Kapitel 1

Allgemeines

1.1 Entstehung

Die Geschichte von Linux reicht zurück ins Jahr 1991 zurück, als der finnische Student Linus Torvalds nach einer Alternative zu MS DOS suchte, da es unter DOS unmöglich war, mehrere Programme gleichzeitig laufen zu lassen. Bei seiner Suche gelangte er zu Minix, von Andy Tannenbaum entwickelt. Minix ist ein Unix Clone in einer recht rudimentären Fassung. Linus Torvalds beschäftigte sich mit diesem Betriebssystem intensiv, erweiterte es um Funktionen und ersetzte immer mehr Code durch eigenen. Nachdem er die ersten Versuche noch in Assembler programierte, entschied er sich bald für die geeignetere Programiersprache C.

Linux erblickte das Licht der Welt, und da Torvalds den Sourcecode an interessierte Minix User verschickte, fanden sich bald weltweit Interessenten zusammen, die fortan über das Internet kommunizierten und ihre neuesten Erweiterungen zu Linux verbreiteten.
Von Anfang an stellte Torvalds seine Sourcen unter die Verantwortung der GPL[1], so dass diese frei kopiert werden konnten und jedem Interessenten zur Verfügung standen. Gleichzeitig war man bei der Entwicklung auf Konformität zum POSIX-Standard[2] bedacht, wodurch Linux ohne großen Aufwand auf andere Hardware-Plattformen portierbar wurde.

1.2 Was ist Linux eigentlich?

- "Linux" ist genaugenommen ein Betriebssystemkern (engl. "kernel"), der durch Anwenderprogramme zu einem kompletten Betriebssystem mit dem vollen Funktionsumfang moderner Bertiebssysteme wird. Der Name geht auf "Linus und Unix" zurück und zeigt damit an, dass Linux ein Unix-artiger Betriebssystemkern ist.
- Dieser Betriebssystemkern ist nach dem Start des Computers die oberste Instanz und hat im wesentlichen folgende Fähigkeiten und Eigenschaften:

    - "multitasking": mehrere Aufgaben (engl. "tasks") können gleichzeitig erledigt werden.
    - "multiuser": jede Personen, die den Computer benutzt, hat eine eigene "Privatsphäre" (geschützt durch ein Passwort).Gleichzeitig können theoretisch unbegrenzt viele Nutzer mit nur einem System arbeiten.
    - "multiprocessing": d.h. für bis zu 16 Prozessoren (Intel und SPARC)
    - die Ressourcen des Computers (wie Arbeitsspeicher [RAM], Rechenleistung und Netzwerk) werden den konkurrierenden Programmen zugeteilt (d.h. keine Selbstbedienung), dadurch wird eine gewisse Sicherheit erreicht.
    - verschiedene Modelle von Computerbestandteilen mit gleicher Funktion (z.B. verschiedene Soundkarten) können von den Programmen in stets gleicher Art und Weise angesprochen werden; eine große Hilfe für Programmierer.
    - Der Betriebssystemkern von Linux ist in der Programmiersprache C geschrieben wodurch eine hohe Portabilität erreicht wird.

[1] GNU General Public License
[2] bezeichnet einen Programmierstandard in der Linux-Welt

- Netzwerkfähig, alle gängigen Netzwerkprotokolle, insbesondere TCP/IP
- Unterstützung »fremder« Dateisysteme: DOS, Windows 9x, NT und OS/2

1.3 Die wichtigsten Merkmale von Linux

- Mehrbenutzer-Betriebssystem (Multiuser)
- Multitasking
- Multiprocessing
- Netzwerkfähig
- Betriebssystemkern im Quellcode
- Geräteunabhängigkeit
- POSIX-konform
- Demand Paging, bedeutet seitenweise Prozessauslagerung. Es wird erforderlich, wenn der Hauptspeicher nicht alle zum Prozess notwendigen Teile speichern kann. So müssen im Moment nicht genutzte Seiten oder Pages anderer Prozesse aus dem Hauptspeicherentfernt werden, um Platz zu schaffen. Werden nun jene Seiten wieder benötigt, müssen sie wieder in den Hauptspeicher gebracht werden; sie dürfen also nicht gelöscht worden sein. Dieses Aus- und spätere Wiedereinlagern wird als demand paging bezeichnet.
- Virtuelles Speichermanagement (Paging), heißt, es existiert ein spezieller Speicherbereich, der neben dem Hauptspeicher bereitgehalten wird.
- Timesharing, d.h. die Prozessorzeit wird zwischen mehreren Prozessen aufgeteilt.
- Riesige Palette an kostenloser Anwendersoftware.
- Linux wird von einer Fangemeinde aus Programmierern entwickelt, die Linux selbst einsetzen; daher werden Fehler schnell erkannt und behoben; die Entwicklung ist sehr rasant.
- Linux ist kostenlos, zuverlässig, schnell.
- Nahezu jede Server-Funktion läßt sich mit Linux realisieren. Sei es als File-Server für DOS/Win-PCs, FTP-Server, Firewall

1.4 Regeln für den Umgang mit Linux

- Alle Befehle werden ohne Nachfrage ausgeführt, also Vorsicht!!!
- Regelmässig Sicherheitskopien anlegen.
- Befehle zur Systemverwaltung sind dem Benutzer "root" vorbehalten.
- Es wird i.d.R. zwischen Gross- und Kleinschreibung unterschieden.

Kapitel 2

Erste Schritte mit Linux

2.1 Login Prompt

Linux (und UNIX im Allgemeinen) ist ein Mehrbenutzersystem, d.h. gleichzeitig können theoretisch unbegrenzt viele Nutzer mit nur einem System arbeiten. Um Dateien eindeutig einem Nutzer zuordnen zu können, benötigt Linux Informationen über diesen, d.h. ein Nutzer muss sich beim System anmelden:

```
login:
```

Ein Nutzer könnte sich leicht als ein anderer ausgeben, z.B. als root, einer Nutzerkennung, die auf jedem Unix-System existiert. Um vor Missbrauch zu schützen, muss dieser sich dem System gegenüber auch ausweisen, ein geheimes Passwort sorgt für den Schutz:

```
Password:
```

Stimmen Nutzerkennung und Passwort überein, weist sich der Benutzer dem System gegenüber als berechtigter User aus, dem dann definierte Rechte zugestanden werden. Linux startet eine Shell, einen Kommandozeileninterpreter, der die Interaktion zwischen Benutzer und System steuert.

2.2 Logout

Möchten Sie die aktuelle Sitzung beenden, so verabschieden Sie sich vom System:

```
user@linux> logout
```

Das Login-Prompt fordert zum erneuten Anmelden auf. Ein analoges Verhalten bewirken Sie mittels der Tastenkombination **[Strg]-[D]** (falls konfiguriert) bzw. mit der Eingabe von

```
user@linux> exit.
```

Unter DOS war es noch üblich, den Ausschalter zur Beendigung einer Sitzung am Rechner zu nutzen. Auch unter Linux dient der Ausschalter dem endgültigen Knock-out des Rechners. Zuvor jedoch ist ein ordnungsgemäßes Herunterfahren des Systems dringend zu empfehlen, um einen möglichen Datenverlust zu vermeiden:

```
root@linux> reboot
root@linux> shutdown –r now
```

startet den Rechner neu und

```
root@linux> reboot
root@linux> shutdown –h now
```

halten das System dagegen an.

