Linux für Einsteiger – Teil 3

Das Command Line Interface (CLI)

sylvia@cyber4edu.org micu@cyber4edu.org

Warum beschäftigen wir uns damit?

```
user@hostname:~$
```

- Das CLI ist **wie ein Zauberwerkzeugkasten** für deinen Computer.
- Wer erstmal gelernt hat, ein wenig damit umzugehen, kann den Computer **sehr schnell und effektiv** bedienen!
- Oder gar **automatisieren**
- Es gibt unendlich viel zu **entdecken** – los geht's!

CLI via Linux Console Terminal

Als es noch keine graphischen Interfaces gab, sah man nach dem Boot einfach nur einen **Login-Prompt**, und das ist heute auch noch so bei Systemen, die ohne GUI installiert werden (z.B. viele Server-Systeme).

Hier wird dann direkt auf diesem Console Terminal gearbeitet.

Terminal-Anwendungen

Es gibt aber auch “virtuelle” Terminal-Anwendungen, auf die über die GUI zugegriffen werden kann.

- Der **Terminal** ist das Programm, welches über eine sog. Shell Befehle entgegen nimmt.
- Jeder Linux-“Flavour” bringt seine eigene Version eines Terminal-Programms mit, oft direkt verbunden mit der Desktop-Applikation: GNOME hat den GNOME-Terminal, Ubuntu MATE hate den MATE Terminal, welche manchmal aber auch universal installierbar sind.
- Die Unterschiede sind oft im Design und manchmal in den Funktionen, wobei grundsätzliche Befehle und Funktionen sehr ähnlich sind.

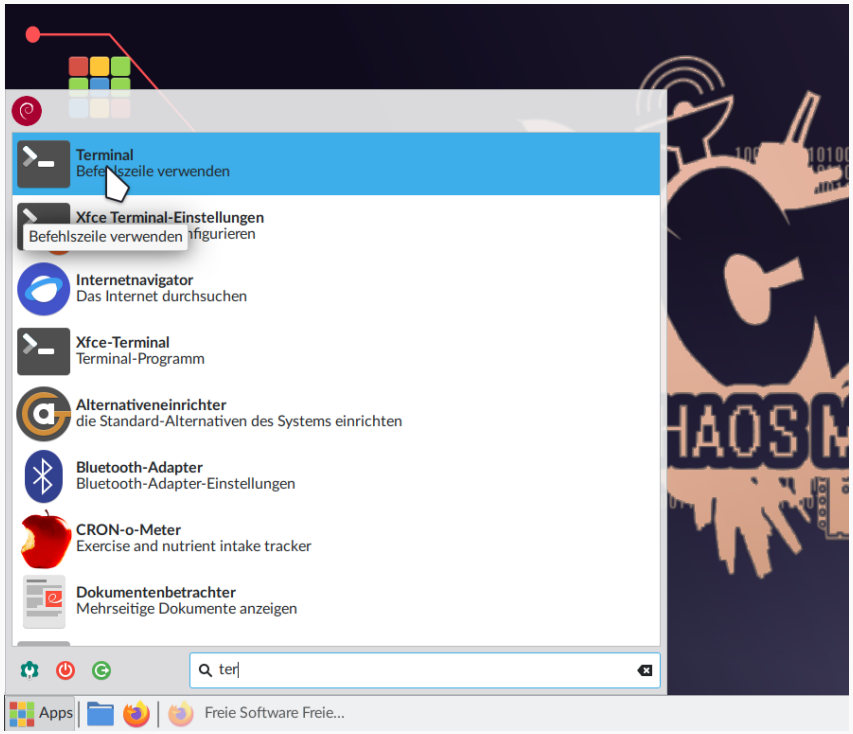
Shell

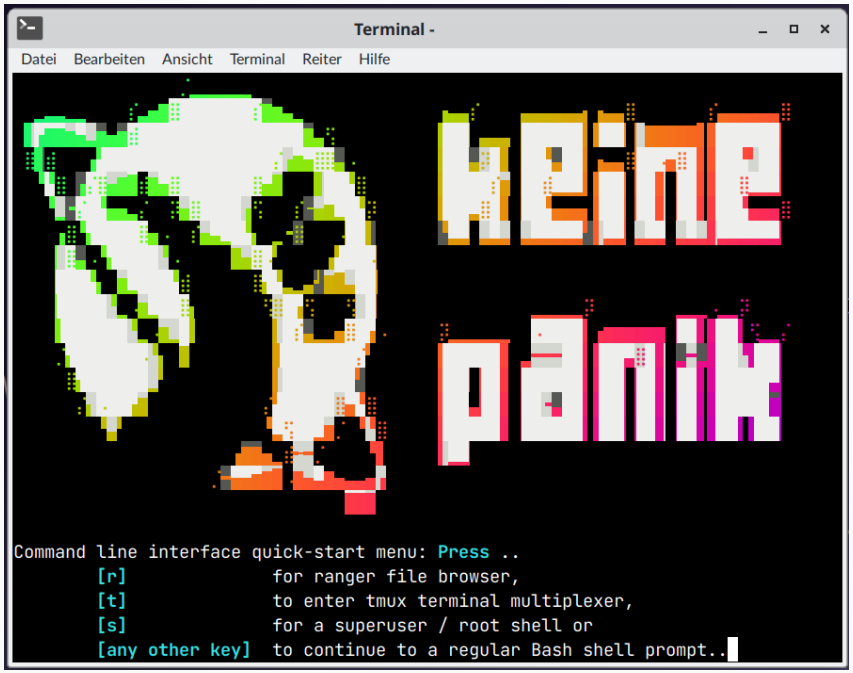
- Eine Shell nimmt Befehle, welche über das Terminal eingegeben wurden, entgegen, verarbeitet diese, und interpretiert.
- Beispiele sind “bash”, “zsh”, “csh”, “ksh”, “fish”
- Die **bash** ist weit verbreitet und auf den meisten Linux-Systemen verfügbar

Basics

- Das Linux Terminal ist eine **textbasierte Schnittstelle**, die es dir ermöglicht, Befehle an das Betriebssystem zu übermitteln.
- Oder dort Ausgaben zu lesen.
- Linux und vor allem alle Terminal-Anwendungen sind "Case-sensitive", d.h. es ist bei allen Befehlen sehr wichtig, sie richtig zu schreiben – inklusive **Groß- und Kleinschreibung!**
- Verwende zur Vervollständigung von existierenden Dateinamen oder Ordnernamen `[TAB]`.
- Das funktioniert auch bei vielen Befehlen und/oder ihren Optionen.
- Mit dem `[Pfeil nach oben]` bekommt ihr wieder den letzten eingegebenen Befehl.

