R|R|Z|N

Leibniz Universität Hannover

Linux-Einführung

Mark Heisterkamp

heisterkamp@rrzn.uni-hannover.de

1. – 2. Juni 2010

Inhalt

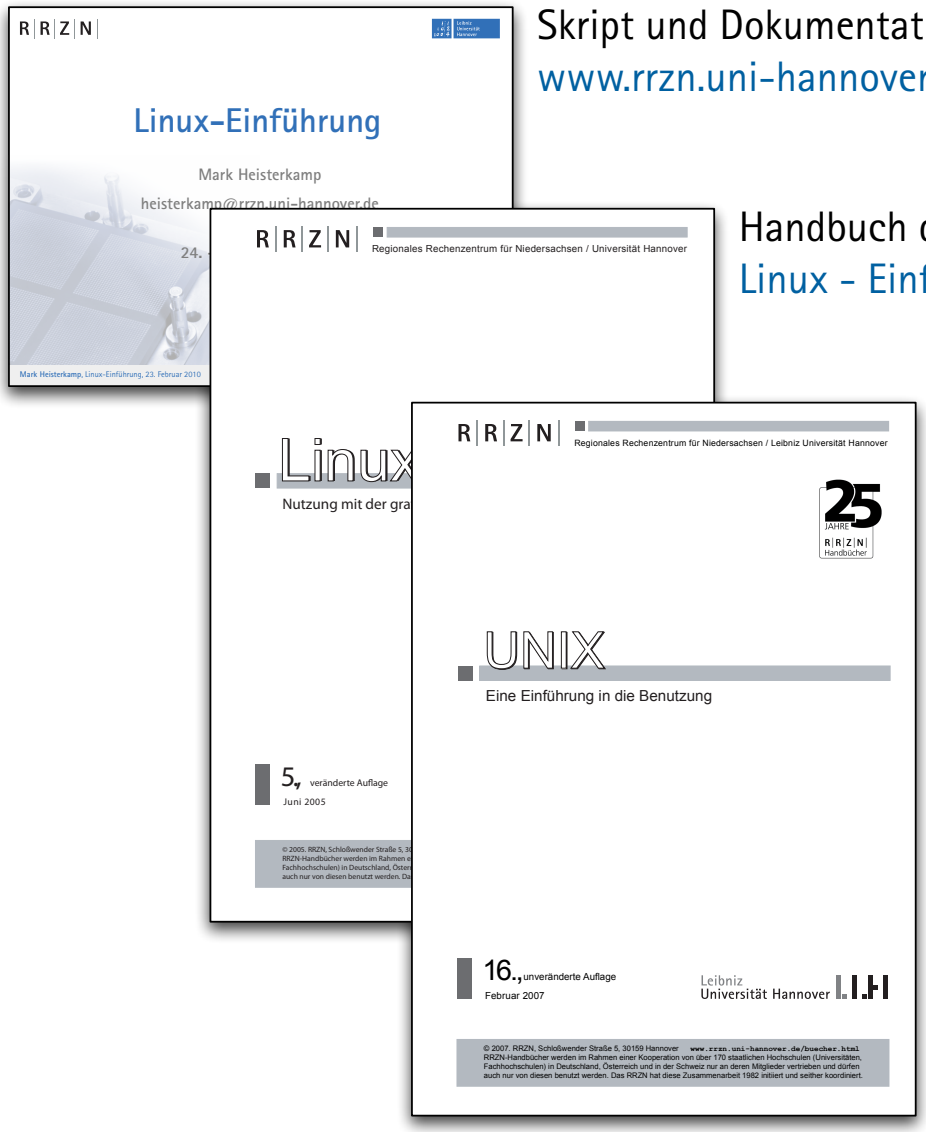
Dieser Kursus richtet sich an Linux-Anfänger.

Ziel ist es, grundlegende Linux-Kenntnisse für die Benutzung eines Linux-Desktops zu vermitteln.

Folgende Themen werden behandelt:
Aufbau des Dateisystems, Bearbeiten und Editieren von Dateien, Ein-Ausgabe-Umlenkung, Linux-Shells, verschiedene Hilfemöglichkeiten.

Ferner werden Linux-Programme zur Kommunikation mit anderen Rechnern (SSH, FTP ...) vorgestellt.

Literatur



Skript und Dokumentation zum Kurs
www.rrzn.uni-hannover.de/unixgk.html

Handbuch des RRZN
Linux - Einführung

Handbuch des RRZN
Unix - Einführung

Rechner und Betriebssystem

- Das BIOS (Basic Input Output System) hält einige grundlegende Informationen über die Rechnerkomponenten vor.
- Der Rechner »weiß« bis auf die BIOS-Informationen nichts über sich.
- Das Betriebssystem kann die Komponenten und den Rechner interaktiv (d. h. unter Einflussnahme des Benutzers) verwalten.
- In der Form der Verwaltung dieser Rechnerkomponenten unterscheiden sich Betriebssysteme voneinander.

Entwicklung...

- **60er–70er:** Großrechner im Batch-Betrieb, d. h. alle Programme wurden nacheinander ausgeführt (Stapelverarbeitung), ohne den Benutzer in den Programmablauf zu einzubeziehen.
- **70er:** Workstations wurden entwickelt, und der Bedarf nach dialogorientierten Betriebssystemen nahm zu. Die Entwicklung von UNIX nahm damit ihren Anfang.
- **80er:** Der PC erschien mit dem Betriebssystem DOS, das der niedrigen Leistungsfähigkeit des PCs Rechnung trug. Für ein UNIX oder ein ähnlich mächtiges Betriebssystem war der PC zu klein.

...

. . . Entwicklung

- **Ende 80er:** Der PC wurde leistungsfähig genug, um eine grafische Benutzeroberfläche (Windows) verwalten zu können. UNIX- und Apple-Rechner verfügten bereits über eine solche Oberfl äche.
- **90er:** Die PCs wurden so leistungsfähig, dass sie ein vollwertiges UNIX verkraften konnten. Die Entwicklung von Linux begann.
- **Heute:** PCs auf Linux-Basis konkurrieren mit »echten« UNIXen und werden auch in empfindlichen EDV-Bereichen eingesetzt (Netzwerke, Server . . .).

Geschichte von UNIX...

- 1969 begann Ken Thompson an den Bell Laboratories die Entwicklung von UNIX. Er wollte die Bedienung von Rechnern benutzerfreundlicher gestalten, gleichzeitig sollte das Betriebssystem relativ kompakt sein, so dass es auch auf einer Workstation laufen konnte. Außerdem sollte das Betriebssystem unabhängig von der Rechnerarchitektur sein.
- Die erste Variante war noch in Maschinensprache geschrieben. Wegen der Portierung auf andere Rechner erfand er die Programmiersprache B.
- Dennis Ritchie entwickelte B zur Programmiersprache C weiter.

...

...Geschichte von UNIX

- 1971 wurde UNIX zum ersten mal komplett (bis auf den Kernel) in C geschrieben. Die Quellcodes dieser Variante waren frei erhältlich, und jeder konnte somit das Betriebssystem nach Belieben weiterentwickeln.
- Es entwickelten sich viele Dialekte, die teilweise inkompatibel waren.
- Die Entwicklung war schnell und der Bedarf nach UNIX hoch. Eine Standardisierung wurde notwendig.
- 1991 wurde der Quasi-Standard Unix System V, Release 4 (SVR4) entworfen, an dem sich alle kommerziellen UNIXe anpassen sollten.
- Zwar unterschieden sich die UNIXe noch immer, aber nur geringf ügig, und der Umstieg von einer Variante zur anderen wurde problemlos.