2.3 Kommandozeile - die Shell

Nach erfolgreichem Login ist man auf der Kommandozeile angelangt, der sog. Shell. Die Kommandozeile beginnt etwa so:

```
          [user@linux tmp]$
          `----`------`---`
             |      |     |
  "login-name"     |      Verzeichnis, auf das sich die
                   |      Kommandos beziehen
           Name des Linux-PCs
```

Den Bereich vom Anfang der Zeile bis zum Cursor bezeichnet man als "Prompt". Man kann es selbst definieren, z.B.:

```
PS1='\u@\h:\w> '
```

(´\u´= "username", ´\h´= "hostname", ´\w´= "working directory")

- Die Shell verkehrt mit dem Kern über einen definierten Satz von Systemaufrufen.
- Die Shell stellt dem Benutzer die Dienst- und Anwendungsprogramme zur Verfügung und ist der Kommando-Interpreter des UNIX-Systems.
- Sie dient zum Aufruf der Programme, mit denen man eigentlich arbeiten will. Dabei kann die Shell sowohl Systemkommandos (= Kommandos, die vom System als Dienstleistung zur Verfügung gestellt werden) als auch Benutzerkommandos (= vom Benutzer selbst erzeugte Kommandos) ausführen. Ein typischer Programmaufruf sieht etwa so aus:

```
[user@linux procMail]$  less  -MMI  File
`-------- Prompt ----`  `---` `---` `----`
                          |     |      |
                Programmname    |      |
                             Optionen  |
                                     Datei
```

Optionen:

- Die Informationen über das was, wen und wie gibt man nach dem Programmnamen an.
- "Optionen" beginnen meist mit einem ´-´ und beeinflussen die Grundfunktionen des Programms.

Beispiel: `less –MMI`

Zwei Formate zur Angabe von Optionen sind verbreitet:

- herkömmliche Kurzform: '**-**Option'

Beispiel: `[user@linux procMail]$` **`less -?`**
für das Anzeigen der Hilfe

- GNU[3]-Format: '**--**Option'

Beispiel: `[user@linux procMail]$` **`less --help`**
für das Anzeigen der Hilfe

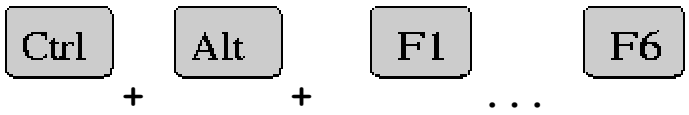
- Die Shell ist aus Sicht des Betriebssystems ein normales Programm. Aus diesem Grund existieren unter Linux eine Vielzahl von Kommandozeileninterpretern mit unterschiedlichsten Funktionalitäten. Zur Standardshell unter Linux hat sich die **`bash`** (Bourne Again Shell) gemausert, die viele Eigenschaften der (teils) historischen Shells `csh` (C-Shell), `bsh` (Bourne Shell) und `ksh` (Korn Shell) in sich vereint.

2.4 Virtuelle Konsolen

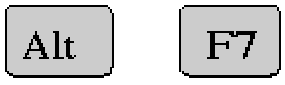
Linux ist ein Multitasking- und Multiuser-System und bietet damit mehrere Möglichkeiten, Kommandos gleichzeitig auszuführen. Mit den Tastenkombinationen

lässt sich zwischen den so genannten virtuellen Konsolen wechseln. Auf diesen Konsolen erscheint ein Login-Prompt und man kann auf jeder Konsole eine Sitzung eröffnen. So kann man auf der ersten Konsole ein Programm editieren und dieses auf der nächsten Konsole compilieren, ohne den Editor beenden zu müssen. Zu jeder Zeit kann natürlich nur eine physische Konsole (Bildschirm) aktiv sein.

Arbeitet man unter dem X-Window System (z.B. KDE, GNOME etc.) erreicht man eine der Textkonsolen über die Tastenkombinationen



Der X-Server selbst ist auf der 7. Konsole aktiv, zu der man über



gelangt.

[3] Das "rekursive Akronym" GNU steht für **G**NU Is **N**ot **U**nix , ein Projekt das in erster Linie für die Verfügbarkeit freier Software eintritt

2.5 Hilfe

2.5.1 Manual Pages

Unixe und so auch Linux verfügen von Haus aus über ein ausgereiftes Hilfesystem, die **Manual Pages**. Zu nahezu jedem Kommando findet man dort eine ausführliche Beschreibung und jeder Programmierer ist dazu aufgefordert, zu seinen Programmen ebenfalls Manuals zu erstellen.

```
user@linux> man ls
LS(1)                          FSF                           LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by
       default). Sort entries alphabetically if none of -cftuSUX
       nor --sort.

       -a, --all
              do not hide entries starting with .

       -A, --almost-all
              do not list implied . and ..

       -b, --escape
              print octal escapes for nongraphic characters

       --block-size=SIZE
              use SIZE-byte blocks
...
```

zeigt eine kompakte Beschreibung des Kommandos **ls** mit all seinen Optionen, Argumenten, Informationen zu bekannten Problemen, Verweisen zu verwandten Kommandos.
Selbstverständlich gibt es zur Bedienung von `man` selbst ein Manual.
Die Manuals selbst folgen einem festgelegten Format:

NAME
: Kommandoname (bzw. Dateiname...) und Kurzbeschreibung

SYNOPSIS
: Aufrufsyntax

DESCRIPTION
: Detaillierte Beschreibung der Wirkungsweise aller möglichen Optionen

FILES
: Vom Kommando benötigte/modifizierte Dateien

SEE ALSO

Hinweise auf verwandte Kommandos

DIAGNOSTICS

Erläuterungen zu Fehlercodes, die das Kommando zurückliefert

BUGS

Bekannte Fehlverhalten des Kommandos, aber auch Hinweise auf Wirkungen, die gewollt aber ungewöhnlich sind

EXAMPLE

Beispiele zur Verwendung des Kommandos (fehlt leider oft)

2.5.2 apropos und whatis - Welches Manual weiß was?

...doch oft ist man sich seiner Sache nicht so sicher - dann hilft einem das Kommando **apropos** auf die Sprünge. So gibt es zum Bsp. das Kommando write als Systemaufruf (schreibt in eine Datei) als auch als Dienstprogramm für Benutzer (schreibt Nachricht auf das Terminal eines anderen Benutzers).

```
user@linux> apropos write
wall (1)             - write a message to users
write (1)            - send a message to another user
write (2)            - write to a file descriptor
writelog (8)         - add a entry to an INN log file.
...
```

Das Kommando durchsucht die DESCRIPTION-Zeilen aller Manuals und schreibt die übereinstimmenden Zeilen auf die Standardausgabe. Analog zu **apropos** arbeitet **man -k**.

Wenn ich weiß, dass die Betriebssystemfunktionen des Befehls **write** in der Sektion 2 enthalten sind, und ich eben diese Beschreibung von **write** benötige, so gebe ich die gewünschte Sektion gleich an:

```
user@linux> man -S 2 write

oder kurz

user@linux> man 2 write
```

Die meisten heutigen Linux-Distributionen beinhalten auch noch das Kommando **whatis**, das ähnliche Informationen wie **apropos** liefert, aber mittels einer kleinen Datenbank (oft »/usr/man/whatis« - eine sortierte Textdatei) arbeitet. **whatis** ist somit sehr schnell, bedarf aber einer ständigen Aktualisierung der Datenbasis.

```
user@linux> whatis write
write (1)            - send a message to another user
write (2)            - write to a file descriptor
```

2.5.3 help - Kurzbeschreibung zu builtin-Kommandos

Ein Kommandozeileninterpreter besitzt, teils aus Notwendigkeit, teils wegen besserer Performance, einige integrierte Kommandos, zu denen man manchmal vergeblich nach Manuals suchen würde. Speziell für solche builtin-Kommandos existiert das (builtin-Kommando) **`help`**:

```
user@linux> help help
help: help [pattern ...]
    Display helpful information about builtin commands. If PATTERN is
    specified, gives detailed help on all commands matching PATTERN,
    otherwise a list of the builtins is printed.
```

2.5.4 info - die neuen Manuals?