Öffne ein Terminal!





. . . und drücke irgendeine Taste, aber nicht r, t oder s

In der Shell...

- Du siehst, dass da etwas steht wie:

```
user@hostname:~$
```

- Was da steht nennt man **Prompt** und der sieht je nach Shell und Konfiguration anders aus.
- Je nach Shell funktionieren manche Dinge auch etwas anders – aber prinzipiell ähnlich.
- Dieser Prompt hier zeigt
  - den aktuell angemeldeten User `user`,
  - am Rechner mit dem Namen `hostname` und
  - im persönlichen Verzeichnis dieses Benutzers (`~`).
- Hier kann der/die jeweilige Benutzer*in Dateien abspeichern.
- Auch Programme legen hier Benutzer*innen-Spezifisches ab.

Das Command ls

```
user@hostname:~$ ls -la
```

- "ls" ist ein Kürzel für "list". Also listet der Befehl Dateien auf in dem Verzeichnis, wo ich mich gerade befinde.
- "ls" wird hier mit den Optionen "la" aufgerufen
  - hinter dem Minus
  - die Reihenfolge dieser beiden Optionen ist egal (`al` oder `la`)
- Das "a" bedeutet, dass ALLE Dateien angezeigt werden sollen, auch sog. versteckte Dateien. Das sind die Dateien mit einem Punkt vor dem Namen.
- Und das "l" bedeutet, dass einige Zusatzinformationen angezeigt werden sollen, nämlich die Spalten vor dem Dateinamen.

Command Struktur

Du siehst am Beispiel des “ls”-Commands, dass es eine Struktur gibt:

Command	Option	und vielleicht ein Argument
ls	-la	/home/user/

Probiere auch `ls -la /usr/bin | more`. Was macht das?

man pages

Man muss sich Optionen zu Befehlen nicht merken, denn es gibt die sogenannten “man pages”.

Teste

```
user@hostname:~$ man ls
```

Die man pages sind gruppiert:

```
user@hostname:~$ man man
```

TLDR

Eine Kurzversion mit Beispielen liefert

```
user@hostname:~$ tldr man
```

```
man
Format and display manual pages.More information:
https://www.man7.org/linux/man-pages/man1/man.1.html.
 - Display the man page for a command:
   man {{command}}
 - Display the man page for a command from section 7:
   man {{7}} {{command}}
 - List all available sections for a command:
   man -f {{command}}
 - Display the path searched for manpages:
   man --path
```

help!

Bei den meisten Kommandos gibt es auch noch `--help`:

```
user@hostname:~$ ls --help
```

Das allgemeine `help` (ein eigener Befehl) liefert die Hilfe für alle internen Shellkommandos.

```
user@hostname:~$ help
GNU bash, Version 5.1.16(1)-release
 ↪  (x86_64-pc-linux-gnu)
Diese Shellkommandos sind intern definiert. Geben Sie
 ↪  »help« ein, um diese Liste zu sehen.
Geben Sie »help Name« ein, um die Beschreibung der
 ↪  Funktion »Name« zu sehen. (...)
```

Eh... welcher Wochentag war der 23. dieses Monats?

Probiere einmal `cal` aus:

```
cal
```

oder Weihnachten nächstes Jahr

```
cal 12 2025
```

Falls auf deinem Computer eine Meldung wie “command not found” kommt, musst du wahrscheinlich das Programm “cal” installieren mit

```
sudo apt install ncal
```

`sudo` macht, dass du die Berechtigung hast, zu installieren, `apt install` sagt dem Computer, dass er etwas installieren soll.

Wie überprüfe ich, was in welcher Version installiert ist und welche Pakete es gibt?

Dazu gibt es verschiedene Möglichkeiten (hier am Beispiel des Programms *toilet* – dazu gleich mehr):

```
which toilet
toilet --version
apt search toilet
apt show toilet
```

- **which** sagt uns, wo das Programm liegt
- --**version** gibt bei vielen Programmen die aktuelle Versionsnummer aus
- **apt search** und **apt show** geben Infos zu dem **Paket**
- Häufig heißen Programm und Paket gleich.
- Bei `cal` heißt das dazugehörige Paket aber `ncal`.

figlet und toilet

Du hast vielleicht schon einmal gesehen, wenn ASCII-Text auf einem Terminal toll dargestellt wird.

Installiere figlet und toilet und spiele damit!

```
sudo apt install figlet
sudo apt install toilet
figlet "38C3"
toilet "38C3"
```

Und schau dir die man pages dazu an, um die Möglichkeiten zu erfahren.

`showfigfonts` zeigt verschiedene verfügbare Schriftarten an.

Wer ist alles angemeldet?

Unix und Linux sind **Mehrbenutzersysteme**. Das bedeutet auch, dass mehrere Personen gleichzeitig am selben Rechner angemeldet sein und arbeiten können. Oder man selber an mehreren Terminals.

```
user@hostname:~$ who
```

touch

- Mit `touch file` kann man eine neue Datei “file” anlegen
- Existiert die Datei bereits, dann wird ihr Änderungsdatum aktualisiert (modification timestamp)
- Es können auch mehrere Dateien angelegt werden:

```
touch file1 file2 file3
```

cat

- mit `cat file` kann man sich den Inhalt einer Datei ansehen
- mit `cat file1 file2` wird der Inhalt mehrerer Dateien hintereinander ausgegeben
- `cat -n` gibt den Inhalt mit Zeilennummern aus.
- `zcat` zeigt den Inhalt gezippter Dateien

less

less ist ein sog. Pager, mit dem sich Dateien anschauen lassen (aber nicht editieren)

- Mit `less file` kann man sich den Inhalt einer Datei ansehen.
- Wenn der Dateiinhalt länger als eine Bildschirmseite ist, stoppt die Anzeige.
- Mit `[ENTER]` geht es eine Zeile weiter.
- Mit `[SPACE]` (der Leertaste) geht es eine Seite weiter.
- Mit `q` (für quit) könnt Ihr die Anzeige (ggf. auch vorzeitig) wieder verlassen
- Habt Ihr auf die letzte Zeile geachtet?