Abgrenzung zu anderen Betriebssystemen

Gemeinsamkeiten:

- Grafische Benutzeroberfläche mit Maussteuerung.
- Multitasking

Unterschiede:

- Linux besitzt eine sehr zuverlässige Prozesskontrolle.
- Wegen seiner wissenschaftlichen Ausrichtung besitzt Linux einen gro ßen Umfang an Entwicklungs- und Programmierwerkzeugen.
- Linux besteht aus vielen kleinen nützlichen Programmen, die nur eine spezielle Aufgabe sehr effektiv erledigen.
- Mittels sogenannter Skriptsprachen kann man die kleinen Programme zu leistungsfähigen »großen« Programmen zusammenbauen.
- Viele Prozesse können unter Linux mittels Tastatur gesteuert werden.

Vor- und Nachteile von UNIX

Nachteile:

- Gewöhnungsbedürftige Bedienung.
- Möglicherweise komplexe Installation.
- Unter Umständen komplexe Nachinstallation von zusätzlichen Programmen.
- SVR4 ist kein 100%er Standard.

Vorteile:

- Sehr hohe Stabilität und Verfügbarkeit.
- Durch Multitasking und Multiuserbetrieb sind komfortable Nutzung und Administration möglich.
- Umfangreiche Standardinstallation
- Ausgereifte Entwicklungs- und Programmierwerkzeuge.
- In seiner Erscheinungsform als Linux ist UNIX kostenlos.

Erste Schritte ...

Anmeldung am System:

- Benutzername (bzw. User-ID, Login-Name)
- Passwort

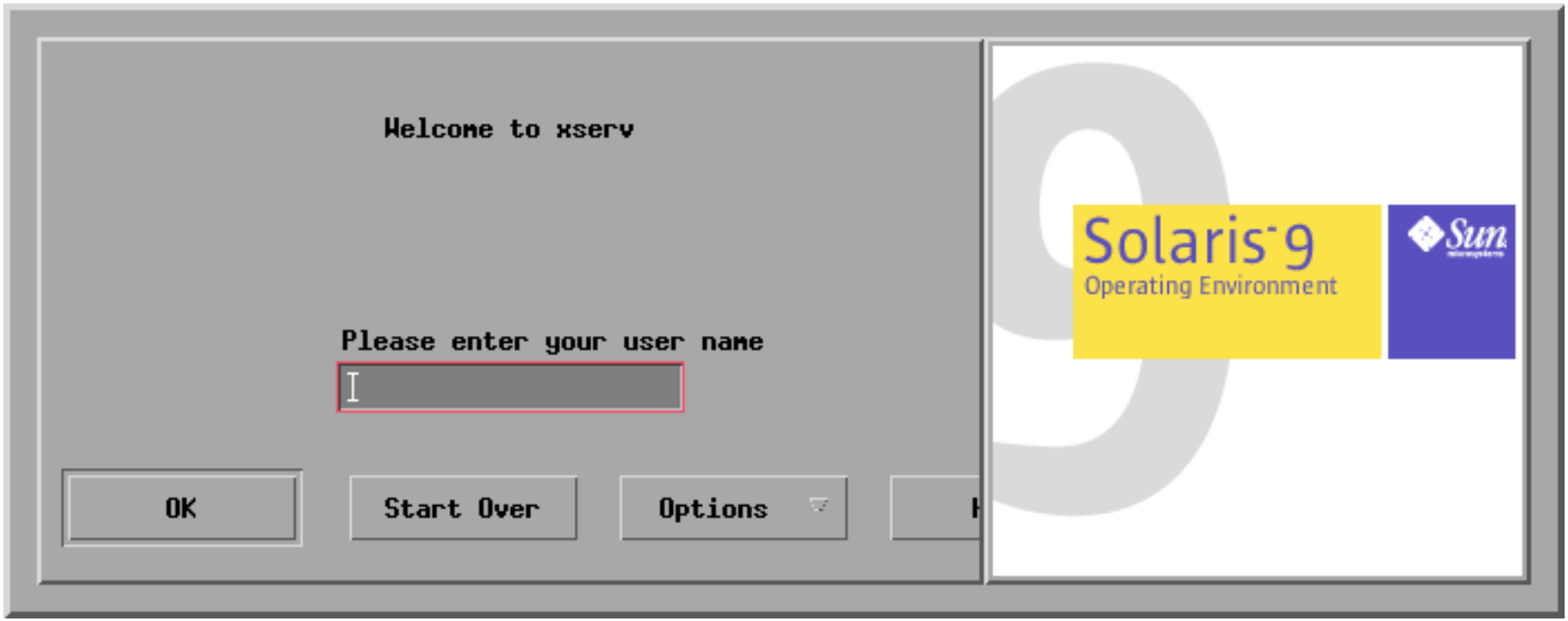
Zwei Arten der Anmeldung:

- grafisch
- textorientiert (Konsole)

Grafische Anmeldung unter Linux (KDE 3.5)

Grafische Anmeldung unter Mac OS X

Grafische Anmeldung auf xserv



Textorientierte Anmeldung (Konsole)

```
Debian GNU/Linux testing/unstable pc225h tty2
pc225h login: mheiste
Password: |
```

Benutzername und Passwort

- Groß- und Kleinschreibung
- unter Umständen keine Bildschirmausgabe

Erste erfolgreiche Anmeldung

- Man befindet sich entweder auf der grafischen Benutzeroberfläche mit Maussteuerung oder auf der Konsole.
- Um im System arbeiten zu können braucht man einen Prompt, d. h. die Möglichkeit, Befehle per Tastatur an den Rechner geben zu k önnen.
- Auf der Konsole hat man bereits einen Prompt.
- Auf der grafischen Oberfläche muss noch ein sogennantes XTerminalfenster (X-Terminal, XTerm) geöffnet werden.

Befehlseingabe

```
Befehl Optionen Parameter
```

Bestimmte Optionen bzw. Parameter können in Abhängigkeit vom Befehl optional oder obligatorisch sein. In der Darstellung wird das durch folgende Schreibweise erreicht:

```
Befehl [optionale Angaben] <obligatorische Angaben>
```

Dabei werden die Klammern nicht mit eingegeben!

Beispielsweise bedeutet die Angabe von

```
mkdir [Option] <Verzeichnis>
```

dass dem Befehl `mkdir` einige optionale Angaben folgen können, aber die Angabe eines Verzeichnisses folgen muss.

ändern des Passwortes

Mit dem Befehl

```
passwd
```

kann man sein Passwort ändern. Nach einmaliger Eingabe des (noch) aktuellen Passwortes wird man zweimal nach der Eingabe des neuen Passwortes gefragt.

Manpages

Eine wichtige Funktion sind die sogenannten Manpages. Durch Eingabe von

```
man <Befehl>
```

kann man sich zu jedem Befehl eine englischsprachige Hilfefunktion aufrufen (unter Linux teilweise auch auf deutsch). Der Hilfetext kann durch die »b«, Leer- und die Returntaste rauf- und runtergescrollt werden.

Beendet wird die Hilfefunktion durch Druck auf die Taste »Q«.