Andere Informationsquellen sind die *Info*-Seiten, die man durch Eingabe von

```
user@linux> info
File: dir       Node: Top       This is the top of the INFO tree

  This (the Directory node) gives a menu of major topics.

  Usage in info-mode of EMACS:
    Typing "d" returns here,       typing "?" lists all INFO commands,
    typing "q" exits,     typing "h" gives a primer for first-timers,
    pressing 2nd button on a highlighted word follows cross-reference.

---- AUTOCONFIGURED BY SuSEConfig: EDIT WITH CARE: ----
---- Only descriptive text for otherwise empty topics will survive ---
-

* Menu: The list of major topics begins on the next line.


* a2ps: (a2ps).
               PostScript Generating Utility
* PreScript: (a2ps) PreScript.
               Input language for a2ps
...
```

erreicht. Die folgende Aufzählung fasst die wichtigsten Tastencodes zusammen, um innerhalb der Info-Seiten navigieren zu können. Zuerst sollte man sich der interaktiven Hilfe zu **`info`** widmen, um die Bedienung zu verstehen.

h

Startet die Hilfe zu `info`

?

Listet alle Kommandos von `info` auf (Ende mit »l«)

[Leertaste]

Im Text eine Seite nach unten scrollen

[Backspace]

Im Text eine Seite nach oben scrollen

n

Springt zum nächsten Thema

p

Kehrt zum letzten Thema zurück

m

Thema auswählen:

1. Cursor auf einen Menüeintrag des aktuellen Themas setzen und »m« eingeben
2. »m« eingeben gefolgt vom gewünschten Thema ([Ctrl]-[G] verwirft die Eingabe)

q

info beenden

Die Info-Seiten selbst sind als Nachfolger der Manual Pages vorgesehen, in einigen Manuals findet man daher den Verweis auf ihre Info-Versionen. Trotz seiner erweiterten Möglichkeiten - wie einer Hypertext-artigen Verlinkung - hat sich **`info`** bislang nicht durchsetzen können.

2.5.5 HowTo's - verschiedene Kochrezepte

In den vom **Linux Documentation Project**[4] gepflegten HOWTO's findet man Informationen zur Installation, Konfiguration, zu von Linux unterstützter Hardware usw.
Die komprimierten ASCII-Quellen finden sich nach der Installation meist im Verzeichnis »/usr/doc/howto/« oder »/usr/share/doc/howto« und lassen sich z.B. mit dem Pager **`less`** betrachten.
Darüber hinaus liegen zahlreiche HowTo's auch als HTML-, Postscript- oder PDF-Version vor. Ebenso existieren Übersetzungen der zumeist aus dem englischen Sprachraum stammenden Originale.

Homepage des Deutschen Linux HOWTO Projektes:

http://www.linuxhaven.de/dlhp/

2.6 Der Editor VI - Kurzreferenz

Der Editor vi ('vi' steht für 'visual') ist ein bildschirmorientierter Editor, d. h. der Text ist in seiner aktuellen Version auf dem Bildschirm zu sehen. Da der vi auf jedem UNIX und Linux System vorhanden ist, ist es unabdingbar, grundlegende Kenntnisse dieses Editors zu erlangen.

Der vi kennt drei Betriebsarten:

1. Der ***visual mode***, ist die Standardbetriebsart des vi, d.h., man befindet sich direkt nach dem Start des vi darin. Aus allen anderen Betriebsarten kommt man jederzeit

[4] http://www.tldp.org/

durch Drücken der *Escape* Taste zurück. Die Idee dahinter ist, daß man, solange kein Text eingegeben wird, ohne Hilfe von Maus oder erweiterter Tastatur (Pfeiltasten usw.) in der editierten Datei durch Bewegen des Cursors, Springen und mit Hilfe von Bookmarks navigieren kann. Das ermöglicht Arbeit mit einem sehr hohen Tempo und auch auf Terminals ohne erweiterte Tastatur.

2. Der ***ex mode***. Der Ex Mode dient dazu, auch komplexere Kommandos oder Makros eingeben zu können, die durch jeweils einfache Tastendrücke im Visual Mode so nicht möglich wären. Man erreicht den Ex Mode aus dem Visual Mode heraus und zwar durch Drücken von ":". Ein Kommando im Ex Mode wird abgebrochen durch *Escape* oder beendet durch *Enter*.
3. Der ***input mode***, in dem Text eingegeben werden kann. Der Input Mode dient zum Eingeben von Text. Hier werden die normalen Tasten als einzugebende Buchstaben interpretiert. Andere Befehle aus dem Visual Mode, die auf Tasten liegen, die so nicht druckbar sind, wie z.B. *Ctrl-F* und *Ctrl-B* (*PageDown*und *PageUp*), stehen weiterhin zur Verfügung.Der Input Mode wird verlassen durch *Escape*, man landet somit wieder im Visual Mode.

Es werden hier nur die wichtigsten Kommandos für Einsteiger aufgeführt.

Aufruf:

```
vi [OPTIONEN][DATEINAME]
```

Ist die Datei vorhanden, wird sie in den Editorpuffer geladen, andernfalls wird sie neu angelegt.

vi starten

-i	startet vi gleich im Eingabemodus
-R	Datei read-only öffnen
+zeilennummer	springt direkt zur angegebenen Zeile
+/[muster]	Springt an die Stelle an der das erste mal [muster] auftaucht

Beispiel:

`vi +23 test.txt` öffnet die Datei test.txt und der Cursor steht in Zeile 23

vi beenden

:wq	Datei abspeichern und vi verlassen
:q	vi verlassen falls Datei abgespeichert wurde
:q!	vi ohne abspeichern verlassen
:w	abspeichern der Datei

Dateien laden

:e Datei	Datei laden, wenn die Datei nicht existiert wird eine erzeugt
:next	Die nächste Datei laden falls vi mit mehrern Dateien aufgerufen wurde
:prev	Die vorherige Datei laden falls vi mit meheren Dateien aufgerufen wurde

Text eingeben

i	(insert) Eingabe vor dem aktuellen Zeichen
a	(append) Eingabe nach dem aktuellen Zeichen
I	(Insert) Eingabe am Anfang der aktuellen Zeile
A	(Append) Eingabe am Ende der aktuellen Zeile
o	neue Zeile erzeugen und Eingabe nach der aktuellen Zeile
O	neue Zeile und Eingabe vor der aktuellen Zeile
Ctrl-v	Eingabe eines Steuerzeichens

Text ändern

[count] Zeichen	(replace), Änderung des aktuellen Buchstabens in Zeichen
R	(Replace), Überschreibemodus vom aktuellen Buchstaben aus
cw Wort	Ersetzt das aktuelle Wort vor dem Cursor durch Wort
cc Zeichenkette	ersetzt das aktuelle oder nächte Zeile durch Zeichenkette
J	hängt die der aktuellen folgenden Zeilen and die aktuelle an und positioniert den Cursor dazwischen

Suchen und Ersetzen

/ suchwort	Suche vorwärts nach dem Ausdruck suchwort
? suchwort	Suche rückwärts nach dem Ausdruck suchwort
n	Wiederholt das letzte Suchkommando.
N	Wiederholt das letzte Suchkommando in die jeweils andere Richtung.
f Zeichen	Sucht nach Zeichen in der aktuellen Zeile vorwärts
F Zeichen	Sucht nach Zeichen in der aktuellen Zeile rückwärts
:%s/Quelle/Ziel/	Ersetzt Quelle im Text einmal durch Ziel.
:%s/Quelle/Ziel/g	Ersetzt Quelle im Text überall durch Ziel.

Kapitel 3

Benutzerverwaltung

Nach der Installation von Linux existiert nur ein Benutzerkonto. Dieses Benutzerkonto ist *root*, das Konto für den Superuser. Daher muß der Administrator (Superuser) für die Arbeit weitere Konten anlegen.