Cowsay... wth cowsay?

```
sudo apt install cowsay
cd /usr/games/
cowsay "38c3"
```

Falls ihr etwas Anderes als eine Kuh wollt, werft doch mal einen Blick in das Verzeichnis `/usr/share/cowsay/cows/`. Testet die Option `-f`.

Probiert gerne auch die grafische Variante `xcowsay`.

alias

Vielleicht habt Ihr gerade gesehen, dass statt der Kuh auch ein süßer Pinguin was sagen kann.

```
cowsay -f tux "38c3"
```

Vielleicht wollt ihr den Parameter nicht jedes Mal mitgeben.

Dafür kann ein **alias** definiert werden.

```
alias tuxsay="cowsay -f tux"
tuxsay "38c3 mit Pinguin!"
```

Probiert es aus.

echo

`echo` gibt einfach den nachfolgenden Text aus.

```
user@hostname:~$ echo "Das ist ein Test"
Das ist ein Test
```

Das kann man nutzen, um in eine Datei zu schreiben.

Achtung: Bevor Ihr das testet

- Beim Umleiten der Ausgabe in eine **vorhandene** Datei mit > wird ihr Inhalt **überschrieben:**
- Existiert die Datei nicht, wird sie angelegt.

```
user@hostname:~$ echo "Das ist ein Test" > file
user@hostname:~$ cat file
Das ist ein Test
```

Ausgabeumleitung mit >

Achtung: beim Umleiten der Ausgabe in eine Datei mit > wird ihr Inhalt überschrieben:

```
user@hostname:~$ cat file
Das ist ein Test
user@hostname:~$ echo "Das auch" > file
user@hostname:~$ cat file
Das auch
```

Ausgabeumleitung mit >>

Wenn man Inhalte **anhängen** will, muss man **>> benutzen**:

```
user@hostname:~$ echo "Das ist ein Test" > file
user@hostname:~$ cat file
Das ist ein Test
user@hostname:~$ echo "Das auch" >> file
user@hostname:~$ cat file
Das ist ein Test
Das auch
```

Durch die Verzeichnisse wandern

Mit `cd` (Change Directory) kannst du das Verzeichnis wechseln, wobei du absolute oder relative Pfade angeben kannst.

```
cd directory
cd /home/user/Downloads/
cd ..
```

- Das Beispiel `cd /home/user/Downloads/` zeigt einen absoluten Pfad.
- Ein relativer Pfad beginnt nie mit / und gibt das Ziel aus dem Blinkwinkel des derzeitigen Ortes an.
- Mit Leerstelle und den beiden Punkten gehen wir eine Ebene nach “oben” – Richtung Wurzelverzeichnis.
- Mit `cd` kommst Du wieder zurück in dein Benutzerverzeichnis `/home/user`.

Durch die Verzeichnisse wandern (contd.)

Mit

```
cd ../Dokumente
```

gehen wir eine Ebene nach oben und dann direkt in den Ordner Dokumente.

Mit `[TAB]` kannst du die Namen immer vervollständigen.

```
cd ../Do[TAB]
```

hätte hier auch geklappt, wenn “Dokumente” hier der einzige Ordner ist, der mit “Do” anfängt (sonst werden Dir bei einem weiteren `[TAB]` alle möglichen Treffer angezeigt).

Wo bin ich?

Hier sehen wir, dass wir uns im Verzeichnis `/home/user` befinden:

```
user@hostname: /home/user $
```

Oft sieht man nur eine Abkürzung, bei der `~` für das eigene Home-Verzeichnis steht:

```
user@hostname: ~ $
```

Aber nicht jeder Prompt zeigt immer an, wo man sich gerade befindet. Dann hilft `pwd` (kurz für Print Working Directory).

```
$ pwd
/home/user
```

mkdir und copy

Vielleicht möchtest du Dateien im Filesystem hin und her kopieren. Du befindest dich im Verzeichnis `/home/user/`.

Mit

```
mkdir directory1 directory2
```

erstellst du zwei neue Verzeichnisse.

Mit

```
cd directory1
```

gehst du in das neue directory1-Verzeichnis.

mkdir und copy (contd.)

Mit

```
touch newfile
```

erstellst du im directory1 eine neue Datei “newfile”.

Dann gehst du mit

```
cd ../directory2
```

in das Verzeichnis “directory2”.

mkdir und copy (contd.)

Du möchtest nun die neue Datei, die sich in directory1 befindet, in das directory2 kopieren, also dahin, wo du gerade bist.

Das machst du mit

```
cp ../directory1/newfile .
```

Der Punkt am Ende gibt an, dass du die Datei in dasjenige Verzeichnis kopierst, wo du dich gerade befindest.

Du hättest genauso gut auch eine absolute Pfadangabe machen können

vi / vim

Es gibt verschiedene Programme, mit denen Dateien editiert werden können, eines davon ist **vi**/**vim**. Es ist immer noch eines der beliebtesten Textbearbeitungsprogramme und auch wenn die Kommandos erstmal etwas obskur erscheinen – wenn man die Basics erstmal drauf hat, lassen sich Dateien superschnell editieren!

Mit diesem Kommando erzeugst du eine neue Datei namens "filename" und springst auch gleich in die Datei hinein:

```
user@hostname: /home/user $ vim filename
```

vi / vim – command mode

Wenn du eine Datei öffnest, bist du zunächst im “command mode” oder “normal mode”, du kannst also Kommandos eingeben, wie z.B.:

```
h -> move left
j -> move down
k -> move up
l -> move right
PageDown oder Ctrl+F -> move one page forward
PageUp oder Ctrl+B -> move one page backward
shift + G -> move to last line
gg -> move to first line
```

Es gibt noch mehr Kommandos, die lernen wir aber später kennen!

vi / vim – ex mode

Hier lassen sich hinter einem Doppelpunkt Kontrollkommandos eingeben, wie z.B.:

```
:q -> quit an unchanged file
:q! -> quit and discard changes
:w filename -> save file under different name
:wq -> save file and quit
```

vi / vim – Insert/Append Mode

```
i -> Füge Zeichen links von der i-Eingabe ein
a -> Füge Zeichen recht von der a-Eingabe ein
```

Mit `Esc` kommen wir aus dem Insert-/Append-Mode wieder raus.

