startx -> start xorg

2.1. USBs:

mkfs.vfat /dev/sdb bzw Name des USBs aus lsblk -> formatiert USB zu fat
 Mounten:

- 1) lsblk -> zeigt Laufwerke an
- 2) mount /dev/{USB-Name bei lsblk} {den Ordner im eigenen System, mit dem man mounten will}
 #ja, hier sieht man richtig. Zum Mounten braucht man einen eigenen Ordner. Wissen die wenigsten
- 3) umount /dev/{USB-Name bei lsblk} -> gibt Laufwerk wieder frei

3. Shells scripting

3.1. chmod

chmod +x script //macht das script ausführbar
 r (read) =4
 w (write) = 2
 x (execute)=1
 0 -> kein Zugriff
 7->rwX; 5->rx; 6->rw
 chmod 555 script //Lesen und Ausführen für Nutzer, Gruppe und Andere
 ls -la -> gibt Befugnisse der Dateien im Verzeichnis aus
 chmod -R 777 ordner //ändert Befugnisse im ganzen Ordner
 chown
 sudo chown nutzer:gruppe text.txt //text.txt gehört Nutzer/gruppe
 chown -R nutzer:gruppe ordner // im ganzen Ordner gehört alles nutzer/gruppe

3.2. Shells scripts

\$SHELL -> Shell, die grad läuft
 \$PATH -> Eine Liste von Verzeichnissen, in die man scripts einfügen kann, um sie von allen Verzeichnissen zugänglich zu machen
 export PATH=\$PATH:/kompletter/Verzeichnis/Pfad/vom/gewünschten/Ordnern //kann man in .profile einfügen, um ein neues Verzeichnis zu der Liste hinzuzufügen

Zu Skripten:

Scripts muss man mit chmod erst einmal ausführbar machen. Des Weiteren gehört in die erste Zeile, welche Shell verwendet wird. -> #!/bin/sh oder statt sh bash oder zsh oder dash, etc...

Das script lässt sich dann mit ./scriptname oder sh scriptname ausführen

3.3. Variablen:

irgendein_string = "wow"
 echo \$irgendein_string //gibt Variable aus

```
read hi
echo $hi //eingelese Variable wird ausgegeben
< $hi //überschreibt hi
$1 -> erstes Argument, welches Befehl bzw script übergeben wurde
```

3.3.1. Variablen addieren

```
z=$(( $x + $y ))
falls man Bock hat Kommazahlen zu addieren:
z=$(python -c "print($x + $y)")
```

3.4. Arrays

```
ein_array=(eins zwei drei)
echo ${ein_array[@]} //gibt ganzen array aus
echo ${ein_array[2]} //gibt zweites Element aus
```

3.5. if

```
if [ $name = "lol" ]; then
    echo "hi, lol"
elif [ $name = "nope" ]; then
    echo "nope"
else
    echo "something else"
fi
```

bzw bei Variablen statt Strings:

```
[ $x -eq 5 ]; then #eq = equals; -lt = <; -gt = >; -z = 0 bzw leerer String; -ne = not equals
```

3.6. for-Schleife

```
for i in ${ein_array[@]}; do
    echo $i;
done
//um statt eines Arrays den Befehlsoutput zu iterieren
for i in $(ls); do
    echo $i;
done
```

3.7. while-Schleife

```
i = 0
while [ $i -lt 10 ]; do
    echo $i; let i = i + 1 #anders kann man nicht iterieren
done
```

3.8. Funktionen

```
function hifunktion { echo hi } bzw hifunktion() { echo hi } #Deklaration und Definition
hifunktion #Funktionsaufruf
```

3.9. Zusätzliche Shellsyntax:

```
Befehl; Befehl2 -> zweiter Befehl wird ausgeführt, als sei er in der nächsten Zeile
Befehl || Befehl2 -> wenn Befehl nicht richtig ausgeführt wird, wird Befehl2 ausgeführt
Befehl & Befehl2 -> beide Befehle werden gleichzeitig ausgeführt
Befehl && Befehl2 -> Befehl2 wird nur ausgeführt, wenn Befehl korrekt ausgeführt wird
```

Achtung: Skripte werden immer in einer separaten Subshell ausgeführt, weswegen man bei einem `cd` in einem Skript nicht in dem Ordner bleiben wird, sondern "zurücktransportiert" wird!

