

# Crashkurs Kommandozeile

Thomas Werner



**Linux User Group Peine**  
**WWW.LUG-PEINE.ORG**



This work is licensed under the Creative Commons Attribution-ShareAlike 2.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/de/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

# Was ist eine Kommandozeile?

Informatik-Duden:

|                         |
|-------------------------|
| Anwender-Programme      |
| Shell                   |
| Standard-Programme      |
| UNIX-Kernel             |
| Hardware-Schnittstellen |
| Hardware                |

„Der Kommandointerpreter, Shell genannt, ist ein Standard-Programm. Es liest eine Zeile ein und führt die darin angegebenen Kommandos aus.“

# Was kann eine Shell?

- Starten von Kommandos
- Wiederholung und Editieren früherer Kommandos
- Ein-/Ausgabe-Umlenkung
- Verketteten mehrerer Prozesse
- Job-Kontrolle
- Definition der Such-Reihenfolge externer Kommandos
- Scripting

# Starten von Kommandos

- Ein Kommando besteht aus einer Zeile Text

```
ls<ENTER>
```

- Die Zeile kann länger als 80 Zeichen sein
- Beim Scripten / Programmen im aktuellen Verzeichnis muss zur Sicherheit der Pfad angegeben werden:

```
./script
```

```
./programm
```

# Stoppen von Kommandos

- Ctrl + C

Bricht das aktuelle Kommando ab und gibt die Kontrolle zurück an die Shell

- Ctrl + D

Bricht die aktuelle Sitzung in der Shell ab. Das Kommando `exit` hat die gleiche Wirkung

# Mögliche Fehler

- Mit Ctrl + C werden Prozesse „unsanft“ beendet
- Programme können nicht mehr aufräumen
- Wenn es zu Fehlern in der Darstellung kommt, reicht oft ein `clear`
- Bei gravierenden Fehlern:  
    `Ctrl + J`  
    `reset`  
    `Ctrl + J`

# Welche Kommandos gibt es?

- Es gibt etliche Standard-Programme (siehe Liste)
- Im folgenden werden folgende genutzt:
  - cd
  - less
  - ls
  - sort

# Wiederholung früherer Kommandos

- Eingegebene Kommandos werden gespeichert
- Und können mit ↑ wieder abgerufen werden
- Erspart eine Menge Tipp-Arbeit

# Ein-/Ausgabe-Umlenkung



- Prozesse haben Ein- und Ausgabe-Kanäle
- Als Eingaben können Dateien genutzt werden
- Als Ausgabe-Ziel können Dateien genutzt werden

# Ein-/Ausgabe-Umlenkung

- Datei als Eingabe

```
$ sort < unsortiert.txt
```

- Datei als Ausgabe

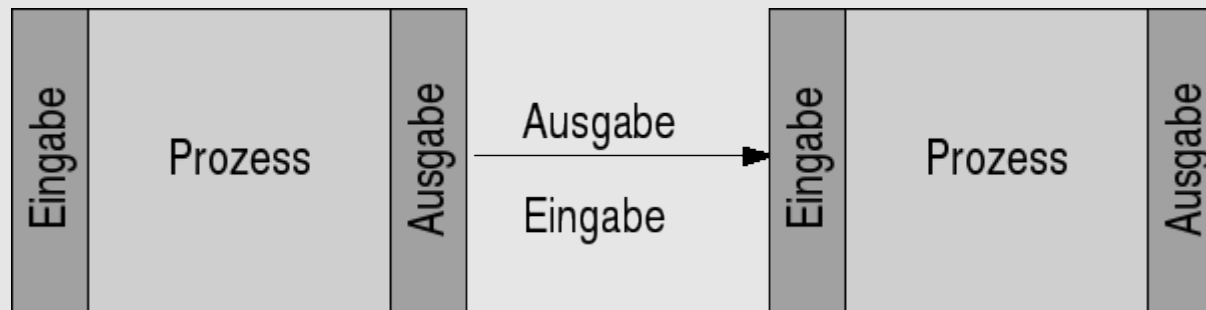
```
$ ls > dir.txt           (ersetzt)
```

```
$ ls >> dir.txt         (fügt an)
```

- Ausgabe von Fehlern in Dateien

```
$ ls > dir.txt 2> err.txt
```

# Verketteten mehrerer Prozesse



- Prozesse können direkt verkettet werden:  
Dabei wird die Ausgabe des ersten Prozesses als Eingabe des zweiten Prozesses genutzt

```
$ programm1 | programm2
```

```
$ ls | sort -r
```

# Job-Kontrolle

- Shells können Programme im Hintergrund bzw. im Vordergrund ausführen
- Dies ermöglicht Multitasking in der Shell
- Vordergrund-Job
  - Läuft interaktiv und belegt den Prompt
- Hintergrund-Job
  - Läuft, belegt aber nicht den Prompt. Weitere Befehle möglich