Es gibt noch einige mehr Kommandos und die wollen wir nun nicht nur kennenlernen, sondern auch üben!

Öffne

```
vimtutor
```

nano

`vim` oder `vi` ist auf praktisch jedem Linux-System verfügbar, daher lohnt es sich, sich mit der Bedienung auseinander zu setzen und zumindest die Basis-Tasten zu lernen.

Es gibt aber auch andere Editoren, die Viele einfacher zu bedienen finden und daher bevorzugen. Einer davon ist **`nano`**.

Startet `nano` (für eine neue leere Datei) oder `nano file` zum Bearbeiten der Datei *file*.

Unten seht ihr eingeblendet die Tastenkommandos für die häufig benutzten Befehle, wobei `^` für die CTRL- bzw. STRG-Taste steht und `M-` für die ALT-Taste.

Probiert es aus.

grep – Basics

Mit grep können wir Text in Dateien suchen.

```
grep bananas foo.txt
grep -i bananas foo.txt
grep -v bananas foo.txt
grep -r bananas /home/user
grep -lr bananas /home/user
```

- Statt `grep -i` geht oft auch `igrep` für case insensitive.
- `grep -v` zeigt alle Zeilen, wo der Suchbegriff *nicht* vorkommt
- `grep -r` durchsucht alle Dateien in einem Verzeichnis
- `grep -l` zeigt nur die Dateinamen, wo der Text vorkommt

Schau dir `man grep` an und probiere aus, wie du eine Datei `foo.txt` erstellst mit dem Text-Inhalt “bananas”, die Datei abspeicherst und dann nach dem Text “bananas” suchst!

grep – Suche im FSFW-Material (1)

Suche im mitgelieferten FSFW-Material nach **linux**:

```
grep linux -lr ~/FSFW-Material
```

```
/home/user/FSFW-Material/hello-world-collection/README.md
```

Suche nach **linux**, aber **case insensitive**:

```
grep linux -lri ~/FSFW-Material
```

(erinnere Dich: die Reihenfolge der Optionen ist egal)

```
/home/user/FSFW-Material/hello-world-collection/README.md
/home/user/FSFW-Material/latex-vorlagen/praesentation/fsfw-beame
/home/user/FSFW-Material/Notes/Home.txt
/home/user/FSFW-Material/Notes/Linux-Befehle.txt
```

grep – Suche im FSFW-Material (2)

Durchsuche die zim-Notebooks für Programme im mitgelieferten FSFW-Material.

```
ls -l ~/FSFW-Material/zim-notebooks/Notes/Programme
```

Es gibt Infos zu *ksnapshot* und *zim*.

```
-rw-r--r-- 1 root root  510 19. Nov 09:05 ksnapshot.txt
-rw-r--r-- 1 root root 4885 19. Nov 09:05 zim.txt
```

Steht in einer der beiden Dateien “Windows”?

```
grep -nir windows
 ↪  FSFW-Material/zim-notebooks/Notes/Programme
```

```
.../Programme/ksnapshot.txt:11:(Meta = Windows-Taste)
```


(die Option `n` zeigt die Zeilennummer des Treffers)

grep – Alles außer Windows – geht das?

Wir wollen aber alle Dateien haben, in denen *nicht* Windows vorkommt (also hier die `zim.txt`) und probieren die Option `-v`:

```
grep -nirv windows
↪   FSFW-Material/zim-notebooks/Notes/Programme
```

Das sieht anders aus als wir erwarten. Warum?

`grep` durchsucht die Dateien **zeilenweise** und liefert uns nun jede einzelne Zeile, in der nicht *Windows* vorkommt!

Wir sehen, dass die Option `-v` also auch die `ksnapshot.txt` liefert und nur die Zeile 11 weglässt (das war die Zeile mit *Windows*).

Auch ein `grep -nirvl windows FSFW-Material/zim-notebooks/Notes/Programme` liefert immer noch beide Dateien. Was nun?

grep – Alles außer Windows – na klar!

Mit der Option `-L` werden alle Dateien **ohne Treffer** angezeigt!

```
grep -irL windows
 ↪  FSFW-Material/zim-notebooks/Notes/Programme
```

find

Mit find kann man Dateinamen im Filesystem finden.

```
find /home/user -name "*.txt"
find /home/user -iname "foo.txt"
```

Finde im Verzeichnis /home/user den Dateinamen (case insensitive mit "i") alle Dateien mit der Endung **"txt"** bzw. **"foo.txt"** (klein oder gross geschrieben).

Finde im FSFW-Material alle Markdown-Dateien (Endung **md**)!

```
find ~/FSFW-Material/ -name "*.md"
```

Wildcards

Als wir gerade nach Dateien mit der Endung `.txt` oder `.md` gesucht haben, haben wir ein `*` als Platzhalter für beliebige Zeichen verwendet.

Es gibt noch mehr Platzhalter (engl. *Wildcards*).