3.1 Der Superuser root

Das Konto *root* wird nicht umsonst als Superuser bezeichnet. Der Benutzer unter diesem Konto darf im System alles. Er hat vollen Zugriff auf alle Verzeichnisse, Dateien und Geräte im System. Damit kann er auch alles löschen. Diese Aktion kann aber zu erheblichen Schäden am System führen, somit sollte man als Superuser sehr vorsichtig handeln.
Der Superuser *root* besitzt die UID 0 (*User IDentification*) und wird wie alle anderen Benutzer in der Datei `/etc/passwd` definiert. Da das System die Benutzer nicht nach ihrem Namen identifiziert, sondern nach ihrer **UID**, kann der Superuser auch einen anderen Login-Namen bekommen.
Für die alltäglichen Aufgaben sollten Sie sich nicht als *root* einloggen, da dies ein nicht zu unterschätzendes Risiko darstellt. Im normalen Betrieb sollten Sie als einfacher Benutzer arbeiten. Nur für administrative Aufgaben loggen Sie sich als *root* ein und erledigen die Aufgaben, um danach als einfacher Benutzer weiterzuarbeiten.

3.2 Das Benutzerkonto

3.2.1 Einrichten eines neuen Benutzers

Die Einrichtung eines **Benutzers erfolgt durch den Systemverwalter** mit den Kommandos **`useradd`** (Anlegen eines Benutzers).

```
useradd [OPTIONEN] [BENUTZER]
```

Beispiel:

`root@linux> useradd -m mustermann`	legt den Benutzer "mustermann" mit seinem Homeverzeichnis an und kopiert alle Dateien aus /etc/skel in dieses Verzeichnis.

3.2.2 Ändern des Passworts

Um ein Kennwort zu setzten oder zu ändern wird der Befehl **`passwd`** verwendet. Der Benutzer kann nur sein eigenes Kennwort ändern, während der Superuser alle Kennwörter ändern kann.

```
passwd [BENUTZER]
```

Beispiel:

```
root@linux> passwd mustermann
```

Wenn ein Benutzer sein Kennwort ändern will, dann braucht er natürlich keinen Benutzernamen anzugeben, sondern lediglich das Kommando **passwd** absetzten.

Beispiel:

```
mustermann@linux> passwd
```

Die Richtlinien für die Kennwörter werden in der Datei **/etc/login.defs** festgelegt. Das kann die Mindestlänge des Kennworts, das Auslaufdatum des Kennworts und die Warnung davor sein.

3.2.3 Löschen eines Benutzers

Der Befehl **userdel** dient dazu einen Benutzer zu entfernen.

```
userdel [OPTIONEN] BENUTZER
```

Optionen

-r Löscht auch das Heimatverzeichnis des Benutzers

Die Ausführung des Befehls ohne Optionen löscht den Benutzer aus der Datei **/etc/passwd** und aus anderen Systemdateien, enfernt aber nicht sein Heimatverzeichnis. Dies hingeben bewirkt der Schalter **-r**. Dateien des Benutzers, die sich außerhalb seines Heimtverzeichnis befinden, werden **nicht** gelöscht.

3.3 /etc/passwd

Die **Identifikation eines Benutzers** beim Einloggen in das System erfolgt über seinen **Benutzernamen** - **intern** wird der Benutzer jedoch über eine Benutzernummer (= **UID**) verwaltet - **und - falls verlangt** - über ein dem Benutzer zugeordnetes **Paßwort**. Außerdem wird jeder Benutzer nach dem Einloggen einer **bestimmten Benutzergruppe zugeordnet** - auch diese Gruppe wird **intern** über eine Gruppennummer (= **GID**) verwaltet. Diese Daten über die definierten Benutzer eines UNIX-Systems sind in der **Datei /etc/passwd** gespeichert. Zusätzlich zu diesen Daten sind in dieser Datei das **Home-Directory** des Benutzers und die Shell vorgegeben, die der Benutzer nach dem Einloggen erhält (= **Login-Shell**).

Beispielhaft könnten die Daten eines Benutzers in dieser Datei wie folgt aussehen:

```
asw:x:212:50:Erster User ASW:/usr/asw:/bin/sh
 |  |  |  |        |             |          |
 |  |  |  |        |             |          └── Shell
 |  |  |  |        |             └─────────────── Home-Directory
 |  |  |  |        └──────────────────────────── Beschreibung
 |  |  |  └──────────────────────────────────────── GID
 |  |  └─────────────────────────────────────────── UID
 |  └────────────────────────────────────────────── (*)
 └───────────────────────────────────────────────── Benutzername
```

Erläuterung zu (*):

Die Speicherung des Kennworts, wenn auch in einer verschlüsselten Form, in einer von allen lesbaren Datei **`/etc/passwd`** ist ein Sicherheitsrisiko. Der Einsatz von Entschlüsselungsprogrammen ermöglicht es relativ leicht an die Kennwörter zu gelangen. Deswegen ist der Einsatz des **Shadow-Paßwort-Systems** bei den heutigen Linux-Distributionen üblich. In der **`/etc/passwd`** steht für das Kennwort nur noch ein `**x**'. Die Kennwörter stehen in der **nur für *root*** zugänglichen **`/etc/shadow`**.

3.4 /etc/shadow

Die **`/etc/shadow`** enthält nicht nur den Namen und das Kennwort des Benutzers, sondern noch weitergehende Informationen zum Konto.

Beispielhafter Auszug der /etc/shadow:

```
mustermann:9Fd4SU/BbEP52:11317:0:10000::::
  |          |              |    |    |   ||└──── (8)
  |          |              |    |    |   |└───── (7)
  |          |              |    |    |   └────── (6)
  |          |              |    |    └────────── (5)
  |          |              |    └─────────────── (4)
  |          |              └──────────────────── (3)
  |          └─────────────────────────────────── (2)
  └────────────────────────────────────────────── (1)
```

Die Felder von links nach rechts sind:

(1) der Benutzername
(2) das verschlüsselte Kennwort.
(3) die Anzahl von Tagen zwischen dem 01.01.1970 und der letzten Kennwortänderung.
(4) die Zeit in Tagen, die zwischen zwei Kennwortänderungen liegen muß.
(5) die Zeit in Tagen, wie lange ein Kennwort gültig ist.
(6) die Zeit in Tagen, wie lange der Benutzer vor dem Auslaufen des Kennworts gewarnt wird.
(7) die Zeit in Tagen bis das Konto nach dem Auslaufen des Kennworts gesperrt wird.

(8) Auslaufen des Kontos in Tagen seit dem 01.01.1970.

Im Gegensatz zur **/etc/passwd** darf das Kennwortfeld nicht leer sein. Um einen Benutzer daran zu hindern sein Kennwort zu ändern, brauchen Sie nur das minimale Kennwortalter größer zu machen als das maximale Kennwortalter.

3.5 /etc/group - Gruppenzugehörigkeit

Eine **zweite Datei**, die der Identifikation von Benutzern dient, ist die Datei **/etc/group**. In dieser Datei sind die im UNIX-System definierten Gruppen und die zugehörigen Gruppenmitglieder gespeichert.

Beispielhaft könnten die Daten einer Gruppe in dieser Datei wie folgt aussehen:

```
asw:x:50:asw1,asw2,asw3,asw4
|   | |   |
|   | |   └──────────────────────── Benutzernamen der Gruppenmitglieder
|   | └──────────────────────────── GID
|   └────────────────────────────── verschlüsseltes Gruppenpaßwort
└────────────────────────────────── Gruppenname
```

Das Gruppenpasswort ist für den Wechsel in eine Gruppe gedacht. Normalerweise ist dieses Feld leer oder enthält nur einen Asterisk (*).

Gruppen und ihre Mitglieder können mit den Kommandos **groupadd** und **groupdel** verwaltet werden.

3.5.1 groupadd

Der Befehl **groupadd** legt eine neue Gruppe an.

groupadd [OPTIONEN] GRUPPENNAME

Optionen

-g GID	GID für die neue Gruppe vorgeben
-o	Mehrfache Nutzung der GID zulassen
-r	Systemgruppen einrichten (nicht S.u.S.E)

Die Gruppe wird mit einem leeren Gruppenkennwort eingerichtet und enthält noch keine Mitglieder. Die können mit dem Befehl **gpasswd** oder durch editieren der Datei **/etc/group** hinzugefügt werden.