Ein paar nützliche Befehle

`date` Ausgabe des aktuellen Datums und der Uhrzeit.

`who` Ausgabe aller am System eingeloggten Benutzer.

`whoami` Ausgabe des eigenen Benutzernamens.

`echo` Der nach `echo` folgende Text wird am Bildschirm ausgegeben.

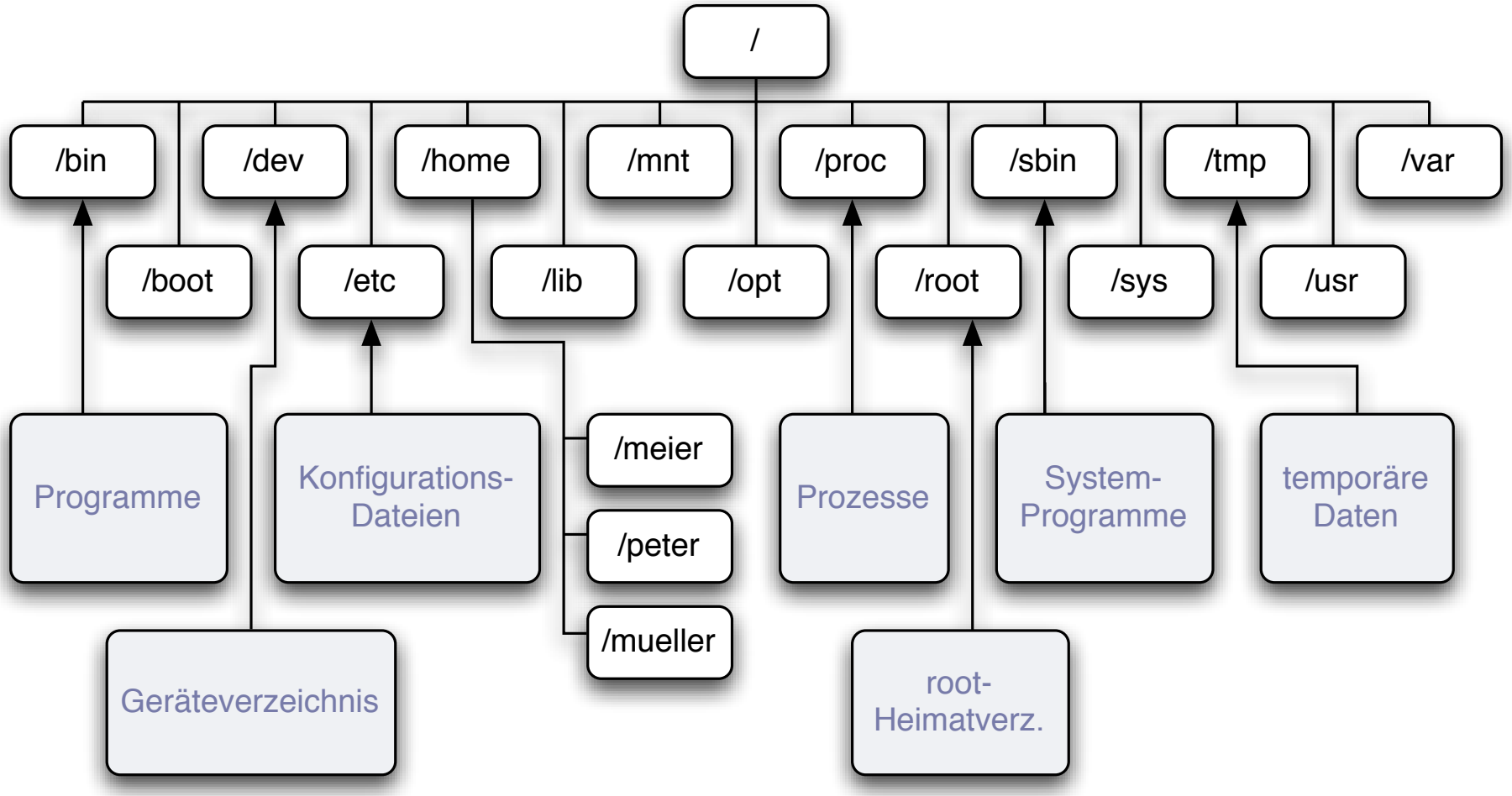
`more` Der Inhalt der nach `more` genannten Datei wird auf den Bildschirm ausgegeben.

`less` Vgl. `more`, die Cursorsteuerung ist allerdings komfortabler mit den Cursortasten möglich.

Dateiverwaltung

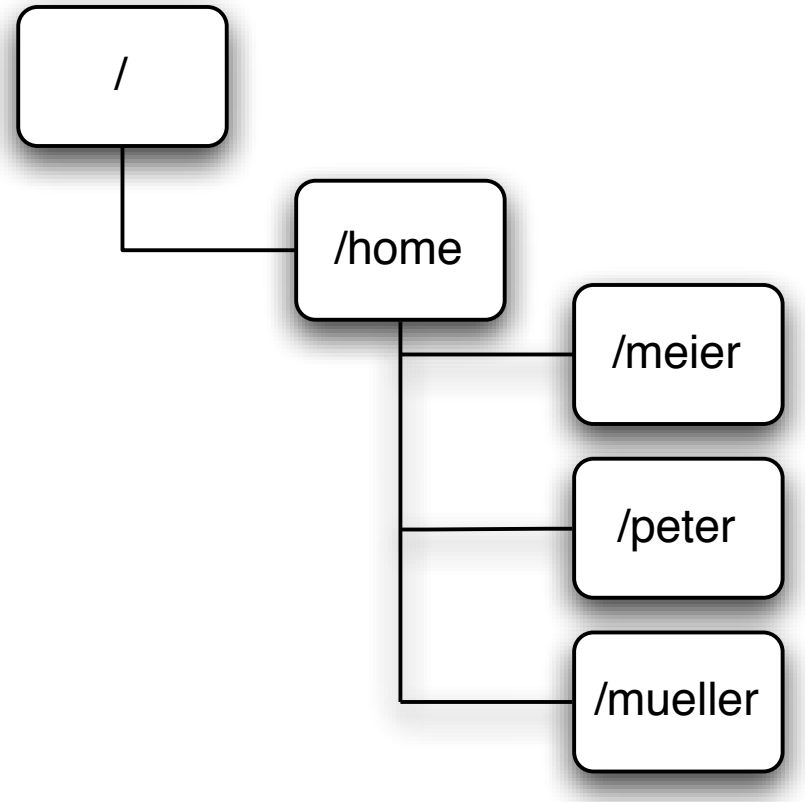
- hierarchisch (Baumstruktur)
- ausgehend von der sogenannten Wurzel »/«
- Jedes Verzeichnis enthält mind. 2 Einträge: ».« und »..«
- ».« steht für das Verzeichnis selbst.
- »..« steht für das übergeordnete Verzeichnis.
- Absolute und relative Pfade sind dadurch möglich.

Exemplarischer Aufbau eines Dateibaums



Homeverzeichnis

Jeder Benutzer unter Linux hat einen eigenen Raum auf der Festplatte, der ihm zugeteilt wurde, das sogenannte Homeverzeichnis.