Wildcard	Bedeutung
`*`	irgendwelche Zeichen, egal wie viele
`?`	irgendein Zeichen (genau eins)
`[Zeichen]`	ein Zeichen aus der Liste
`[!Zeichen]`	ein Zeichen, das nicht in der Liste ist
`[[:Zeichenklasse:]]`	ein Zeichen einer bestimmten Klasse

Wildcards – Zeichenklassen

Zeichenklassen	Bedeutung
`[:alnum:]`	Irgendein alphanumerisches Zeichen
`[:alpha:]`	Irgendein alphabetisches Zeichen
`[:digit:]`	Irgendeine Ziffer
`[:lower:]`	Irgendein Kleinbuchstabe
`[:upper:]`	Irgendein Großbuchstabe

Wildcards – Beispiele

Pattern	Gilt für
`*`	Alle Dateien
`g*`	Jede Datei, die mit **g** beginnt
`b*.txt`	Jede Datei, die mit **b** beginnt und mit **.txt** endet
`Data???`	Jede Datei, die mit **Data** beginnt und danach noch **genau drei Zeichen** hat
`[abc]*`	Jede Datei, die mit **a**, **b** oder **c** beginnt
`BACKUP.[0-9]`	Jede Datei, die mit **BACKUP.** beginnt und danach noch **genau eine Ziffer** hat
`[[:upper:]]*`	Jede Datei, die mit einem **Großbuchstaben** beginnt
`[![:digit:]]*`	Jede Datei, die **nicht** mit einer **Ziffer** beginnt
`*[[:lower:]123]`	Jede Datei, die mit einem **Kleinbuchstaben** oder mit den Ziffern **1**, **2** oder **3** endet
`[a-cx0-9]*`	Jede Datei, die mit den Buchstaben **a**, **b**, **c** oder **x** oder einer **Ziffer** beginnt

Das Linux Filesystem

Gib folgendes in deinen Terminal ein:

```
user@hostname: /home/user $ ls /
```

Mit “/” landest du im “Wurzelverzeichnis”.

Das Linux Filesystem (contd.)

- /bin oder /sbin -> Binarys, ausführbare Dateien, GNU Utilities
- /boot -> Boot Dateien
- /dev -> Geräte-Verzeichnis
- /etc -> System-Konfigurationen
- /home -> Ort für Benutzerverzeichnisse
- /lib -> Ort für Programm-Bibliotheken
- /media -> Zugriffspunkt für “removable media”, z.B. USB Sticks
- /mnt -> Ort für zeitweise eingebundene Dateisysteme
- /opt -> oft für Softwarepakete verwendet
- /proc -> Prozess-Verzeichnis

Das Linux Filesystem (contd.)

- /root -> Das Homeverzeichnis des mächtigsten Benutzers
- /sys -> System-Verzeichnis für Kerne
- /tmp -> Verzeichnis für temporäre Dateien, wird bei jedem Reboot ausgeleert
- /usr -> Darin findet sich oftmals eine zweite "/-Struktur" mit commands, source code Dateien, Spielen, etc.
- /var -> Verchiedenes, v.a. für Logfiles

Datei-Berechtigungen in Linux

Es gibt 3 Personenkreise, die Berechtigungen bekommen können:

- Den **Eigentümer** einer Datei
- Eine bestimmte **Gruppe von Benutzern** (unabh. vom Eigentümer)
- **alle anderen**

Es gibt 3 Berechtigungsstufen für Dateien:

- **Lesen** (r wie read)
- **Schreiben** (w wie write)
- **Ausführen** (x wie execute)

Fast das gleiche gilt für Verzeichnisse:

- Dateien eines Verzeichnisses **auflisten** (r)
- **Dateien erzeugen** (w)
- **In Verzeichnisse wechseln** und Dateien aufmachen (x)

Datei-Berechtigungen in Linux (contd.)

Wie drückt man nun aus, wer was darf?

Die Berechtigungen werden von links nach rechts aufgereiht:
Eigentümer Gruppe Alle

Beispiele:

Muster	Berechtigungen
`rwxrwxrwx`	alle dürfen alles
`rwxr-xr--`	Eigentümer darf alles, Gruppe darf lesen und ausführen, alle dürfen lesen

Ein Bindestrich mittendrin heisst, dass dieses Recht nicht eingeräumt wird.

Datei-Berechtigungen in Linux (contd.)

Zurück zu ls

```
user@hostname:~$ ls -la
```

```
drwx------ myuser somegroup 21359 19 Nov 2024 Downloads
```

- Das “d” steht für “directory”
- Auf dieses Verzeichnis hat nur “myuser” mit Lesen/Schreiben/Ausführen Zugriff
- Alle anderen haben keinen Zugriff!
- Die Zahl nach “somegroup” ist einfach die Grösse des Ordners
- Danach kommt das Datum und der Verzeichnisname “Downloads”.

Datei-Berechtigungen in Linux (contd.)

```
$ ls -l foo.txt
```

```
-rwxr----- myuser somegroup 6 19 Nov 2024 foo.txt
```

myuser ist Datei-“Eigentümer*in”, die Gruppe somegroup darf immerhin lesen.

Wir könnten das leicht ändern mit:

```
$ chown otheruser:somegroup foo.txt
$ ls -l foo.txt
```

```
-rwxr----- otheruser somegroup 6 19 Nov 2024 foo.txt
```

Datei-Berechtigungen in Linux (contd.)

Die Berechtigungen selber können auch verändert werden.
Dateiberechtigungen sind Bits, z.B.:

user	group	all
rwx	r--	r--
111	100	100
7	4	4

1 heisst, dass das Bit "eingeschaltet" ist, 0 schaltet das Bit aus. 111 als Binärzahl = 7 in Dezimal, denn eigentlich ist das:

$$2^2 + 2^1 + 2^0 = 7$$

Datei-Berechtigungen in Linux (contd.)

Ich möchte, das “all” nicht mehr lesen darf, dafür soll “group” schreiben können. Das erreiche ich mit

```
chmod 750 filename
```

Puh, das war vielleicht kompliziert, vielleicht eine Runde spielen?

Es gibt eine Reihe ganz coole Spiele, die sich am Terminal spielen lassen.

```
sudo apt install bsdgames
cd /usr/games/
ls -la
./tetris-bsd
./snake
```

Probiere ein paar Spiele aus!

top und htop

top und htop zeigen im Live-Update Infos über die Systemauslastung und die Prozesse an.

ps und pstree

ps zeigt Informationen über laufende Prozesse, jeder Prozess hat eine Prozess-ID -> “PID”.

`ps aux` zeigt alle Prozesse und zusätzlich den User an, unter dem der jeweilige Prozess läuft. Im process state gibt es verschiedene Zustände:

- R: running
- S/D: asleep
- Z: Zombie

pstree zeigt auch die Abhängigkeiten zwischen Prozessen an. Prozesse können mit dem `kill`-Kommando zerstört werden. Schau dir `man kill` an!