3.5.2 groupdel

Das Löschen einer Gruppe erfolgt mit dem Befehl **groupdel**.

```
groupdel GRUPPE
```

Sollten Sie diesen Befehl auf eine Gruppe anwenden, die einem Benutzer als Login-Gruppe zugewiesen ist, wird der Befehl nicht ausgeführt.

3.6 Umgebung eines Benutzers

Durch das Einloggen des Benutzers wird für diesen Benutzer eine Shell gestartet. Die Shell, die beim Einloggen für den Benutzer gestartet wird, ist in der Datei /etc/passwd für den jeweiligen Benutzer festgelegt (siehe oben).

Das Aussehen und die Funktion der Shell ist durch die Umgebungsvariablen festgelegt.
Zu dieser Umgebung eines Benutzers gehören **mindestens die folgenden Informationen**, die in sogenannten **Umgebungs-Variablen** abgelegt sind:

- **Benutzter Terminaltyp** - abgelegt in **TERM**
- **Promptzeichen der Shell** - abgelegt in **PS1**
- **Home-Directory** - abgelegt in **HOME**

> **Hinweis:**
>
> Die Variable `$HOME` wird beim Login des Benutzers mit dem in der Datei `/etc/passwd` festgelegten Home-Directory belegt!

- **Suchpfad für Programme** - abgelegt in **PATH**

> **Hinweis:**
>
> Die im Suchpfad enthaltenen Verzeichnisses werden jeweils mit **:** voneinander getrennt.

Beispiel:

```
PATH=/bin:/usr/bin:/usr/local/bin
```

Die Standardeinträge für die Umgebungsvariablen für alle Benutzer, die sich interaktiv einloggen, befinden sich in der **/etc/profile**. Um systemweite Änderungen in den Umgebungen der Benutzer durchzuführen, sollte diese Datei editiert werden.

Wenn der Benutzer sein eigenes Umfeld gestalten will, kann er das in der Datei **.profile** machen, die in seinem Heimatverzeichnis liegt. Wenn ein Benutzer sich einloggt, dann liest das Betriebssystem erst die **/etc/profile** aus. Dann sucht es nach **.profile** im Heimatverzeichnis des Benutzers.

Hinweis:

Der **Inhalt *einer* Umgebungsvariablen** kann **angezeigt** werden mit dem Kommando

echo $*UMGEBUNGSVARIABLE*

Der **Inhalt *aller* Umgebungsvariablen** kann **angezeigt** werden mit dem Kommando

env

3.7 Informationskommandos über Benutzer

3.7.1 whoami

zeigt dem Benutzer den Namen, unter dem er eingeloggt ist

whoami

3.7.2 logname

Der Befehl `logname` erzählt dem Benutzer unter welchem Namen er sich ursprünglich mal eingeloggt hat.

logname

Frisch nach dem Einloggen zeigen die Befehle **whoami** und **logname** keine Unterschiede. Erst wenn ein Identitätswechsel vollzogen wurde, werden die Unterschiede der beiden Befehle deutlich:

```
max@linux:~ > whoami
max
max@linux:~ > logname
max
max@linux:~ > su moritz
Password:
moritz@linux:/home/max > whoami
moritz
moritz@linux:/home/max > logname
max
```

3.7.3 id

Das Kommando dient der Anzeige der **UID** (Benutzeridentifikationsnummer) und **GID** (Gruppenidentifikationsnummer) **des Benutzers**

id [OPTIONEN] [BENUTZERNAME]

3.7.4 who - Wer ist zur Zeit eingeloggt?

Diese Frage stellt sich dem Administrator immer dann, wenn er den Rechner herunterfahren will. Aber auch da kann Linux weiterhelfen. Der Befehl **who** ohne Optionen zeigt alle eingeloggten Benutzer mit ihren Terminals und Einlogzeit an.

```
who [OPTIONEN]
```

Optionen

-H	Zeigt eine Überschriftenzeile
-i, -u	Zeigt an, wie lange ein Benutzer nicht mehr gearbeitet hat
-m	Zeigt den aktuellen Benutzer an
-q	Zeigt die eingeloggten Benutzer und ihre Anzahl an
T	Zusätzliche Anzeige, ob das Terminal des Benutzers für Nachrichtenempfang (über write und wall) gesperrt ist

Für die Ausgabe des Befehl wertet er die Dateien /var/run/utmp und /var/log/wtmp aus.

Beispiel:

```
moritz@linux:~ > who
moritz   tty1     Feb 10 13:26
moritz   tty3     Feb 10 15:28
moritz   tty4     Feb 10 15:26
max      pts/0    Feb 10 14:05

moritz@linux:~ > who -Hi
BENU     LEIT     LOGIN-ZEIT   RUHIG VON
moritz   tty1     Feb 10 13:26 05:29
moritz   tty3     Feb 10 15:28 00:15
max      pts/0    Feb 10 14:05   .
root     pts/1    Feb 10 18:49 00:05

moritz@linux:~ > who –uT
root     +   tty1     Feb 10 13:26 05:29
moritz   -   tty3     Feb 10 15:28 00:15
moritz   -   tty4     Feb 10 15:26 00:30
max      +   pts/0    Feb 10 14:05 …
```

Anmerkung:

> Die Terminals **tty1** und **tty2** sind **Terminalsitzungen auf der Konsole,** das **Terminal pts/0 (Pseudoterminal)** ist die **erste geöffnete Terminalleitung (0) über das Netzwerk**.

3.7.5 whoami

zeigt dem Benutzer den Namen, unter dem er eingeloggt ist.

whoami

Wer es liebevoller mag, dem sei die Version "**who mom likes**" an Herz gelegt.

3.7.6 w

Der Befehl **w** zeigt wie **who** die eingeloggten Benutzer an. Er liefert allerdings weitergehende Informationen wie Einlogzeitpunkt, CPU-Nutzung und was der Benutzer gerade ausführt.

w [OPTIIONEN]

Beispiel:

```
moritz@linux:~ > w
  4:23pm  up  2:00,  6 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM         LOGIN@   IDLE   JCPU   PCPU  WHAT
moritz   tty1     -            2:24pm  1:58m 44.56s  0.04s  sh startx
root     tty2     -            4:22pm 22.00s  0.12s  0.06s  -bash
lori     tty3     -            4:23pm 14.00s  0.08s  0.05s  -bash
moritz   tty5     -            4:21pm  1:33   0.08s  0.04s  -bash
moritz   pts/0    -            4:21pm  0.00s  0.08s  0.03s  w
moritz   pts/1    -            3:37pm  2:05   2.32s  0.10s  bash
```

3.7.7 finger

Auch der Befehl **finger** zeigt Informationen über die eingeloggten Benutzer an. Dabei kann er eine einfache Übersicht über die Benutzer liefern oder ausführliche Informationen über einzelne Benutzer.

finger [OPTIONEN] [BENUTZER]

Der Befehl **finger** wertet für seine Ausgabe u.a. die Datei **/etc/passwd** aus.

Beispiele:

```
max@linux:~ > finger
Login      Name               Tty    Idle    Login Time      Where
max        Max Mustermann     3         5    Sun 16:23
root                          2         5    Sun 16:22
liese      Lieschen Müller    1      2:04    Sun 14:24
hans       Hänschen Klein    pts/0     -     Sun 16:21    (10.1.1.14)
```

Die genausten Informationen erhält man mit **finger -l**. Dabei werden zusätzlich die Dateien **.plan** und **.project** im Heimatverzeichnis des jeweiligen Benutzers ausgewertet,

sofern diese angelegt sind. Diese beiden Dateien geben Auskunft über zukünftige Pläne und Projekte.

