



## Shell-Safari / KISS 23.10.14

Die KISS ist vorbei und ihr habt noch Fragen, Kritik oder Verbesserungsvorschläge? Kein Problem, schreibt einfach an:

christoph.gordalla@rwth-aachen.de

### Über diesen Handout

Ziel dieses Workshops ist nicht, dass ihr nach dem Workshop jeden Befehl auf diesem Blatt blind anwenden könnt. Dafür ist auf diesem Blatt viel zu viel Stoff in zu kleiner Schrift, der auch noch in zu wenig Zeit durchgegangen wird. Vielmehr soll euch dieser Workshop einen Überblick verschaffen, was die Shell ist und kann, und **euch dazu einladen, nach dem Workshop selbst weiter zu experimentieren!** Programmierung kann man sich nicht anlesen, man muss sie selbst machen.

Als Hilfe dabei geben euch die Tuxe neben den Überschriften an, wie schwer jeweils ein Unterthema ist.

### Warum Linux-Shell?

Die Linux-Shell kann alles und sie kann es automatisiert. Egal, ob ihr:

- Textdateien mit Daten modifizieren (s. später)
- Bilder zu verschiedenen Formate konvertieren (convert)
- Plots erstellen (gnuplot)
- MP3s bearbeiten und umwandeln
- ...

müsst, ihr werdet das passende Programm finden, das in der Shell läuft! Nur warum will man sich so etwas antun, wenn es doch schöne grafische Programme für alles gibt? Ganz einfach, wenn ihr auf einmal **vor der Aufgabe steht, sehr viele Dateien zu bearbeiten, die möglicherweise auch noch in verschiedenen Ordnern liegen**, kann die Shell euch eine Menge Zeit sparen! Welches grafische Programm kann schon 500 Dateien mit Messdaten auf einen Schlag bearbeiten oder eben mal 2000 Urlaubsbilder neu sortieren? Die Shell kann es!

### Das Linux-Terminal

- Terminal öffnen: Strg + Alt + t
- Autovervollständigung über Tabulator
- ↑, ↓: alte Befehle noch einmal ansehen
- Nach alten Befehlen suchen: Strg + r
- Einen Prozess abbrechen oder die geschriebene Zeile löschen: Strg + c
- Kopieren: Strg + Umschalt + c
- Einfügen: Strg + Umschalt + v

#### Die wichtigsten Pfade in aller Kürze

- /home/USER Das für dich wichtigste Verzeichnis. Hier liegen deine Benutzerdaten und persönlichen Konfigurationsdateien und hier hast du alle Benutzerrechte (Achtung, du kannst daher auch ohne sudo alles löschen!)
- /media Hier werden USB-Sticks und externe Festplatten oder Partitionen in der Regel eingehangen.
- /dev steht für devices. Hier können alle Partitionen und Laufwerke angesprochen werden. /dev/sda2 ist z.B. eine Festplattenpartitionen. Ihr Inhalt kann aber nicht über den Pfad /dev/sda2 gelesen werden. Dazu muss diese erst gemountet werden (s. nächsten Punkt).
- /mnt Hier werden Laufwerke gemountet. Befindet sich z.B. auf /dev/sda2 die Partition mit Windows, kann diese über den Befehl `sudo mount /dev/sda2 /mnt` auf /mnt gemountet werden. Die "Eigenen Dateien" können dann unter dem Pfad "/mnt/Document and Settings/USERNAME/" erreicht werden.

### Pfade in Linux

Pfade beginnen in Linux immer mit dem **Wurzelfad** '/'. Wie in DOS kann man mit `cd` durch die Pfade springen:

- In den relativen Pfad (=Unterordner) Videos springen: `cd Videos`, in den absoluten Pfad /home/USER/Videos springen: `cd /home/USER/Videos`
- Dateien im Ordner lesen: `ls` oder `ls -la` (mit Details)
- In den letztbefindlichen Pfad springen: `cd -`
- In den Home-Ordner des Users springen: `cd` (ohne etwa dabei) oder `cd`
- Einen Pfad zurück: `cd ..`
- Aktuellen Pfad anzeigen `pwd`

### Copy, Paste, Entfernen, Dateien anzeigen

Kopiert, verschiebt oder löscht man in der Shell Dateien, sollte man immer wachsam sein. Im Gegensatz zum Dateimanager fragt die Shell in der Regel **nicht nach, wenn man gerade im Begriff ist, Daten zu löschen**. Generell gilt hier also umso mehr: Man sollte wissen, was man tut!