Verzeichnis /proc

Jede PID hat ein eigenes Verzeichnis in /proc/, z.B. /proc/42. Darunter befindet sich jede Menge Information über den jeweiligen Prozess.

Schau dir `man proc` an!

Wetterbericht benötigt?

Probiere mal:

```
curl -H "Accept-Language: de" https://wttr.in/Hamburg
```

Falls `curl` nicht geht...

```
sudo apt install curl
```

disk usage

- `du` zeigt dir, wieviel Platz Dateien brauchen
- `df -h` zeigt dir, wieviel freier Platz auf jeder Partition ist, "-h" steht für human readable
- `df -i` zeigt dir an, wie viele sog. "inodes" noch zur Verfügung stehen. Inodes sind Metainformationen über angelegte Dateien. Wenn keine inodes mehr zur Verfügung stehen, können keine Dateien mehr angelegt werden!
- `ncdu` zeigt an, was viel Disk Space verbraucht und es kann auch durch die Anzeige navigiert werden.

Pipes

Manchmal ist es wünschenswert, dass ein Ergebnis eines eingegebenen Kommandos als Input für ein weiteres Kommando dient.

Wir können mehrere Befehle mit sogenannten **Pipes** aneinander stecken.

Das nutzen wir, um die Ausgabe eines Befehls an die Eingabe eines anderen weiterzuleiten.

```
ls | wc -l
```

Dieses Kommando macht ein "list files" innerhalb eines Verzeichnisses und zählt dann die Zeilen. Lies `man wc` und probiere den Befehl aus!

Eine Beispieldatei

Wir wollen gleich mit ein paar Kommandos herumspielen. Dafür benutzen wir eine Bücherliste in einer Datei.

Legt eine neue Datei `animals.txt` an (egal ob mit `vi`, `nano` oder `echo >>`) und schreibt folgenden Inhalt hinein (benutzt dabei genau 1 TAB zum Trennen der einzelnen Spalten):

```
python  Programming Python       2010  Lutz, Mark
snail   SSH, The Secure Shell    2005  Barrett, Daniel
alpaca  Intermediate Perl        2012  Schwartz, Randal
robin   MySQL High Availability  2014  Bell, Charles
horse   Linux in a Nutshell      2009  Siever, Ellen
donkey  Cisco IOS in a Nutshell  2005  Boney, James
oryx    Writing Word Macros      1999  Roman, Steven
```

wc kennt Ihr schon.

Probiert es mit der `animals.txt`:

```
wc animals.txt
```

```
7 51 325 animals.txt
```

wc gibt nacheinander die Anzahl der Zeilen, der Wörter und der Zeichen aus.

Wenn Ihr nur eins davon wissen wollt, schaut Euch die Optionen `-l`, `-w` und `-c` an.

head

Mit **head** könnt Ihr die ersten Zeilen einer Datei ausgeben. Mit der Option `-nX` könnt Ihr die Anzahl der gewünschten Zeilen angeben, wobei *X* eine Zahl ist – für die ersten drei Zeilen also `-n3`.

Probiert es mit der `animals.txt`.

```
head -n3 animals.txt
```

```
python  Programming Python      2010    Lutz, Mark
snail   SSH, The Secure Shell   2005    Barrett, Daniel
alpaca  Intermediate Perl       2012    Schwartz, Randal
```

head kann sowohl als Input als auch als Output dienen.

```
head -n3 animals.txt | wc -w
ls /bin | head -n3
```

cut

Mit **cut** könnt Ihr eine (oder mehrere) Spalte(n) aus einer Datei ausgeben.

Probiert folgende Befehle:

```
cut -f2 animals.txt
cut -f1,3 animals.txt
cut -f2-4 animals.txt
cut -c1-3 animals.txt
```

Was ist der Unterschied zwischen `-f` und `-c`?

Mit `-d` können wir ein Trennzeichen mitgeben. Könnt Ihr die Nachnamen der Buchautoren ausgeben?

```
cut -f4 animals.txt | cut -d, -f1
```

Verkettung mehrerer Pipes

Da die Pipe die Ausgabe eines Befehls mit der Eingabe eines anderen Befehls verbindet, der wiederum selbst eine Ausgabe hat, kann man dies auch mehrfach hintereinander schalten.

Versucht zu erklären, was folgender Befehl macht:

```
ls -l FSFW-Material/hello* | cut -c1 | grep d | wc -l
```

Wenn es euch schwer fällt, das zu überblicken, könnt ihr ja erstmal nur Teile davon ausführen und euch die (Zwischen-)Ergebnisse ansehen.

sort

Mit **sort** könnt ihr – wer hätte das gedacht – sortieren!

Probiert

```
sort animals.txt
sort -r animals.txt
```

Mit `-r` wird einfach rückwärts (absteigend) sortiert.

Zeigt nun eine sortierte Liste der Autoren an.

```
cut -f4 animals.txt | sort
```

Oder das früheste Erscheinungsdatum aus dieser Liste. . .

```
cut -f3 animals.txt | sort -n | head -n1
```

uniq (1)

Mit **uniq** können wir mehrfach vorkommende Werte reduzieren.

Schaut euch an, mit welchen Buchstaben die Nachnamen der Autoren beginnen.

```
cut -f4 animals.txt | cut -c1
```

Ihr bekommt eine Liste von Buchstaben, aber einzelne kommen mehrfach vor. Probiert

```
cut -f4 animals.txt | cut -c1 | uniq
```

Nicht ganz was wir wollten. **uniq** berücksichtigt nur direkt aufeinander folgende Zeilen. Lasst uns also erst sortieren.

uniq (2)

```
cut -f4 animals.txt | cut -c1 | sort | uniq
```

```
B
L
R
S
```

Besser, aber wie oft kommt denn jeder dieser Buchstaben ursprünglich vor? Dafür kennt `uniq` die Option `-c` (für *count*).

```
cut -f4 animals.txt | cut -c1 | sort | uniq -c
```

```
3 B
1 L
1 R
2 S
```

Duplikate finden mit md5sum (1)

md5sum bildet eine Prüfsumme, die sich ändert, wenn der Inhalt sich ändert. Das können wir nutzen, um inhaltliche Duplikate zu finden.