Beispiel:

```
max@linux:~>$ finger -l
Login: max                              Name: Max Mustermann
Directory: /home/max                    Shell: /bin/bash
On since Wed Sep 25 12:56 (CEST) on :0 (messages off)
On since Tue Oct  8 14:22 (CEST) on pts/0 from 10.1.1.240
New mail received Mon Oct  7 21:30 2002 (CEST)
     Unread since Thu Aug  8 21:30 2002 (CEST)
Project:
Linux Grundlagen Akademie der Saarwirtschaft
Plan:
Urlaub vom 12.03.2001 - 22.03.2001

Login: root                             Name: root
Directory: /root                        Shell: /bin/bash
On since Tue Oct  8 11:55 (CEST) on pts/10 from linux.localhost
   4 hours 10 minutes idle
     (messages off)
New mail received Tue Oct  8 04:16 2002 (CEST)
     Unread since Tue Aug  6 04:15 2002 (CEST)
No Plan.
```

3.7.8 last

Der Befehl **`last`** zeigt die Benutzer an, die sich zuletzt eingeloggt haben. Er durchsucht dafür die Datei `/var/log/wtmp`. Das Kommando darf allerdings i.d.R. nur der Superuser **`root`** ausführen.

`last [OPTIONEN] [BENUTZER]`

Daneben zeigt er auch an, seit wann die Datei `/etc/log/wtmp` Informationen enthält.

```
max@linux:~ > last -10
moritz   tty3                    Sun Feb 11 16:23   still logged in
root     tty2                    Sun Feb 11 16:22   still logged in
moritz   pts/0                   Sun Feb 11 16:21   still logged in
max      pts/0                   Sun Feb 11 14:47 - 15:54  (01:07)
root     tty1                    Sun Feb 11 14:24   still logged in
reboot   system boot  2.2.14     Sun Feb 11 14:23         (04:32)
moritz   pts/2                   Sat Feb 10 19:09 - 01:05  (05:55)
max      pts/1                   Sat Feb 10 18:49 - 01:05  (06:15)

wtmp begins Mon Feb  5 20:18:53 2001
```

Kapitel 4

Das Linux Dateisystem

4.1 Partitionen

Festplatten müssen vor ihrer Benutzung als Speichermedium in Partitionen unterteilt werden, die jeweils mit voneinander unabhängigen Dateisystemen und von unterschiedlichen Betriebssystemen formatiert werden können. Notwendig sind solche Einteilungen, um eine Art Inhaltsverzeichnis auf der Festplatte einzurichten, anhand dessen die gespeicherten Daten effizient verwaltet werden können.

Um Partitionen einzurichten wird das Programm **`fdisk`** verwendet:

`fdisk [FESTPLATTE]` `z.B. fdisk /dev/hda`

Hinweis:

Eine Festplatte besteht aus mindestens einer Partition und kann max. 4 Partitionen (4 primäre oder 3 primäre und 1 erweiterte) angelegt werden. Sind weitere Partitionen erforderlich, können diese als logische Partitionen innerhalb der erweiterten Partition angelegt werden. Eine erweiterte Partition kann nicht direkt zum Speichern von Daten verwendet werden.

4.1.1 Partitionsnamen

Die Partitionen werden direkt nach dem Typ des Festplattencontrollers benannt. Dabei steht für IDE-Platten der Buchstabe **h** und für SCSII-Platten der Buchstabe **s**. Dann folgt der Buchstabe **d** für *Disk*. Der dritte Buchstabe nummeriert die Festplatte und die vierte Ziffer gibt die Partitionsnummer an. Die erste Partition der ersten Festplatte mit einem IDE-Controller heißt also `hda1`. Mit einem SCSII-Controller würde sie `sda1` heißen.

Name	Controller	Disk	Partition
`hda1`	IDE-Controller	Erste Festplatte	Erste Partition
`sda1`	SCSII-Controller	Erste Festplatte	Erste Partition
`hdb3`	IDE-Controller	Zweite Festplatte	Dritte Partition
`sdc4`	SCSII-Controller	Dritte Festplatte	Vierte Partition

Tabelle: Beispiele für Partitionsnamen

4.2 Dateiysteme und Mount Points

Auf einer Partition kann dann das Dateisystem (z.B. ext2, ext3, ReiserFS, etc.) angelegt werden. Dazu wird das Kommando **mkfs** verwendet. Das Kommando mkfs formatiert eine Partition mit dem angegeben Dateisystem.

```
mkfs [-t DSTYP] GERÄT [BLOCKS]
```

Beispiel:

mkfs –t vfat /dev/hda3	erstellt das vfat-Dateisystem auf der dritten Partition der ersten IDE Festplatte

Damit ein Filesystem benutzt werden kann, muss es mit dem Kommando **mount** in den Verzeichnisbaum eingebunden werden. Dabei wird der **mount point** (ein leeres Verzeichnis) durch das Wurzelverzeichnis des einzubindenden Filesystems ersetzt.

Mindestens einer Partition muss bei der Installation von Linux ein solcher Mount-Point zugeordnet werden, der **/** für das Root-Filesystem (Wurzeldateisystem).
Das Root-Filesystem wird bereits vom Kernel "gemountet", alle anderen Dateisysteme können auch per Hand nachträglich eingehangen werden:

Beispiel:

```
root@linux> mount /dev/hda1 /mnt/win
```

/dev/hda1 bezeichnet hier die erste Partition auf der ersten IDE-Festplatte, also das, was unter DOS/Windows als Laufwerk C: angesprochen wird.

4.2.1 /etc/fstab

Soll ein Filesystem beim Booten automatisch eingebunden werden, so muss eine entsprechende Zeile in die Datei **/etc/fstab** eingetragen werden.

Beispielsweises Listing der Datei /etc/fstab:

```
/dev/sda1         /boot           ext2         defaults   1 2
/dev/sda2         /               ext2         defaults   1 1
/dev/sda3         swap            swap         defaults   0 2
/dev/sda5         /home           ext2         defaults   1 2
proc              /proc           proc         defaults   0 0
usbdevfs          /proc/bus/usb   usbdevfs     defaults   0 0
devpts            /dev/pts        devpts       defaults   0 0
/dev/cdrom        /cdrom          auto         ro,noauto,user,exec 0 0
/dev/fd0          /floppy         auto         noauto,user 0 0

Gerätedatei       MountPoint      Typ des      Optionen(s. unten)
                                  Dateisystems
```

4.2.1.1 Optionen in der Datei /etc/fstab:

- **ro** = read only
  montiert das Dateisystem mit Nur-Lese-Recht
- **rw** = read/write
  montiert das Dateisystem mit Lese- und Schreibzugriff
- **auto**
  das Dateisystem wird automatisch beim Booten montiert
- **noauto**
  das Dateisystem wird nicht automatisch beim Booten montiert
- **user**
  erlaubt normalen Benutzern das Montieren des Dateisystems
- **nouser**
  verbietet normalen Benutzern das Montieren des Dateisystems
- **exec**
  erlaubt die Ausführung von Binärdateien in dem montierten Dateisystem
- **noexec**
  verbietet die Ausführung von Binärdateien in dem montierten Dateisystem
- **defaults**
  = rw, auto, nouser, exec
- **1** in **Spalte 5**
  das Dateisystem wird vor dem Aushängen gedumpt (= gesichert)
- **0** in **Spalte 5**
  das Dateisystem wird vor dem Aushängen nicht gedumpt (= gesichert)
- **1/2** in **Spalte 6**
  legt die Reihenfolge fest, in der die Dateisysteme beim Booten gecheckt werden
- **0** in **Spalte 6**
  das Dateisystem wird beim Booten nicht gecheckt

4.3 Arbeiten mit dem Dateisystem

4.3.1 mount

Der `mount`-Befehl bindet ein Dateisystem in den aktuellen Verzeichnisbaum ein.

```
mount [OPTIONEN] [GERÄT] MOUNTPOINT
```

wichtige Optionen

`-a`	Mounte alle Dateisysteme, die in `/etc/fstab` aufgeführt werden.
`-f`	Überprüfe ob das angegebene Dateisystem gemountet werden kann
`-t DSTYP`	Typ für das zu mountende Dateisystems

Beim Einbinden eines Dateisystems mit `mount` wird die Datei `/etc/fstab` ausgewertet. Wenn das Gerät in der Datei angegeben ist, so reicht für das Mounten die Angabe des Mount Points aus.