Deswegen muss man zum Beispiel so auf sein eigenes Skript zugreifen, damit man im gewollten Ordner bleibt:

```
alias cs="source ~/.local/bin/cs"; cs
```

3.10. bashisms (nicht POSIX-Syntax)

`less <(cat file)` //output vom Command wird direkt als Argument eingegeben

```
[[ "$BROWSER" == "lynx" ]] && echo "sehr minimalistisch..." //glob-Matching-Bashism
```

```
[ "$BROWSER" = "lynx" ] && echo "sehr minimalistisch..." //auf die POSIX-Art
```

3.11. rc file (läuft in jedem nicht loginshell)

`~.bashrc` bzw. `.zshrc`

3.12. profile file (profile läuft nach dem Nutzerlogin)

`~.bash_profile` bzw. `.zprofile` bzw. `.profile` für POSIX shell

4. vim

Navigation bei vim:

```

      k
      ^
h <-|-> l
      v
      j

```

Befehle funktionieren nur im Normalmode bzw Visual Mode

`i` -> insertmode

`esc` -> normalmode

`ZZ` -> speichern und schließen

`:wq` -> speichern und schließen

`:q!` -> abrupt schließen

`!:shellbefehl` -> führt shellbefehl in vim aus

`a` -> append/insertmode

`e` und `a` -> gute Kombi

`A` -> appended am Zeilenende

`o` -> neue Zeile unten/insertmode

`O` -> neue Zeile oben/insertmode

`d` -> löschen bzw schon eher ausschneiden, da `d` das gelöschte yankt

`dw` -> löscht wort ab dem gerade ausgewählten Zeichen

`de` -> `dw`, aber lässt Leerzeichen dastehen

`d2e` -> `d2e` nur zweimal

`dd` -> löscht ganze Zeile

`2dd` -> löscht zwei ganze Zeilen

`d0` -> löscht bis Zeilenanfang

`d$` -> löscht bis Zeilenende

`x` -> löscht Zeichen

`r` -> ersetzt ein Zeichen

`R` -> ersetzt mehrere Zeichen

`c` -> change/wechseln

`ce` -> löscht das Wort, um es zu editieren

`cw` -> das selbe wie `ce`

`cc` -> wechselt ganze Zeile

`gg` -> geht zum Dateianfang

`G` -> geht zum Dateiende

`34G` -> geht in Zeile 34

0 -> geht zum Zeilenanfang
 \$ -> geht zum Zeilende
 b -> geht zum Wortanfang des letzten Wortes
 w -> geht zum Wortanfang des nächsten Wortes
 ge -> geht zum letzten Wortende zurück
 e -> geht zum Wortende
 6w -> geht sechs Wortanfänge weiter
 6e -> geht sechs Wortenden weiter
 gU -> macht Buchstaben groß
 gu -> macht Buchstaben klein
 gq -> macht aus einer langen Zeile mehrere kürzere Zeilen
 gk -> geht eine visuelle Zeile nach oben
 gj -> geht eine visuelle Zeile runter
 { -> geht Absatz hoch
 } -> geht Absatz runter
 J -> joint/verbindet obere Zeile mit unterer
 % -> von einer Klammer zu ihrer korrespondierenden Klammer
 s/thee/the/g -> substitutes thee with the, g heißt, dass das nicht nur beim ersten Ergebnis substituiert wird
 u -> undo
 strg + r -> redo
 v -> visual mode
 V -> ganze Zeile wird ausgewählt
 4V -> vier ganze Zeilen werden ausgewählt
 y -> das ausgewählte wird kopiert
 yw -> Wort wird geyankt/kopiert
 yy -> ganze Zeile wird geyankt/kopiert
 p -> das kopierte wird eingefügt
 :r dateiname -> fügt Inhalt von dateiname in den jetzigen Text ein
 /suchbegriff -> öffnet Suche und sucht nach Suchbegriff nach Enter (casesensitive)
 /
 n -> geht in der Suche zum nächsten Patternmatch
 N -> geht bei der Suche zurück
 ?suchbegriff -> Suche nur rückwärts
 paar Sucheinstellungen:
 :set hlsearch -> Highlightsearch wird gesetzt
 :nohlsearch -> no Highlightsearch einstellen
 :set noic -> casesensitive einstellen
 :set ic -> ignore case bzw ignoriert Groß- und Kleinschreibung

5. Vim mapping

Nur zur info: Kommentare bei vim werden mit " gemacht

Die mappings kommen bei vim in -> ~/.vimrc
 und bei neovim/nvim in -> nvim/init.vim

imap kürzel ausgabe //im Insertmode wird kürzel durch die ausgabe ersetzt
 bsp: imap ;h <html> //ersetzt ;h immerzu durch <html>
 noch ein bsp: imap ;l .LP<Enter> //für groff-dokumente ganz nett, es gibt noch <Esc> und <Space>
 nmap mk :w<Enter>:!make %:r<Enter>:q<Enter> //macht make und kompiliert Datei; %:r steht für Dateiname ohne Dateiendung
 nmap kürzel ausgabe //im Normalmode werden bei kürzel die Ausgabebefehle ausgeführt