Verzeichnisinhalt

```
ls [Optionen] [Zielverzeichnis]
```

Optionen:

`--color=auto` farbige Darstellung

`-l` listenartige Ausgabe mit Zusatzinformationen

`-a` es werden alle Dateien (auch versteckte) angezeigt

`-r` Umgedrehte Reihenfolge

`-t` Nach Größe sortiert

`-h` Dateigrößen besser lesbar (»1M' statt '1024«)

Verzeichniswechsel

```
cd [Zielverzeichnis]
```

Ausgabe des aktuellen Pfades

```
pwd
```

Löschen

rm [Optionen] <Datei/Verzeichnis>

Optionen:

- `-r` Notwendige Option für das rekursive Löschen von Verzeichnissen (samt Inhalt).
- `-i` Jeder Löschvorgang muss bestätigt werden (interrogation)
- `-f` Keine Nachfragen (force).

Anlegen eines Verzeichnisses

```
mkdir [Optionen] <Verzeichnis>
```

Optionen:

- `-m` Zugriffsrechte werden mitgesetzt.
- `-p` Es werden auch oberhalb liegende, möglicherweise noch nicht existente Verzeichnisse angelegt.

Metazeichen bzw. Wildcards

Metazeichen sind Platzhalter. Sie können überall dort eingesetzt werden, wo Datei- oder Verzeichnisnamen angegeben werden.

`?`	ein einzelnes beliebiges Zeichen.
`*`	beliebig lange Folge beliebiger Zeichen.
`[...]`	Auswahlliste einzelner Zeichen an dieser Stelle
`[!...]`	Auswahlliste einzelner Zeichen, die nicht an dieser Stelle auftauchen dürfen.

Beispiel für Metazeichen ...

Folgende Dateien liegen zum Beispiel in einem Verzeichnis:

```
myfile.txt
brief.txt
bericht.txt
protokoll-a.doc
protokoll-b.doc
```

...und die Entprechung der Metazeichen

`protokoll-?.doc`	`protokoll-a.doc` `protokoll-b.doc`

`*.txt`	`myfile.txt`
	`brief.txt`
	`bericht.txt`

`b*`	`brief.txt` `bericht.txt`
`[abc]* oder [a-c]*`	`brief.txt` `bericht.txt`

`[!p]*`	`myfile.txt`
	`brief.txt`
	`bericht.txt`

Prozessverwaltung

- Prozesse sind gestartete Programme.
- Jeder Prozess hat einen Eigentümer.
- Jeder Prozess hat einen Mutterprozess und ist Kindprozess eines solchen Mutterprozesses.
- Prozesse können ruhen, aktiv sein, angehalten sein, auf die Festplatte ausgelagert oder speicherresistent sein, im Vordergrund oder im Hintergrund laufen, und sie können beendet sein, ohne den Mutterprozess über die Beendigung verständigt zu haben.
- Die Prozesse sind mittels der PID (Process Id entification Number) durchnummeriert.
- Die Prozesse sind in einer Baumhierarchie angeordnet.

Wichtige Prozesse

- Mutter aller Prozesse ist der Prozess `init` mit der PID 1. Wird er beendet, wird der Rechner runtergefahren.
- Die Anmeldung eines Benutzers entspricht dem Login-Prozess, der dem neuen Benutzer gehört, und der die Mutter aller von diesem Benutzer gestarteten Prozesse ist.
- Es gibt eine Reihe systemrelevanter Prozesse, die im Hintergrund laufen (sogenannte Dämonen). Sie übernehmen Aufgaben wie die Netzwerkanbindung, die Druckausgabe, Darstellung der graphischen Benutzeroberfläche etc.

Das Kommando top

```
load averages:  0.05,  0.03,  0.03
15:03:07
95 processes:  94 sleeping, 1 on cpu
CPU states: 99.7% idle,  0.0% user,  0.3% kernel,  0.0% iowait,  0.0%
swap
Memory: 1664M real, 1323M free, 72M swap in use, 3680M swap free

   PID USERNAME THR PRI NICE  SIZE   RES STATE    TIME    CPU COMMAND
 28970 root       1  59    0 7464K 3176K sleep    0:00  0.28% sshd
 28974 mheiste    1  59    0 1936K 1320K sleep    0:00  0.23% ksh
 29002 mheiste    1  59    0 2240K 1272K cpu/1    0:00  0.18% top
 27896 roll       1  59    0 7800K 2712K sleep    0:30  0.17% sshd
 28946 roll       1  59    0 4408K 3488K sleep    0:00  0.12% xterm-220
 28963 roll       1  59    0 3656K 2560K sleep    0:00  0.11% ssh
   391 root      20  59    0 5248K 4104K sleep    3:23  0.04% nscd
 28972 mheiste    1  59    0 7456K 2224K sleep    0:00  0.02% sshd
   360 root       2  59    0 4272K 2760K sleep   87:22  0.01% automountd
   557 root       7  59    0 2480K 2080K sleep   17:46  0.00% mibiisa
   547 root       1  59    0 3592K 1624K sleep    1:59  0.00% sshd
  4529 noack      1  59    0 2304K 1512K sleep    1:30  0.00% fetchmail
   307 root       1  59    0 2552K 1632K sleep    0:56  0.00% rpcbind
```

Wichtige Prozessdaten bei `top`

PID	Process Identification Number
USERNAME	Eigentümer des Prozesses
SIZE	Gesamtgröße des Prozesses
RES	Größe des Prozesses im Arbeitsspeicher
STATE	Prozesszustand
TIME	Laufzeit des Prozesses
CPU	Prozentuale Angabe der CPU-Auslastung durch den Prozess
COMMAND	Kommando, das den Prozess gestartet hat

Das Kommando ps

```
ps [Optionen]
```

Mit ps können die Prozessdaten auf der Konsole ausgegeben werden. Durch das Setzen von Optionen kann die Ausgabe sehr flexibel angepasst werden.

ps eignet sich deshalb im Gegensatz zu top sehr gut f ür den Einsatz in Skriptsprachen.

ps bietet abhängig vom System sehr unterschiedliche Optionen, deren Nutzung man deshalb fast immer nur durch Studium der dazugehörigen Manpage herausfindet.

Löschen eines Prozesses

Ist ein Prozess / Programm hängengeblieben, so kann man dieses Programm notfalls von Hand beenden. Dazu muss man zunächst Eigent ümer des zu beenden Prozesses sein, und man muss seine PID kennen. Das Kommando lautet:

```
kill [-Killsignal] <PID>
```

Wird das `Killsignal` weggelassen, so wird der Prozess aufgefordert, sich ordentlich zu beenden. Bleibt der Prozess dennoch in der Prozesstabelle erhalten, so kann man ihn zur »Aufgabe« zwingen, indem man das `Killsignal` 9 verwendet:

```
kill -9 <PID>
```

Die Shell

- Die Shell ist ein sogenannter Kommandointerpreter.
- Sie liefert den Prompt und ist auch an dessen Darstellung m öglicherweise erkennbar.
- Sie ist die eigentliche Schnittstelle zum System auf der Kommandozeile.
- Alle bisherigen Befehle wurden bereits mittels der so genannten Bash (Bourne Again Shell) eingegeben!
- Die Shell legt sich wie eine Muschel um den Systemkern, interpretiert die eingegebenen Befehle und reicht ihre Ergebnisse an den Systemkern weiter.
- Sie bietet eingebaute Kommandos, die die Kommandoeingabe teilweise extrem vereinfachen.

Die verschiedenen Shells

Bourne-Shell `sh` älteste Shell unter Linux. Benannt nach ihrem Entwickler Steven R. Bourne.