Beispiel

Ist der Mount Point in der /etc/fstab eingetragen, so reicht ein

```
user@linux:~> mount /floppy
```

aus.

Ist dies nicht der Fall, so kann das Gerät über

```
mount -t vfat /dev/fd0 /floppy
```

gemountet werden, hier mit Angabe des Dateisystemtyps vfat.

Auch die **Verzeichnisstruktur auf einer CD-ROM** oder **Diskette** wird als **Dateisystem** betrachtet. Um Zugriff auf die CD-ROM/Diskette zu haben, muß das betreffende Dateisystem montiert werden, siehe Bsp. oben.

4.3.2 umount

Der `umount`-Befehl entfernt ein Dateisystem aus dem aktuellen Verzeichnisbaum.

```
umount [OPTIONEN] [GERÄT] [MOUNTPOINT]
```

wichtige Optionen

`-a`	Unmounte alle Dateisysteme, die in `/etc/mtab` aufgeführt werden.
`-t DSTYP`	Typ für das zu unmountende Dateisystems

Hinweis:

Aktuell benutzte Dateisysteme können nicht demontiert werden!
Zur Ausführung der Kommandos **mount** und **umount** für ein Dateisystem muß **Benutzern** in der Datei **/etc/fstab explizit Berechtigung** gegeben werden (siehe unten).

4.3.3 Anzeige montierter Dateisysteme

Die **aktuell montierten Dateisysteme** sind in der Datei **/etc/mtab** gespeichert! Der Befehl **`mount`** ohne Parameter benutzt diese Datei für seine Ausgabe. Die Befehle **`mount`** und **`umount`** verändern die Datei. Wer also sehen will was zur Zeit gerade gemountet ist, gibt dazu den Befehl **`mount`** ohne Optionen ein oder **`cat /etc/mtab`**.

Der Befehl df (engl. **d**isk **f**ree) zeigt den freien Platz auf allen gemounteten Laufwerken an.

Beispielausgabe des Kommandos **df**:

```
Filesystem   1k-blocks        Used   Available   Use%   Mounted on

/dev/sda2       3099140    1273656    1668048     43%    /
/dev/sda1         15522       2757      11964     19%    /boot
/dev/sda5       5162532     177268    4723008     4%     /home
/dev/sda6        396500         13     376006     0%     /dos
usbdevfs         396500     396500          0     100%   /proc/bus/usb

Dateisystem     Anzahl      belegt      frei      proz.  Mountpoint-
Gerätedatei     Blöcke                            Beleg.
```

4.3.4 Überprüfen eines Dateisystems

Das wichtigste Programm für die Wartung des Dateisystems ist **`fsck`**. Es wird zur Überprüfung des Dateisystems und zur Behebung von Unstimmigkeiten verwendet.
In den meisten Fällen basiert ein solcher Dateisystemfehler auf einem Systemabsturz. Der Kernel ist dann nicht mehr in der Lage die im Cache zwischengespeicherten Daten auf die Platte zu schreiben.

```
fsck [OPTIONEN] [-t DSTYP] [GERÄT]
```

Optionen

`-a`	Das Programm läuft ohne Rückfragen ab
`-c`	Untersucht auf defekte Blöcke

Hinweis:

Vor dem Überprüfen sollte das betreffende Dateisystem (außer Root-Dateisystem!) demontiert werden. **`fsck`** kann nur vom Superuser root ausgeführt werden.

4.4 Verzeichnisstruktur

Linux speichert Dateien **im Gegensatz zu MS-DOS und Windows** in einem großen Verzeichnisbaum, der mit dem sog. Root- oder Wurzelverzeichnis beginnt und sich schnell weit verzweigt. Das Wurzelverzeichnis wird mit einem "**/**" dargestellt.
Auf diese Weise entseht ein hierarchischer Dateibaum, durch den sich der Benutzer hangeln kann. Dies wird auch als hierarchisch gegliedertes, geräteunabhängiges Dateisystem bezeichnet.
Die nachfolgende Grafik stellt nur einen Ausschnitt eines Systems dar und muss nicht in allen Einzelheiten auf jedes System zutreffen.

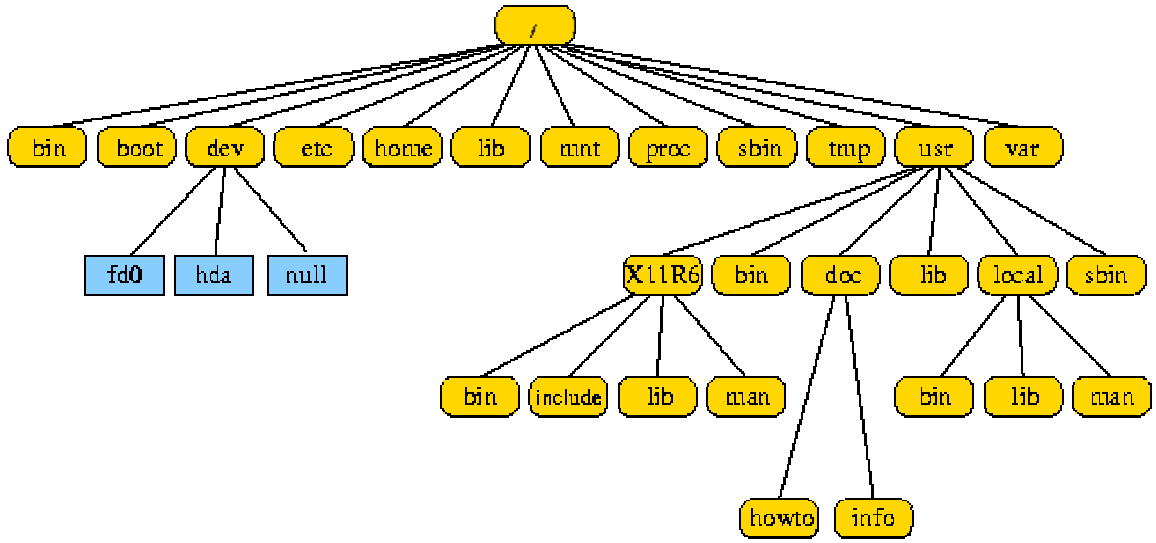


Abbildung: Die Linux-Verzeichnisstruktur

Lange Zeit hat jedes Unix-Derivat seine eigenen Vorstellungen vom Aufbau seiner Dateiverwaltung mitgebracht, aber unterdessen ist man sich mehr oder weniger einig geworden und erarbeitete den **Filesystem Hierarchie Standard**, der wichtige Strukturen definiert. Die meisten Distributionen folgen diesen Richtlinien, wobei Abweichungen oft durch unterschiedliche Auslegung des Standards begründet werden.
Neben der Beschreibung der vorgesehenen Verwendung jedes Verzeichnisses, werden auch konkrete Kommandos genannt, die mindestens in diesen Verzeichnissen vorhanden sein müssen. Des Weiteren finden systemspezifische Vorgaben und optionale Komponenten Erwähnung.

Sobald der Kernel aktiv ist, lädt er als erstes das Root-Dateisystem, in dem alle für die Aufgaben des Kernels notwendigen Programme und Konfigurationsdateien angesiedelt sein müssen.

Zu den Programmen gehören:

- Dienstprogramme zum Prüfen und Reparieren des Dateisystems
- Programme zum Sichern der Systemdaten und zur Installation neuer Systemteile
- Eventuell wichtige Netzwerkprogramme

4.4.1 Inhalt des Wurzelverzeichnisse

Wenden wir uns also zunächst dem Inhalt des Wurzelverzeichnisses zu.