6. ssh (secure shell)

6.1. ssh-Server starten (openrc)

```
sudo rc-service sshd start
rc-service sshd stop
rc-update show //zeigt start scripts
rc-update add sshd
rc-update del sshd
```

6.2. auf ssh-server vom Client aus zugreifen

```
ip route get 1.2.3.4 | awk '{print $7}' bzw ip a //gibt ip-adresse (macht man aufm Server)
ssh username@192.168.0.1 //remote zugriff auf Server (username und ip vom Server)
ssh-keygen -t ed25519 //generiert key zur Authentifikation
ssh-copy-d username@192.168.0.1 //, der erlaubt ssh ohne login zu benutzen
exit //im ssh prompt, logt aus der remote maschine
```

7. rsync (Dateitransfer über ssh)

```
rsync file newfile --progress //cp syntax
rsync file [username]@192.168.0.1:/ordner_wo_datei rein soll //kopiert mit ssh
rsync -urvp [username]@192.168.0.1:/ordner_wo_datei rein soll //kopiert mit ssh
//u -> aktualisiert nur Dateien, die schon da sind, kopiert nichts neues rein
//r -> Ordner können mit ihrem Inhalt rekursiv kopiert werden
//v -> Die Ausgabe wird verbaler bzw man kriegt eine Ausgabe
//P -> progress
```

8. wc (word count)

```
wc -l -> gibt Zeilenanzahl in der Eingabedatei aus
wc -w -> gibt Wörteranzahl in der Eingabedatei aus
wc -b -> gibt Bytesanzahl in der Eingabedatei aus
wc -> gibt Zeilenanzahl, Wörteranzahl, Bytesanzahl aus
```

9. diff, patch und git

9.1. diff (stellt Unterschiede fest)

```
diff datei aktualisiertedatei -> gibt die Unterschiede zwischen beiden Dateien aus;
diff -u datei aktualisiertedatei > diffdatei.diff -> gibt die Unterschiede zwischen beiden Dateien aus, formatiert als eigene Diff-Datei
```

9.2. patch (patcht die Unterschiede)

```
patch < diffdatei.diff -> patch die alte Datei mit den neuen Sachen
patch -R < diffdatei.diff -> macht patch rückgängig
```

9.3. git (modernere Versionskontrolle)

```
git init -> erstellt repository
git add . -> alles im Ordner kommt ins Repository
git commit -m "ein commit"
git log -> gibt log der commits aus
git checkout commitnummer -> geht zu diesem commit zurück
```

10. grep (filtert Eingabe nach regulären Ausdrücken)

```
grep ".*rc" textdatei //markiert jede Zeile, die mit rc endet und eine beliebige Anzahl an vorherigen Chars hat, wobei egal ist, was für chars dies sind
```

```
find . | grep ".*.c" //suche in diesem Verzeichnis und gebe alles aus, was mit .c aufhört
grep "k+.c" txt //mindestens ein Zeichen muss zwischen k und .c sein
grep "c$" txt //sucht nach c am Zeilenende
grep "^b" txt //sucht nach b am Zeilenanfang
grep "\S*boot" txt //sucht nach allen nicht Leerzeichen vor boot
grep "https\?" txt //sucht nach einem Wort, welches mit http oder https anfängt
grep -i it txt //egal ob it klein oder großgeschrieben wurde, standardmäßig ist das aus
reguläre ausdrücke:
. -> egal, was da steht
* -> egal, welche Anzahl von dem, was links neben dem Kleinen Stern ist
\+ -> mindestens ein Zeichen muss da stehen, von dem was links daneben ist
$ -> markiert Zeilenende
^ -> markiert Zeilenanfang
\S -> nicht Leerzeichen, kann mit * rechts kombiniert werden
\s -> Leerzeichen
\? - Zeichen ist optional
[a-z] -> jeder kleine Buchstabe
[A-Z] -> jeder große Buchstabe
[A-Za-z] -> jeder Buchstabe
[0-9] -> jede Ziffer
```

11. find (sucht Dateien in Verzeichnissen)

```
find ~ -name threads.c | grep c++ //sucht Nutzerverzeichnis nach Datei "threads.c" ab
find / -name ".*rc" //sucht ganzes System nach einer Datei ab, die mit rc endet
find ~ -type d -name c++ //sucht nur nach Ordnern, wohingegen jeder find befehl ohne -type d als type -f ausgeführt
wird und nur nach Dateien sucht
find ~ -size +120k -size -1G //sucht nach Dateien im Größenbereich 120kb und 1Gb
```