C-Shell `csh` Weiterentwicklung der Bourne-Shell mit grö ßerem Funktionsumfang, aber teilweise inkompatibel.

Korn-Shell `ksh` Erweiterung der Bourne- und C-Shell bei voller Kompatibilität zu beiden. Benannt nach ihrem Entwickler David Korn. RRZN-Standard.

Bourne-Again-Shell `bash` Massive Erweiterung der drei oberen Shells bei voller Kompatibilität. Unter Linux sehr weit verbreitet.

Die Eingabe-Prompts der Shells

Bourne-Shell `unics:/home/zzzheis:!$`	Der Rechnername `unics` und das aktuelle Verzeichnis wird angezeigt.
C-Shell `unics%`	Lediglich der aktuelle Rechner wird angezeigt.
Korn-Shell `unics::138$`	`unics` ist der Rechnername, und die Nummer gibt die aktuelle Anzahl der Kommandozeilen aus, die man bisher abgearbeitet hat. Löscht man den Inhalt der Datei `.sh_history`, so fängt die Nummerierung wieder bei 1 an.
Bourne-again-Shell `unics:/home/zzzheis:!$`	Darstellung entspricht der der Bourne-Shell.

Verlassen einer Shell

Um eine Shell wieder zu verlassen, benutzt man das Kommando

```
exit
```

Gibt man dieses Kommando in der Login-Shell ein, so meldet man sich vom System wieder ab.

Jedes XTerminal, das geöffnet wird, startet bereits mit der Standard- Shell des Systems (im Falle des RRZN ist das die Korn-Shell). Gibt man hier `exit` ein, so schließt man dieses Fenster wieder.

Jede Shell hat einen ganzen Satz eingebauter Kommandos, die den Umgang mit dem System erleichtern sollen. Alle verfügbaren Kommandos kann man über die Manpage der jeweiligen Shell herausfinden.

Shell-Funktionen

Drei nützliche Funktionen der Bash (auch die Korn-Shell hat sie) kennen wir schon:

- History-Funktion
  das heißt mittels der Cursortasten kann man bereits eingebene Kommandos wieder abrufen und ggf. modifizieren.
- Wildcards bzw. Metazeichen
- Farbausgabe
  mittels der `--color=auto` Option des Kommandos `ls`.

Weitere Shell-Funktionen

- Automatische Vervollständigung von Namen:
  Wann immer man einen Befehl oder einen Datei- bzw. Verzeichnisnamen eintippt, kann man zu jedem beliebigen Zeitpunkt durch Druck auf die TAB-Taste (Bash) oder durch zweimaliges Drücken der ESC-Taste (Korn-Shell) versuchen, das begonnene Wort vervollst ändigen zu lassen.
- Scrollen im Terminalfenster: Durch die bisherigen Ausgaben in einem Terminalfenster kann man durch gleichzeitiges Drücken der Shift- und der PgUp- bzw. PgDn-Taste rauf- und runterscrollen.
- Mehrere Kommandos nacheinander ausführen: Bei der Eingabe von Kommandos kann man mehrere Befehle nacheinander ausführen lassen, indem man diese Kommandos durch Semikolons trennt.
- Selbstständige Prozesse: Durch Eingabe des Zeichens »&« nach einem Befehl und mit einem Leerraum dazwischen, wird der Befehl unabhängig von der aufrufenden Shell. Sie kann also weitere Befehle ausführen.

Umgebungsvariablen der Shell

Die Umgebungsvariablen der Shell teilen ihr mit, wo sie bestimmte Informationen finden kann, welche Geräte sie nutzen kann und wie die Umgebung der Shell auf dem jeweiligen System definiert ist. Alle gesetzten Umgebungsvariablen kann man sich mit dem Kommando

```
env
```

ansehen.

Setzen einer Umgebungsvariable

Die Syntax lautet in dem Fall:

```
export <UMGEBUNGSVARIABLE>=<NeuerWert>
```

Beispielsweise kann man die Umgebungsvariable `PRINTER` auf den Wert `hp4_rz_b219` setzen, indem man eingibt:

```
export PRINTER=hp4_rz_b219
```

Permanentes Setzen einer Umgebungsvariable

Mit der `export`-Funktion wird eine Umgebungsvariable nur für die Dauer der Anmeldung an der momentanen Shell gesetzt. Soll sie auch bei der nächsten Anmeldung und in allen geöffneten Terminalfenstern einen bestimmten Wert besitzen, so muss im Falle der Bash die Datei

```
.bashrc
```

im eigenen Homeverzeichnis entsprechend geändert werden. Dort muss einfach eine Zeile hinzugefügt werden, die die entsprechende `export`-Anweisung enthält.

Löschen einer Umgebungsvariable

Um den Inhalt einer Umgebungsvariable zu löschen, benutzt man den Befehl

```
unset <UMGEBUNGSVARIABLE>
```

Inhalt einer Umgebungsvariable ansprechen

Setzt man vor den Namen einer Umgebungsvariable das Dollarzeichen »$«, so referenziert man deren Inhalt. Angenommen die Umgebungsvariable `PATH` sei auf das eigene Homeverzeichnis gesetzt, ihr Inhalt sei also beispielsweise

```
/home/zzzzheis
```

Die Umgebungsvariable `HOME` kann man direkt auf den Wert von `PATH` setzen, indem man das Dollarzeichen benutzt:

```
export HOME=$PATH
```

Ausgabe einer Umgebungsvariable

Mittels der Eingabe von

```
echo $<UMGEBUNGSVARIABLE>
```

kann man sich den Inhalt einer bestimmten Umgebungsvariable anzeigen lassen.

Wichtige Umgebungsvariablen

PATH	Durch einen Doppelpunkt getrennte Liste aller Pfade, die bei Aufruf eines Kommandos durchsucht werden.
HOSTNAME	Name des Rechners.
PRINTER	Name des Standarddruckers.
PS1	Prompt-String der Login-Shell.
HOME	Pfad zum Homeverzeichnis.
USER	Benutzername der angemeldeten Person.
	...

Ein- / Ausgabeumlenkung und Pipes

Alle vier vorgestellten Shells unterstützen die Eingabeumlenkung und die Kommandoverknüpfung. In der Darstellung beziehen wir uns hier auf die Syntax der Korn-Shell und der Bash.

Ein- / Ausgabeumlenkung: Man kann die Ausgabe eines Prozesses umlenken. das heißt die Ausgabe kann auf einem anderen Gerät stattfinden als es normalerweise der Fall ist. So kann man beispielsweise die Ausgabe des `ls`-Kommandos in eine Datei umlenken, so dass das Inhaltverzeichnis nicht auf dem Bildschirm angezeigt wird, sondern in einer Textdatei gespeichert ist. Ebenso k önnte man die Ausgabe auch direkt auf einen Drucker umleiten.

Pipes: Man kann die Ausgabe eines Prozesses direkt als Eingabe in einen anderen Prozess weiterleiten. So könnte man beispielsweise die Ausgabe einer Suchfunktion direkt an einen Befehl weiterleiten, der die Ergebnisse der Suchfunktion nochmals nach bestimmten Merkmalen filtert.

- `>` Die Ausgabe des links stehenden Befehles wird nach rechts umgelenkt.
- `»` Die Ausgabe des links stehenden Befehles wird rechts angehängt.