- FILE nach PFAD kopieren: `cp FILE PFAD`, z.B.: `cp daten.txt /home/USER/Messung` (absolut) oder `cp daten.txt Messung` (relativ, damit der Befehl dasselbe macht wie der vorherige Befehl, muss der Benutzer in /home/USER sein).
- FILE2 mit FILE1 überschreiben. **Achtung FILE2 ist danach gelöscht!**: `cp FILE1 FILE2`. FILE2 kann dabei auch in einem anderen Ordner liegen: `cp FILE1 Messung/FILE2`. Ist sogar mit obigen Befehlen kombinierbar: `cp FILE1 ../Messung1/FILE2`. Hier würde man erst in den Oberordner vom aktuellen Verzeichnis springen (`→ ..`) und dann in den Ordner Messung1 springen und dort FILE2 mit FILE1 überschreiben.
- `mv FILE PFAD` FILE nach PFAD verschieben (`mv` wie *move*).
- `mv FILE1 FILE2` FILE1 in FILE2 umbenennen (**Achtung:** Falls FILE2 bereits existiert, wird diese überschrieben!)
- `mkdir ORDNER` Neuen Unterordner ORDNER erstellen
- `touch FILE` Leere Datei FILE erstellen
- `rm FILE` Datei FILE löschen
- `rmdir ORDNER/` Ordner ORDNER löschen (geht nur, wenn der Ordner leer ist)

### Rekursives Arbeiten

Will man die obigen Befehle nicht nur auf eine einzelne Datei, sondern etwa auf alle Dateien in einem Ordner sowie auf alle Dateien in etwaigen Unterordnern anwenden, geht dies mit der Option `-r` (für rekursiv):

- Alle Dateien und Unterordner mit Dateien von Ordner ORDNER1 in den Ordner ORDNER2 kopieren: `cp -r ORDNER1/ ORDNER2/`
- Oder aber ihr nehmt das Sternchen \* als Platzhalter für alle Dateien in Ordner ORDNER1: `cp -r ORDNER1/* ORDNER2/`

**Achtung:** Man kann auch `rm` rekursiv benutzen: `rm -r ...`. Aber Vorsicht, falsch angewandt kann dieser Befehl großen Schaden anrichten.

### Textdateien anzeigen

Möchte man sich beim Arbeiten in der Shell eine Textdatei ansehen, will man dafür nicht extra einen grafischen Editor (etwa kate) öffnen und extra aus dem Terminal herauswechseln. Daher gibt es Programme, die einem die Textdatei in der Shell anzeigen:

- `cat TEXTDATEI` zeigt einem die komplette Datei an. Ist die Datei aber zu lang, sieht man nur das Ende. Für diesen Fall gibt es:
- `more TEXTDATEI` oder `less TEXTDATEI`. Hier kann man sich die Textdatei seitenweise anzeigen lassen. Beenden kann man beide Programme einfach mit der q-Taste.
- Es gibt auch **Texteditoren für die Shell**. Ein einfacher und selbsterklärender Editor, der sich für Einsteiger eignet, ist `nano`.

## Textdateien bearbeiten

- **grep** Findet Dateien, die eine bestimmte Zeichenkette enthalten, und gibt die Zeile mit Dateinamen aus. Z.B. findet der folgende Befehl alle Dateien mit der Endung `.c`, die den String `"sqrt"` enthalten: `grep sqrt *.c`
- **cut** Zeigt selektiv eine bestimmte Anzahl an Spalten an. Argumente: `-d` = Trennzeichen (delimiter), `-f` = Spaltenzahl  
Beispiele:
  - `cut -d " " -f 2 test.txt`  
Trennzeichen = Leerzeichen. Gibt die 2. Spalte der Datei an.
  - `cut -d ";" -f 2,5 test.txt`  
Gibt 2. und 5. Spalte an. Diesmal mit `;` als Trennzeichen.
  - `cut -d " " -f 2- test.txt`  
2. bis letzte Spalte der Datei.
  - `cut -d " " -f 2 --complement test.txt`  
Alle Spalten außer die 2.
- `head -5 test.txt` Zeigt erste 5 Zeilen von `test.txt` an.
- `tail -5 test.txt` Zeigt letzte 5 Zeilen von `test.txt` an.
- `wc -l test.txt` Zählt die Anzahl der Zeilen von `test.txt`.
- `paste datei1.txt datei2.txt` Fügt zwei Dateien spaltenweise zusammen und zeigt das Resultat auf der Konsole an.

## Umleitungen > >> | und tr

Man möchte natürlich die Ausgabe nicht nur auf dem Bildschirm haben, sondern eventuell wieder in eine neue Datei schreiben. Dafür gibt es in der Shell die Operatoren: `>`, `>>`, und `|`:

- `cut -d " " -f 2 test.txt > out.txt`  
Schreibt die Ausgabe in die Datei `out.txt` und überschreibt den Inhalt der Datei gegebenenfalls.
- `cut -d " " -f 2 test.txt >> out.txt`  
Hängt die Ausgabe an die Datei `out.txt` an. Es werden in `out.txt` keine Daten überschrieben!