12. cut (schneidet Text reihenbasiert aus)

```
cut -c1-5,8 text //gibt ersten bis fünften und achten Buchstaben in jeder Reihe von text aus
cut -b1,3-4 text //nur erster und dritter bis vierter Byte in jeder Zeile von text wird ausgegeben
cut -d "," -f1 text //alles links vom Komma in jeder Zeile wird ausgegeben
cut -d "," -f2 text //alles rechts vom Komma in jeder Zeile wird ausgegeben
```

13. awk (Text processing (teilweise reihenbasiert))

```
ps | awk '{print $1}' //gibt erste spalte von ps aus ($0 ist jede Spalte)
awk -F ":" '{print $1,$4}' /etc/passwd //macht den Seperator von awk einen Doppelpunkt statt einer Leertaste,
somit gibt awk die Spalte vorm ersten Komma und die Spalte vom vierten Komma aus
awk -F ":" '{print $1 " " $3 "\n" $5}' /etc/passwd //gibt aus ->erste_Spalte dritte_Spalte\fünfte_Spalte
awk 'BEGIN{FS=":"; OFS=" "}' {print $1,$6,$7}' /etc/passwd //FS=Field Seperator OFS=Output-FS der Befehl
gibt die Zeilen nur anders separiert aus
awk -F "/" '^/' {print $NF}' /etc/shells //F bestimmt Fieldseparator; '/' ... '/' ist die Reguläre ausdrucksyntax;
^/ -> sucht jede Zeile mit / als Zeilenanfang; $NF = Endspalte
awk -F "/" '^/' {print $NF}' /etc/shells | sort | uniq //sortiert die Shellausgabe und liefert nur Unikate (uniq braucht
vorsortierte Zeilen)
df | awk '/dev/nvme/ {print $1"\t"$2"\t"$3}' //wie man hier gut sehen kann braucht jedes / als Erkennungszeichen
vorher ein \, damit awk weiß, dass die Suche noch spezifiziert wird; separiert die Spalten mit tabs
df | awk '/dev/nvme/ {print $1"\t"$2+$3}' //addiert die zweite und dritte Spalte zu einer
awk 'length($0) > 7' /etc/shells //filtert die Zeilen danach, ob sie länger als sieben Zeichen sind
ps -ef | awk '{ if($NF == "/bin/zsh") print $0}' //gibt nur Prozesse aus, deren letzte Spalte in der Zeile /bin/zsh sind
awk 'BEGIN { for(i=1; i<=10; i++) print "Das Quadrat von",i, "ist", i*i;}' //BEGIN -> wird nur einmal ausgeführt
und am Anfang; ist eine for-
Schleife
```

awk 'NR==2, NR==5 {print NR, \$0}' /etc/shells //gibt alle Zeilen von 2 bis 5 aus; NR = Zeilennummer; ohne NR bei print hat man keine Zeilennummer

awk 'END {print NR}' /etc/shells /etc/passwd //gibt Summe der Zeilen aus shells und passwd aus; END -> print Befehl wird am Ende ausgeführt; nur eine Eingabedatei sorgt natürlich dafür, dass nur ihre Zeilenanzahl ausgegeben wird

awk '{print substr(\$0, 6)}' /etc/shells //gibt nur den Zeilenrest ab dem sechsten Zeichen

14. sed (Textprocessing, aber diesmal nur mit regulären Ausdrücken und einfacher als awk)

sed -i -> jetzt wird Datei nicht nur gefiltert ausgegeben, sondern auch direkt geändert

sed "s/c/C/g" datei //s=substitute; jedes kleine c wird mit dem großen C ersetzt; g=ersetzt jedes c in einer Zeile; ohne g -> ersetzt nur einmal in jeder Zeile

sed "s/#.*//g" datei // alles nach # wird ersetzt durch nichts, da / alleine steht

sed 11q datei -> bescheuerte Art nur die ersten elf Zeilen einer Datei auszugeben, wenn mans wie head benutzt

sed "s/#.*//g; /cf/ d" datei -> wie das davor, nur löscht alle cf Zeilen (p statt d wäre print und q statt d wäre quit)

sed "s/#.*//g; /^\$/ d" datei -> wenn der Anfang gleich dem Ende ist (also eine Leerzeile vorliegt), lösche die Zeile

Table of Contents

Nutzer- und Gruppenmanagement	1
Systemessentials	1
Shellscripting	2
vim	4
Vim mapping	5
ssh (secure shell)	6
wc (word count)	6
diff, patch und git	6
grep (filtert Eingabe nach regulären Ausdrücken)	6
find (sucht Dateien in Verzeichnissen)	7
cut (schneidet Text reihenbasiert aus)	7
awk (Text processing (teilweise reihenbasiert))	7
sed (Textprocessing, aber diesmal nur mit regulären Ausdrücken und einfacher als awk)	8