# Job-Kontrolle

|               |                               |
|---------------|-------------------------------|
| \$ jobs       | Listet die aktuellen Jobs auf |
| \$ programm & | Job im Hintergrund starten    |
| Ctrl + Z      | Aktuellen Job anhalten        |

Angehaltene Jobs können in Vordergrund und Hintergrund ausgeführt werden:

|             |                            |
|-------------|----------------------------|
| \$ bg <JOB> | Job im Hintergr. ausführen |
| \$ fg <JOB> | Job im Vordergr. ausführen |

# Definition der Such-Reihenfolge

- Shells verfügen über so genannte Umgebungsvariable
- Eine dieser Variablen ist `PATH`
  - Besteht aus einer Liste von Verzeichnissen
  - Verzeichnisse durch `:` getrennt
- Diese Verzeichnisse werden nach Programmen und Scripten durchsucht
- Dabei wird die Reihenfolge der Pfade beachtet

# Anpassen der Such-Reihenfolge

Anzeige des Such-Pfades:

```
echo $PATH
```

Erweitern des Such-Pfades:

```
PATH=$PATH:/home/tom/bin
```

Dauerhafte Anpassung:

Bearbeiten der Bash-Einstellungen unter ~/.bashrc

# Scripting

- ...ist einfach!
- Shell-Kommandos werden in einer Datei gespeichert und können mit einem Aufruf gestartet werden
- Umgebungsvariable können ausgewertet werden
- ... und eigene Variable definiert werden
- Zusätzlich stehen einige Konstrukte einer Programmiersprache bereit

# Beispiel: Sleep-Timer

Problem:

Der PC wird zum Abspielen von Musik genutzt.

Nach 30 Minuten soll sich der PC ausschalten.

Dabei soll er nicht piepen.

```
modprobe -r pcspkr
```

```
shutdown -h +30
```

# Script 1

Datei qshutdown30:

```
#!/bin/bash                                #Sha-Bang
#Mein erstes Script                        #Kommentar
modprobe -r pcspkr                          #Kommandos
shutdown -h +30
```

Aufruf : \$ qshutdown30

Problem : Zeitangabe ist fest kodiert. Man braucht etwa  
105 Scripte für alle möglichen Zeit-Angaben

# Script 2

Datei qshutdown:

```
#!/bin/bash
modprobe -r pcspkr
shutdown -h +$1          #Parameter 1
```

Aufruf : \$ qshutdown 30

Problem : Was passiert, wenn Parameter nicht da ist?

# Script 3

```
#!/bin/bash
if [ ! $# = 1 ]           #if, Anzahl
then                       #then
    echo "Aufruf: $0 Minuten" #Param0
    exit 1                 #Fehler
fi                          #fi
modprobe -r pcspkr
shutdown -h +$1
```

Die Anzahl der Parameter wird geprüft. Bei falscher Anzahl wird eine Fehlermeldung ausgegeben.

# Script 4

## Prüfen des Benutzer-Namens

```
...  
# Check: Bin ich root?  
user=`whoami`          #Var=Ausgabe  
if [ ! "$user" = "root" ] #NOT, Vergleich  
then  
    echo "Sorry $user, das darf nur root"  
    exit 3  
fi  
...
```

Der Benutzername wird durch einen Aufruf des Programmes `whoami` herausgefunden.

# Script 5

expr wertet mathematische und reguläre Ausdrücke aus

...

```
# Check: Ist der Parameter numerisch?
```

```
length=`expr length $1`
```

```
number=`expr match "$1" '[0-9]*'`
```

```
if [ ! $length = $number ]
```

```
then
```

```
    echo "Aufruf: $0 Minuten"
```

```
    exit 2
```

```
fi
```

...

# Script Endversion

```
#!/bin/bash

# Check: Anzahl der Parameter richtig?
if [ ! $# = 1 ]
then
    echo "Aufruf: $0 Minuten"
    exit 1
fi

# Check: Ist der Parameter numerisch?
length=`expr length $1`
number=`expr match "$1" '[0-9]*'`
if [ ! $length = $number ]
then
    echo "Aufruf: $0 Minuten"
    exit 2
fi

# Check: Bin ich root?
user=`whoami`
if [ ! "$user" = "root" ]
then
    echo "Sorry $user, aber das darf nur
root"
    exit 3
fi

# Alles OK - wir koennen loslegen
echo ""
echo "ACHTUNG!"
echo "Rechner wird ausgeschaltet"
echo "Zum Abbrechen Strg + C druecken"
modprobe -r pcspkr
shutdown -h +$1
```

# Links und Literatur

## Bücher:

- [Linux - kurz und gut](#)
- [Ubuntu GNU/Linux](#)
- [Wie werde ich UNIX Guru](#)

## Links:

- [Advanced Bash-Scripting Guide](#)
- [WikiBook: Linux-Kompendium](#)
- [WikiBook: Bourne Shell Scripting](#)

Danke für die Aufmerksamkeit