Zur Eingabeumlenkung verwendete Zeichen:

- `<` Der rechts stehende Inhalt wird nach links umgelenkt.
- `«` Die Eingabe wird nach links solange umgelenkt, bis die rechts stehende Zeichenkette erreicht wird.

Beispiel für > und »

die Eingabe von

```
ls > ls-Datei.txt
```

leitet die Ausgabe des `ls`-Kommandos in die Datei `ls- Datei.txt` um. Falls die Datei noch nicht existiert, wird sie erzeugt, und falls sie existiert, wird sie überschrieben.

```
ls » ls-Datei.txt
```

hängt die Ausgabe des `ls`-Kommandos and den Inhalt der Datei `ls-Datei.txt` an. Existiert `ls-Datei.txt` noch nicht, so wird sie erzeugt.

Beispiel für < und «

die Eingabe von

```
cat < ls-Datei.txt
```

gibt die zuvor erzeugte Textdatei `ls-Datei.txt` auf dem Bildschirm aus. Und

```
cat « ende
```

sammelt solange Eingabedaten von der Standardeingabe (Tatstatur) bis die Zeichenkette »ende« eingegeben wurde.

Kommandoverknüpfung (Pipe)

Auf die Ausgabe eines Kommandos kann man ein weiteres Kommando anwenden, das die Ausgabedaten des anderen Kommandos auswertet. Durch den Trennstrich »|« wird das syntaktisch dargestellt:

```
Kommando 1 | Kommando 2
```

Die Ausgabe von `Kommando 1` wird als Eingabe (Parameter) von `Kommando 2` genutzt.

Beispiel zur Pipe

Das Kommando `grep` durchsucht eine angegebene Datei nach einer Zeichenkette und gibt bei Erfolg die Zeile(n) der Datei aus, die die gesuchte Zeichenkette enthält.

Durch Eingabe von

```
ls | grep pdf
```

erhält man als Ausgabe alle Zeilen des Inhaltverzeichnisses, die die Zeichenkette »pdf« enthalten.

Nützliche Befehle

- `clear`
- `grep`
- `wc`
- `find`
- `cp`
- `mv`

Das Kommando `clear`

`clear` löscht den aktuellen Inhalt eines Terminalfensters.

Das Kommando grep

Die Syntax lautet:

```
grep <Zeichenkette> <Datenquelle>
```

`grep` durchsucht die `Datenquelle` (in der Regel ein Dateiname) nach der `Zeichenkette` und gibt alle Zeilen aus, in denen Die `Zeichenkette` vorkommt.

Das Kommando `wc`

Die Syntax lautet:

```
wc [Option] <Datenquelle>
```

`wc` zählt Worte, Zeilen oder Buchstaben in einer oder mehreren `Datenquellen`. Möchte man mehrere `Datenquellen` angeben, so trennt man ihre Angabe durch die Leertaste. Die Optionen sind:

- `-m` Zeichen werden gezählt.
- `-C` Zeichen werden gezählt.
- `-l` Zeilen werden gezählt.
- `-w` Worte werden gezählt.

Das Kommando `find`

Die Syntax lautet:

```
find <StartVerzeichnis> [Option] <Suchbegriff>
```

Die wohl wichtigste `Option` ist »`-name`«. Wird der Suchbegriff (Wildcards sind erlaubt!) dann als Zeichenkette in Anf ührungsstriche gesetzt, so werden alle Dateien / Verzeichnisse ab dem angebenen Startverzeichnis nach dem Suchbegriff durchsucht.

Kopieren

```
cp [Optionen] <Quelle> <Ziel>
```

Mit den Optionen:

- `-a` Es werden alle Dateien, Unterverzeichnisse und auch Zugriffsrechte mitkopiert.
- `-r` Rekursives kopieren von Dateien und Verzeichnissen

`Quelle` und `Ziel` können sowohl Verzeichniss als auch Datei sein.

Verschieben

```
mv [Optionen] <Quelle> <Ziel>
```

Dieses Kommando eignet sich vor allem zum umbenennen. `Quelle` und `Ziel` können sowohl Verzeichniss als auch Datei sein.

Zugriffsrechte auf Dateien und Verzeichnisse

- Linux ist ein Mehrbenutzersystem.
- Dateien und Verzeichnisse müssen vor unbefugtem Zugriff gesch ützt werden.
- Dateien und Verzeichnisse besitzen
  - Eigentümer,
  - Leserechte,
  - Schreibrechte und
  - Ausführungsrechte.

Arten von Benutzern

`user` wird mit `u` abgekürzt und ist der Eigent ümer.

`group` wird mit `g` abgekürzt und betrifft alle Benutzer, die einer bestimmten Gruppe zugeordnet sind.

`other` wird mit `o` abgekürzt und entspricht dem Rest aller verbliebenen Benutzer im System.

`all` wird mit `a` abgekürzt und fasst die obigen Gruppen zusammen.

Ausführungsrechte dieser Gruppen

- `r` für `read` also Leserecht.
- `w` für `write` also Schreibrecht.
- `x` für `execute` also Ausführungsrecht.

Außerdem gelten noch die Abkürzungen:

- `d` für `directory` also Verzeichnis.
- `l` für `link` also Verweis bzw. Verknüpfung.

`t/T` für `Sticky Bit`

`s/S` für `Set Group ID` (SGID) oder `Set User ID` (SUID)

Anzeigen der Zugriffsrechte

Eingabe von

```
ls -l
```

erzeugt die Ausgabe:

```
drwxr-x--x 2 mheiste ZZZZ 1024 Sep 14 11:17 mydirectory
-rwxr----- 1 mheiste ZZZZ 1206 Sep 14 10:54 textfile.txt
```

Die ersten 10 Zeichen geben Auskunft über die Zugriffsrechte. Die erste Stelle steht für

- `-` Datei,
- `d` Verzeichnis oder
- `l` Link.

Die jeweils nächsten drei Stellen repräsentieren die Benutzergruppen `user`, `group` und `other`.

Sticky Bit (t-Bit)

- spezielles Ausführungsrecht
- Programme mit Sticky Bit bleiben auch nach Beendigung im Speicher
  ➔ beschleunigter Start
- für Verzeichnisse erlaubt das t-Bit Schreib- und Löschoperationen innerhalb des Verzeichnisses nur noch für Eigentümer (wichtig z.B. im `/tmp`-Verzeichnis)

SGID und SUID (s-Bit)

- spezielles Ausführungsrecht
- Ersetzt das `x` für Eigentümer (SUID) oder Gruppe (SGID)
- die entsprechende Datei wird mit den Rechten des Eigentümers oder der Gruppe ausgeführt

Achtung: Root-Dateien mit s-Bit sind gefährlich!

Für Verzeichnisse:

1. Verzeichnis anlegen
2. Verzeichnis einer Gruppe zuordnen
3. s-Bit (SGID) für das Verzeichnis setzen
4. Alle Dateien/Verzeichnisse, die in diesem Verzeichnis angelegt werden, gehören der Gruppe des Verzeichnisses.

ändern der Zugriffsrechte

Die Zugriffsrechte kann ausser `root` nur der Eigentümer ändern.