In beiden Fällen würde die Datei `out.txt` erstellt werden, falls sie noch nicht existiert. Während `>` und `>>` in Dateien schreiben, leitet der Pipe-Operator `|` an andere Programme weiter (Beispiel folgt):

- `cat test.txt | tr , . > test1.txt` Die Ausgabe von `cat` (also die Datei selbst) wird an das Programm `tr` weitergeleitet. Dieses ersetzt in der Datei `test.txt` alle Kommata durch Punkte. `>` schreibt das Ergebnis dann in `test1.txt`.
- `tr echo "Das ist ein Test" | tr e E > test1.txt.`
- `tr -s " "` fasst alle aufeinanderfolgenden Leerzeichen zu einem einzigen Leerzeichen zusammen (in die `" "` kann man natürlich auch jedes andere Zeichen schreiben)

## Variablen, echo und ``

- **echo** Die "Druckfunktion" der Shell, z.B.:  
`echo "Hallo Welt"` Ausgabe: `Hallo Welt`
- **Variablen** in der Shell speichern Strings (Zeichenketten). Man **definiert** sie mit: `var="Tom"` wobei jeder beliebige Name anstelle von `var` stehen kann.
- Auf den **Inhalt** der Variablen kann man mit dem Befehl `echo` und dem `$`-Operator zugreifen:  
`echo $var` Ausgabe: `Tom` oder sogar:  
`echo "Hallo $var"` Ausgabe: `Hallo Tom`
- `echo pwd` Ausgabe: `pwd`  
`pwd` liefert den aktuellen Pfad zurück, wird aber von `echo` nicht als Befehl erkannt. Möchte man `echo` (oder einem anderen Programm sagen), dass es den Befehl `pwd` (oder einen beliebigen anderen Befehl) **erst ausführen soll**, setzt man den Befehl in Accent Graves: ```  
`echo `pwd`` Ausgabe (z.B.): `/home/USER/`

## Hilfe!

Du kommst nicht weiter oder verstehst nicht, wie man das Programm, das du gerade aufgerufen hast, tust? (Fast) Jedes Programm bringt Hilfeseiten und Bedienungsanleitungen mit:

- Eine kurze Hilfeseite (nicht immer vorhanden): `PROGRAMMNAME --help`
- Und die gesamte Bedienungsanleitung: `man PROGRAMMNAME`

Ansonsten lohnt sich die folgende Seite bei Fragen rund um Linux und Ubuntu: [wiki.ubuntuusers.de/](http://wiki.ubuntuusers.de/)

Auch die Hochschulbibliothek bietet einen Haufen Bücher über Linux, wo die Shell ausführlich behandelt wird.

## Skripte erstellen

Klingt kompliziert, ist aber kinderleicht!

1. Man nehme eine leere Textdatei und schreibt:

```
#!/bin/bash

ls -la | tr -s " " | cut -d " " -f 1,9
```

Wobei der Beginn `#!/bin/bash` wichtig ist, damit das Skript als solches erkannt wird und der Rest **ganz gewöhnliche Shell-Befehle** sind, wie sie z.B. auch hier im Handout stehen. Frage an dich: Was macht dieses Skript? → Probier's aus!

2. Ist man mit dem Skript fertig, muss man es ausführbar machen. Dafür tippt man (im gleichen Verzeichnis wie das Skript) in die Shell:

```
chmod u+x SKRIPTNAME
```

3. Um das Skript auszuführen (ebenfalls wieder im gleichen Verzeichnis wie das Skript):  
`./SKRIPTNAME`

## Eigene Befehle

Jetzt, wo man eigene Skripte geschrieben hat, kann man diese auch nutzen, um eigene Shell-Befehle zu schreiben. Angenommen das Skript von oben heißt `'lr'`:

1. Das Skript zuerst ausführbar machen  
`chmod u+x lr`
2. Dann verschieben nach `/usr/local/bin/`:  
`sudo mv lr /usr/local/bin/` (Warum mit `sudo`?)

Und voilà: Man kann jetzt überall `'lr'` in die Konsole eintippen und damit das Skript ausführen!

## if und for im Skript

if	for
<pre>if [ \$a -ge 0 ]; then   ANWEISUNGEN fi</pre>	<pre>for i in file1 file2 file3; do   ANWEISUNGEN done</pre>

- Hierbei ist die Bedingung in der `if`-Abfrage, ob  $a \geq 0$  (`ge` = greater or equal).
- Zusätzlich gibt es:  
`-eq ==`, `-ne ≠`, `-le ≤`, `-e file`: wahr, falls "file" existiert
- Die `for`-Schleife hingegen geht über die Dateien `file1 file2` und `file3`. Diese sind in der Variablen `'i'` in jedem Schleifendurchgang gespeichert.
- `for i in *; do ...` würde über alle Dateien im Ordner gehen.

## Der Paketmanager

Zum Schluss etwas Einfaches. Statt über das Software-Center kannst auch über die Shell Programme installieren:

- `sudo apt-get update` Paketlisten aktualisieren.
- `sudo apt-get upgrade` Pakete updaten.
- `apt-cache search PAKET` Paket suchen.
- `sudo apt-get install PAKET` Paket installieren.
- `sudo apt-get purge PAKET` Paket mit allen Konfigurationsdateien deinstallieren.
- `sudo apt-get remove PAKET` Nur das Paket deinstallieren.