Dazu brauchen wir erstmal Beispieldateien:

```
cp animals.txt books.txt
head -n3 animals.txt > somebooks.txt
ls *.txt  # sollte 3 txt-Dateien zeigen
```

```
md5sum *.txt
```

```
7041084442b115279984b8a1ca7f23a8    animals.txt
7041084442b115279984b8a1ca7f23a8    books.txt
c69d56691a24876e388276f24204c106    somebooks.txt
```

Duplikate finden mit md5sum (2)

Bei nur drei Dateien sehen wir die Duplikate sofort. Bei langen Dateilisten sortieren wir lieber und lassen uns Vielfache anzeigen.

```
md5sum *.txt | cut -c1-32 | sort | uniq -c | sort -nr
 ↪   | grep -v "1 "
```

```
2 7041084442b115279984b8a1ca7f23a8
```

Die beiden Dateien mit demselben Inhalt finden wir nun mit

```
md5sum *.txt | grep 7041084442b115279984b8a1ca7f23a8
 ↪   | cut -c35-
```

```
animals.txt
books.txt
```

bash Tricks

- `cd -` -> gehe in das Directory, wo du zuvor warst
- Zeige deine letzten Kommandos an mit `history`
- Durchsuche deine Kommando-History mit Ctrl+R
- Wenn du ein Kommando eingibst, und direkt vor dem Kommando ein Space/Leerzeichen tippst, dann geht dieses Kommando nicht in die History!
- Ctrl+a -> Geh zum Beginn einer Zeile
- Ctrl+e -> Geh zum Ende einer Zeile
- Ctrl+l -> Clear the Screen
- `watch command` führt command per default alle 2 Sekunden aus

Huch, wieviel Uhr ist es?

Probiert

```
date
```

Automatisch aktualisieren?

```
watch -n 1 date
```

Oder etwas schöner

```
while true; do clear; echo "$(date '+%F %T' | toilet
↪   -f term -F border --gay)"; sleep 1; done
```

Bash brace expansion

```
echo {1..9}                 # liefert 1 2 3 4 5 6 7 8 9
echo {01..10}               # liefert 01 02 03 04 05 06 07
↪ 08 09 10
echo {1..10..2}             # liefert 1 3 5 7 9 (in
↪ 2er-Schritten)
echo {1..10..3}             # liefert 1 4 7 10 (in
↪ 3er-Schritten)
echo {a..z}                 # liefert a b c d e f g h i j k
↪ l m n o p q r s t u v w x y z
echo {a..z}{a..z}           # liefert aa ab ac ad ae af ag
↪ ah ai aj ak al am an ao ap aq ar as at au av aw ax
↪ ay az ba bb bc ....
touch test-{1..3}.txt       # legt die Dateien test-1.txt,
↪ test-2.txt und test-3.txt an
```

Vielleicht wieder Zeit, etwas zu spielen?

Lasst uns “Robotfindskitten” ausprobieren.

```
sudo apt install robotfindskitten
./robotfindskitten
```

system calls

Mit System Calls fragen Programme den Linux-Kernel, Dinge für sie zu tun, wie z.B. “lies etwas von der Festplatte”, “mache eine Netzwerkverbindung”, “erstelle einen neuen Prozess”, “installiere einen Tastatur-Treiber” etc.pp.

Benutzte System calls kann man sehen mit `strace`, z.B. `strace ls`

Update dein Linux über die cli

Mit

```
apt update
```

werden Paketlisten neu eingelesen und aktualisiert.

Diese werden im Verzeichnis `/etc/apt/sources.list` hinterlegt und sind je nach Version unterschiedlich und mit verschiedenen Variablen am Ende versehen.

Wenn es durch `apt update` Neuigkeiten gibt, dann können diese mit

```
apt upgrade
```

eingespielt werden. Das heisst, das ist dann das eigentliche Update.

Update dein Linux über die cli (contd.)

Hinweis: das gilt für Debian- und Ubuntu-basierte Distributionen, die das deb-Format für ihre Pakete verwenden (wie auch unser Schulstick).

Andere Linux-Distributionen benutzen andere Paketmanager, funktionieren aber ähnlich.

Siehe z.B.
https://www.stefanux.de/wiki/doku.php/linux/paketmanager

Update dein Linux über die cli (contd.)

Es gibt noch `apt dist-upgrade`. Dieser Befehl aktualisiert nicht nur die beständigen Pakete, sondern tauscht diese gegen neuere "Major"-Versionen aus.

Dies ist zum Beispiel der Fall, wenn ein System vollständig auf eine neue Version aktualisiert werden soll.

Hierbei müssen jedoch vorab die Paketquellen in der Datei /etc/apt/sources.list neu angegeben werden.

Ein paar Kommandos das Netzwerk betreffend

- `ip addr` oder kurz `ip a` -> Schau dir deine IP Kpnfiguration an
- `ip route` oder kurz `ip r` -> Schau dir deine Routing Konfiguration an
- `ping` -> pinge cyber4edu.org (`ping cyber4edu.org`) und 9.9.9.9 (`ping 9.9.9.9`) Was macht das? Schaue in `man ping` nach
- `traceroute` -> probiere `traceroute cyber4edu.org` Was macht das? Schaue in `man traceroute` nach
- netstat -> Mache `netstat -s` und `netstat -tp` Was macht das? Schaue in `man netstat` nach!
- `dig` -> Mache `dig cyber4edu.org` Was macht das? Schaue in `man dig` nach!
- whois und host -> Mache `whois cyber4edu.org` und `host cyber4edu.org` Was macht das? Schaue in `man whois` nach!

Noch eine andere Möglichkeit, Tetris zu spielen...

```
sudo apt install bastet
./bastet
```

Logging

Schaue nach, welche verschiedenen Logs du in `/var/log`finden kannst. Normalerweise finden sich in `/var/log/syslog` oder `/var/log/messages`. Diese Dateien findest du auf dem Schulstick-Linux nicht.