Der Befehl

```
chmod
```

ändert die Zugriffsrechte. Der Befehl

```
chown
```

ändert Eigentümer und Gruppe.

chmod-Syntax

```
chmod <usergroup><+-><rwx> <Datei/Verzeichnis>
```

Beispielsweise würde die Eingabe von

```
chmod go+x textfile.txt
```

die Ausgabe des vorangegangenen Beispiels ändern in:

```
-rwxr-x--x 1 mheiste ZZZZ 1206 Sep 14 10:54 textfile.txt
```

Alle weiteren Benutzer außer des Eigentümers dürfen nun also auch die Datei `textfile.txt` ausführen.

Oktale Darstellung bei `chmod`

Es gibt noch eine weitere Syntax für `chmod`:

```
chmod ??? <Datei/Verzeichnis>
```

Dabei werden die Fragezeichen jeweils durch eine Ziffer von 0-8 ersetzt.

	user			group			other		
	r	w	x	r	w	x	r	w	x
konventionelle Darstellung:	r	w	x	r	-	x	-	-	x
Binärdarstellung:	1	1	1	1	0	1	0	0	1
Oktaldarstellung:	7			5			1		

Globale Einstellung mittels `umask`

Die Syntax für `umask` lautet:

```
umask [xxx]
```

wobei `xxx` wieder die Oktaldarstellung ist. Duch `umask` werden die Rechte eingeschränkt, also abgezogen. Die gewählten Zugriffsrechte geltenab dem Aufruf von `umask` für alle neu erstellten Dateien bis zum Ende der Sitzung beendet.

Der Aufruf

```
umask 077
```

entfernt alle Rechte der Benutzegruppen `group` und `other`, so dass alle Rechte (`rwx`) für den Eigent ümer (`user`) gesetzt werden.

Vierstellige Rechtedarstellung (oktal)

Es kann den drei genannten eine vierte Zahl vorangestellt werden:

1. `t`-Bit
2. `s`-Bit für die Gruppe
4. `s`-Bit für den Eigentümer

chown-Syntax

```
chown <user:group> <Datei/Verzeichnis>
```

dabei sind die Synonyme für `user` und `group` abh ängig von den Vorgaben innerhalb des jeweiligen Systems. Die Administratoren können für diese Gruppen beliebige Namen vergeben.

Dieser Befehl ist in der Regel ausschließlich `root` vorbehalten

Erstellen einer »leeren« Datei

Mittels des Aufrufs

```
touch <Dateiname>
```

können Dateien erstellt werden, die keinen Inhalt besitzen.

Wird eine Datei angegeben, die bereits existiert, so wird ihr Zugriffsdatum auf den aktuellen Wert gesetzt.

Der Editor emacs

Er wird aufgerufen, indem man eingibt:

```
emacs [Quelle]
```

Wobei die Quelle, falls sie angegeben wird, eine Textdatei sein sollte. Der Emacs ist mit der Maus bedienbar und erklärt sich in seinen einfachsten Funktionen praktisch von selbst.

Weitere verbreitete Editoren sind joe, ed und vi, auf die in diesem Rahmen lediglich auf den vi eingegangen wird.

Der Editor `vi`

Er wird aufgerufen, indem man eingibt:

```
vi [Quelle]
```

Nach Eingabe dieser Zeile wird der Inhalt der `Quelle` am Bildschirm angezeigt. Möchte man diesen Inhalt verändern, muss man zunächst in den sogenannten Kommandomodus durch Eingabe des Doppelpunktes gelangen. Dann kann man durch Eingabe von Befehlen den Text der `Quelle` kommandoorientiert verändern.

Genaugenommen kennt der `vi` noch weitere Modi, aber sie sind alle durch den Kommandomodus erreichbar, weshalb sie nicht explizit benannt werden.

vi-Kommandos

Wichtige vi-Kommandos (durch Eingabe von ESC gelangt man wieder in den ursprünglichen Modus zurück):

Einfügemodi:

- i Einfügemodus
- a Wechsel in den Einfügemodus rechts vom Cursor
- r Zeichen unter dem Cursor ersetzen
- R überschreibmodus
- p Nach dem Cursor Inhalt der Zwischenablage einfügen
- P Vor dem Cursor Inhalt der Zwischenablage einfügen

Löschen:

- dd Zeile löschen
- D Zeile ab dem Cursor löschen

...

vi-Kommandos

Markieren:

- `v` Markiermodus Cursorweise starten
- `V` Markiermodus zeilenweise starten
- `y` Markierter Bereich in die Zwischenablage
- `dd` Marierten Bereich löschen

Bewegen (Cursortasten oder):

- `h` links
- `l` rechts
- `k` hoch
- `j` runter
- `w/W` nächstes Wort rechts
- `^` Zeilenanfang
- `$` Zeilenende

...

vi-Kommandos im Kommandomodus

Wechsel in den Kommandomodus mit ':',
verlassen desselben mit 'ESC':

```
[Bereich]s/Muster1/Muster2/
```

Ersetzen von Muster1 durch Muster2 innerhalb von [Bereich], wobei die Bereichsangabe dem Muster

```
Anfangszeile,Endzeile
```

entspricht. ^ und $ stehen für erste und letzte Zeile.

vi-Kommandos im Kommandomodus

`set nu`	Zeilennummerieirung einschalten
`set nonu`	Zeilennummerierung ausschalten
`syntax on`	Syntax-Highlightning einschalten
`syntax off`	Syntax-Highlightning ausschalten

Verbindung zu anderen Rechnern per `ssh`

`ssh` ist ein Terminalprogramm, um eine verschlüsselte Verbindung zu einem entfernten Rechner im Netz aufzubauen.

Gründe für den Einsatz:

- Zur Administration des entfernten Rechners.
- Um Programme auf dem entfernten Rechner zu starten.
- Um Informationen auf dem entfernten Rechner abzurufen.
- Verbindungen tunneln.
- Portumlenkung.

ssh-Syntax

```
ssh [Optionen] <Hostname>
```

Wichtige Optionen:

- `-l` Loginname
- `-X` X11-Forwarding
- `-p` Port
- `-L` Tunnel
- `-D` Portumlenkung

SSH-Tunnel

Tunnel zu Host1 über Host2:

```
ssh -L <LocalPort>:<Host1>:<RemotePort> <User>@<Host2>
```

Jede Verbíndung über `LocalPort` an `localhost` geht tatsächlich verschlüsselt über `Host2` nach `Host1`.

Portumlenkung mit SSH

```
ssh -D [BindAddress]:<Port> <User>@<Host>
```

Jede Verbindung zur `BindAddress` über Port `Port` geht tatsächlich über `Host`.

telnet

Unverschlüsseltes SSH. Vor allem zum Dienstecheck gut geeignet: Antwortet ein bestimmter Dienst auf einem bestimmten Port?

```
telnet <Hostname> [PortNr]
```

z.B.:

```
telnet mail.rrzn.uni-hannover.de 25
```

erzeugt (hoffentlich) die Ausgabe:

```
Trying 130.75.6.240...
Connected to mail.rrzn.uni-hannover.de.
Escape character is '^]'.
220 mail6240.rrzn.uni-hannover.de ESMTP Sendmail 8.14.1/8.14.1; \
Tue, 13 May 2008 07:15:34 +0200
```

Dateitransfer per `ftp`

- Programm, um Daten zwischen Rechnern hin- und herzuschieben
- kommandozeilenorientiert
- ähnlich zu Telnet
- Download von Daten ist schneller als im WWW.
- Man muss vorher wissen, wo bestimmte Daten zu finden sind.
- Sogenannte *Anonymous FTP-Server* (aFTP) sind jedem zug änlich.