Logdateien seitenweise mit less ausgeben: `less /var/log/boot.log`

Die letzten 10 Logeinträge mit tail anzeigen: `tail -f /var/log/boot.log`

Oder bestimmte Wörter mit `cat` suchen: `cat /var/log/boot.log |grep error`

Nachrichten vom Kernel (bei Hardware Issues nützlich): `dmesg`

Und nun ein kurzer Film zum Entspannen

```
telnet towel.blinkenlights.nl
```

(vielleicht musst du `telnet` installieren)

Ein bisschen Mathe

Rechnen

```
echo $((3+5))
echo $((123*456))
```

Primzahlfaktorzerlegung

```
factor 1001
```

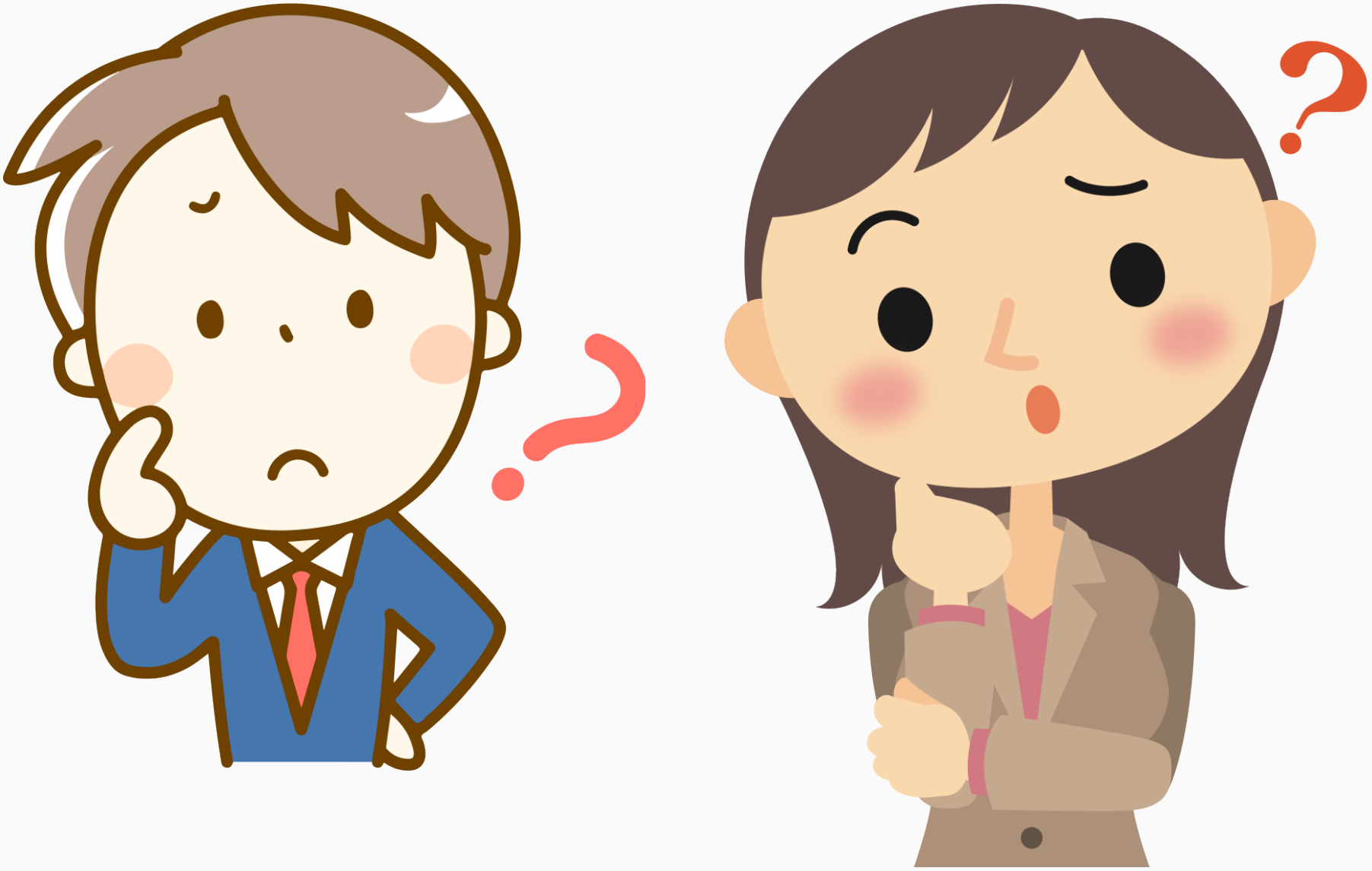
```
1001: 7 11 13
```

Pi

```
pi
```

```
3.14159265358979323846264338327950288419716939937510582097
```

Fragen?



micu@cyber4edu.org sylvia@cyber4edu.org

Links

- **cyber4EDU** https://www.cyber4EDU.org/
- **Chaos macht Schule** https://www.ccc.de/schule
- **Schulstick FSFW Dresden** https://schulstick.org
  - **Download** https://fsfw-dresden.github.io/schulstick-page/#download-und-bespielen-eines-usb-sticks
  - **Quellen** https://github.com/fsfw-dresden/usb-live-linux

Buchtipps

- **Linux – Das umfassende Handbuch**, 16. Auflage, Michael Kofler, Rheinwerk Computing, 978-3-8362-7132-5
- **Linux Kommandozeilenreferenz**, 4. Auflage, Michael Kofler, Rheinwerk Computing, 978-3-8362-6342-9
- **The Linux Command Line – A Complete Introduction**, 2nd Edition, William Shotts, No Starch Press, 7.3. 2019, 978-1-59327-952-3
- **How Linux Works**, 3rd Edition, Brian Ward, 2001, No Starch Press, 978-1-7185-0041-9
- **Efficient Linux at the Command Line**, 7th Release, Daniel J. Barrett, 2024, O'Reilly, 978-1-098-11340-7

Bild-Quellen

- https://freesvg.org/business-man-in-doubt
- https://freesvg.org/questioning-business-lady
- und eigene Screenshots

Lizenz

by sylvia@cyber4edu.org
and micu@cyber4edu.org

==========

Deutsche Übersetzungen:

- https://creativecommons.org/licenses/by-nc-sa/4.0/deed.de
- https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode.de