ftp-Syntax

```
ftp [Optionen] hostname
```

Datenformate bei `ftp`

- ASCII-Dateien (Text, Sonderzeichen, Zeilenumbrüche...)
- Binärdateien (Archive, Programme...)
- Die übertragungsart ist für beide Dateitypen unterschiedlich
- Evtl. müssen ASCII-Dateien nach dem Download noch konvertiert werden, da die verbundenen Rechner über unterschiedliche Betriebssysteme und daher über unterschiedliche Kodierungen für Sonderzeichen (Umlaute etc.) und Zeilenumbrüche verfügen.

ftp-Befehle

`ascii`	übertragungsart für ASCII einstellen.
`bin`	übertragungsart für Binär einstellen.
`ls`	Inhalt des aktuellen Verzeichnisses anzeigen.
`cd ...`	Verzeichniswechsel
`get ...`	herunterladen
`mget <Muster>`	Alle Dateien, deren Namen dem `Muster` entsprechen herunterladen.
`put ...`	hochladen
`mput <Muster>`	Mehrere Dateien hochladen.
`prompt`	Rückfragen beim Down- oder Upload unterdrücken.
`lcd ...`	lokales Verzeichnis wechseln.
`delete ...`	Datei löschen.
`!`	`ftp` kurzfristig verlassen (Rückkehr mit `exit`).
`exit, bye`	`ftp` verlassen.

Shell-Skripten

- Ein Shell-Skript ist die Zusammenfassung von Kommandos in einer Textdatei.
- Diese Textdatei wird wie ein ablaufbares Programm behandelt.
- Es werden von einer Shell auch programmierähnliche Konstrukte, wie Schleifen, Bedingungen, Parameter und Variablen zur Verfügung gestellt.

Erstellung eines Shell-Skriptes

Ein Shell-Skript wird mit einem normalen ASCII-Editor (z.B. `emacs`) erstellt. Das Skript sollte in der ersten Zeile den Hinweis auf die verwendete Shell in folgender Form enthalten:

```
#!/bin/<Shell-Name>
```

Dabei teilen die beiden ersten Zeichen dem Betriebssystem mit, dass es sich um ein Shell-Skript handelt, und der folgende Pfad inklusive Shell teilen dem System mit, für welche Shell das Skript erstellt wurde.

Diese Zeile heißt auch Shebang.

Wo ist mein Programm?

Kennt man den Pfad zur Shell nicht, so kann man zwei Kommandos ausprobieren, die einem den Ort der Shell im Dateibaum anzeigen:

```
locate <Shell-Name>
```

```
which <Shell-Name>
```

Alle Befehle lassen sich im übrigen auch auf alle anderen ausführbaren Programme oder Datei-/Verzeichnisnamen anwenden (dennoch müssen beide Befehle nicht zwingend auf einem System vorhanden sein).

Da Linux unter den Zugriffsrechten auch ein Ausführungsrecht besitzt, das normalerweise beim Anlegen von Textdateien nicht automatisch gesetzt wird, muss dieses Recht erst noch aktiviert werden:

```
chmod u+x <Shell-Skript>
```

Einfaches Beispiel

```
#!/bin/sh

  echo Ich bin ein einfaches Shell-Skript

exit 0
```

Die eingefügten Leerzeilen sind nicht verpflichtend. Ebenso kann die letzte Anweisung weggelassen werden. Bei komplexen Skripten kann der `exit`-Status aber nützlich sein, um Informationen darüber zu erhalten, ob das Skript korrekt abgearbeitet wurde.

Kommandos werden entweder durch Zeilenwechsel oder durch Semikolon getrennt.

Parameterübergabe beim Aufruf

Man kann einem Skript beim Aufruf (beliebig) viele Parameter übergeben, die innerhalb des Skriptes verarbeitet werden können.

```
skript [p1] [p2] ... [pn]
```

Diese Parameter werden als Text interpretiert, der innerhals des Skriptes in den Variablen

```
$1, $2, ... $9
```

gespeichert wird. Auch hier ist das Dollarzeichen `$` wieder die Dereferenzierung auf den Inhalt einer Variablen.

Das Shell-Skript selbst kann nur die Parameter 1–9 ansprechen.

Das Kommando `shift`

Mittels

```
shift n
```

kann der Zugriffsbereich innerhalb des Skriptes um den Betrag `n` vershoben werden, sodass auch Parameter jenseits von 9 angesprochen werden können. Der Parameter `$1` entspricht dann dem `n+1`-ten ursprünglichen Parameter.

```
#!bin/sh
  echo $2 $1
  shift 9
  echo $1
exit 0
```

Dabei spielt es keine Rolle, wenn mehr oder weniger Parameter übergeben werden als innerhalb des Skriptes verarbeitet werden.

Umdefinieren von Parametern

Mit dem Kommando

```
set Parameter1 Parameter2 ...
```

können die übergebenen Parameter neu definiert werden. Dabei entspricht die Reihenfolge nach dem `set`-Kommando der Reihenfolge bei der übergabe.

Beispiel:

```
#!/bin/sh
  echo $1 $2
  set Hallo Welt!
  echo $1 $2
exit 0
```

Besondere Parameter

- `$0` Name des Skriptes
- `$#` Anzahl der übergebenen Parameter
- `$*` Die übergebenen Parameter werden als eine Zeichenkette interpretiert.
- `$$` PID-Nummer des aktuellen Prozesses.
- `$?` Rückgabewert des letzten Prozesses.

Variablen

Durch den Zuweisungsoperator = können auch Variablen definiert und mit Werten besetzt werden:

```
#!/bin/sh
  variable="Hallo Welt!"
  echo $variable
exit 0
```

Wichtig sind die Anführungszeichen, da sonst durch das Leerzeichen »Welt!« als nicht zu »Hallo« gehörend betrachtet würde. Eine Fehlermeldung wäre die Folge.

Variablen mit der Ausgabe eines Kommandos besetzen

Soll beispielsweise eine Variable `Datum` mit dem aktuellen Datum des Systems besetzt werden, so kann `Datum` die Ausgabe des Kommandos `date` zugewiesen werden. Dazu muss das auszuführende Kommando innerhalb der Shell in sogenannte Backticks oder in einen speziellen Klammerterm gesetzt werden:

```
Datum=‘date‘
```

oder

```
Datum=$(date)
```

For-Schleife (1)

Um den Teil eines Skriptes mehr als einmal, abhängig von einer gewissen Bedingung, durchlaufen zu können, gibt es die `for`- Anweisung:

```
for <variable>
do
  Kommandos ...
done
```

Hierbei wird die Schleife sooft durchlaufen, wie Argumente beim Aufruf des Skriptes übergeben wurden. Mit jedem Durchlauf wird die `variable` auf den Wert des nächstes Argumentes gesetzt.

Beispiel

```
#!/bin/sh

for parameter
do
  echo $parameter
done

exit 0
```

For-Schleife (2)

Man kann auch ein konkretes Ziel angeben, mit dem die `variable` in der Schleife verglichen bzw. gesetzt werden soll:

```
for <variable> in <argumente>
do
  Kommandos ...
done
```

Beispiel

```
#!/bin/sh

for parameter in 9 8 7 6 5 4 3 2 1
do
  echo $parameter
done

exit 0
```