/bin

Die wichtigsten Kommandos, um mit dem System arbeiten zu können, findet man hier. Sie dürfen von allen Nutzern ausgeführt werden. Zu den Kommandos gehören `cat, chgr, chmod, chown, cp, date, dd, df, dmesg, echo, ed, false, kill, ln, login, ls, mkdir, mknod, more, mount, mv, ps, pwd, rm, rmdir, sed, setserial, sh, stty, su, sync, true, umount, uname (sh` ist meist ein Link auf `bash)`.

/boot

Hier befinden sich die zum Hochfahren des Systems unbedingt erforderlichen Dateien. In der Hauptsache ist das der Kernel, die Datei mit dem Namen `vmlinuz` .

/dev

In diesem Verzeichnis stehen die Gerätedateien (Devices), die die gesamte Hardware beschreiben (Festplatte, Floppy, RAM...), sowie einige Devices mit speziellen Aufgaben.

Drei Informationen sind für jedes Device relevant:

(1) Art des Zugriffs:

- Blockorientiert (b) - gepufferter Zugriff, z.B. Festplatten
- Zeichenorientiert (c) - ungepufferter Zugriff, z.B. Bildschirm, Drucker

```
brw-rw-rw-   1 root     disk       2,   0 Nov  8 20:48 /dev/fd0
```

(2) Hauptgerätenummer (major device number):

- Nummer des zu verwendenden Treibers
- Unter jeder Nummer existiert je ein Treiber für zeichen- und blockorientierte Geräte (z.B. Nummer 2 für Terminals (c) und Floppys (b))
- Beschreibung vergebener Nummern unter»/usr/src/linux/Documentation/devices.txt«

```
brw-rw-rw-   1 root     disk       2,   0 Nov  8 20:48 /dev/fd0
```

(3) Nebengerätenummer (minor device number):

- Nummer der zuständigen Routine in einem Treiber
- Dient z.B. zur
  - Unterscheidung der Diskettenformate im Floppytreiber
  - Für ein zweites CD-ROM
  - usw.

```
brw-rw-rw-   1 root     disk      2,   0 Nov  8 20:48 /dev/fd0
```

Wichtige Gerätedateien sind:

cdrom

Link auf eine entsprechende Datei (z.B. cdu535)

cua*

(Veraltetes) Devices für serielle Schnittstellen, das für ausgehende Modemverbindungen verwendet wurde (und in manchen Distributionen noch immer wird). Physisch zeigt ein solches Device auf dasselbe Gerät wie /dev/ttys*, jedoch blockiert ein Programm nicht, wenn es das Gerät eröffnet und noch kein Verbindungssignal anliegt. Aktuell sollten die Schnittstellen /dev/ttyS* bevorzugt werden.

fd*

Diskettenlaufwerke

hd*

IDE-Festplatten

kmem

Speicherauszug (core)

lp

Parallele Schnittstellen

mouse

Link auf die entsprechende Datei

port

IO-Ports

sd*

SCSI-Festplatten

tty*

Terminalkonsolen

ttys*

(Veraltetes) Device für die seriellen Schnittstellen, das vornehmlich zur Überwachung eingehender Verbindungen genutzt wurde. Ein Programm, das diese Datei eröffnet, wird blockiert, solange das Modem kein »Carrier Detect« meldet. Aktuell sollten die Schnittstellen /dev/ttyS* bevorzugt werden.

ttyS*

(Neues) Device für die seriellen Schnittstellen, das sowohl für eingehende als auch für ausgehende Verbindungen genutzt werden sollte. Die bei /dev/cua* und /dev/ttys* angedeuteten Probleme mit blockierenden Programmen werden vollkommen durch den Kernel behandelt. Somit ist es (ohne Umwege) möglich, auf einer Schnittstelle auf eingehende Verbindungen zu warten. Solange eine solche Verbindung nicht eröffnet wurde, kann auf derselben Schnittstelle eine ausgehende Verbindung eröffnet werden. Für den Zeitraum einer aktiven Verbindung bleibt der jeweils andere Prozess blockiert.

/etc
Hier sind viele der Konfigurationsdateien untergebracht, die die Einstellungen verschiedener Programme oder auch grundlegende Systeminformationen enthalten.

/home
Alle Heimatverzeichnisse der Nutzer findet man hier. Nach dem Login landet jeder Benutzer (i.d.R.) in seinem Home. Heimatverzeichnisse können vom Systemverwalter auch an anderer Stelle angesiedelt werden.

/lib
Die beim Systemstart benötigten Bibliotheken stehen hier. Ebenso liegen die Kernelmodule in einem eigenen Unterverzeichnis unterhalb von `/lib`.

/mnt
Mountpunkt für temporäre Partitionen

/opt
Software, die nicht zum üblichen Installationsumfang von Unix-Systemen gehören, werden oft unter diesem Zweig installiert.

/root
Heimatverzeichnis des Administrators. In realen Unix-Installationen werden die Heimatverzeichnisse aller Nutzer oft auf einem Server gehalten. Bei einem Ausfall eines solchen Servers sollte aber zumindest Root in der Lage sein, vernünftig mit dem System zu arbeiten. Daher liegt dessen Heimatverzeichnis direkt unterhalb der Verzeichnisbaumwurzel.

/sbin
Wichtige System-Programme (beim Booten benötigt; Ausführung erfordert Root-Rechte)

/tmp
Temporäre Dateien können hier abgelegt werden, jeder Nutzer ist dazu berechtigt.

/usr
Die umfangreichste Verzeichnisstruktur des Systems. Hier liegt der größte Teil der installierten Software. Auf vielen Systemen befinden sich in und unterhalb von /usr mehr Daten als in allen anderen Dateien zusammen.

/var
Unter diesem Verzeichnis werden hauptsächlich Dateien gespeichert, die sich ständig verändern. Der Name /var steht für **variabel**, also **veränderlich**. Hier befinden sich beispielsweise die Logdateien.

/proc

/proc ist eigentlich kein normales Verzeichnis, sondern stellt eine Schnittstelle zum Kernel dar. Jedes laufende Programm wird hier in einem Unterverzeichnis geführt, dessen Dateien viele Informationen z.B. über den aktuellen Programmstatus enthalten. Zudem gibt es eine umfangreiche Verzeichnisstruktur mit Daten über den Kernel und die Hardware des Systems.

4.5 Adressierung von Dateien und Verzeichnissen

4.5.1 Pfadnamen

Soll eine Datei oder Verzeichnis mit einem Kommando angesprochen werden, muss immer der Weg zu dieser Datei beschrieben werden, es sei denn, die Datei liegt im aktuellen Verzeichnis. Der Weg durch die Hierachie der Dateiverzeichnisse wird auch Pfad (path) genannt. Mit der Angabe dieses Weges, dem Pfadnamen, kann jede Datei im Dateisystem eindeutig angesprochen werden.

Ein Pfadnamen besteht entweder aus verschiedenen Dateinamen, die durch einen Schrägstrich / getrennt werden, z.B /verzeichnis1/unterverzeichnis1/unterunter1, oder aus einzelnen Namen. Alle Dateinamen , die durch einen / getrennt werden, müssen Dateiverzeichnisse sein, nur der letzte Name in der Liste kann eine normale Datei sein.

Es werden zwei Möglichkeiten der Pfadangabe unterschieden:

- den absoluten Pfad
- den relativen Pfad

4.5.1.1 Absoluter Pfad

Absolute Pfadnamen beginnen mit dem Root-Verzeichnis "/" und führen über alle Verzeichnisnamen hin zum gewünschten Dateinamen. Der absolute Pfad beginnt also immer ganz oben im Hauptverzeichnis und beschreibt von dort den Weg ins Ziel.

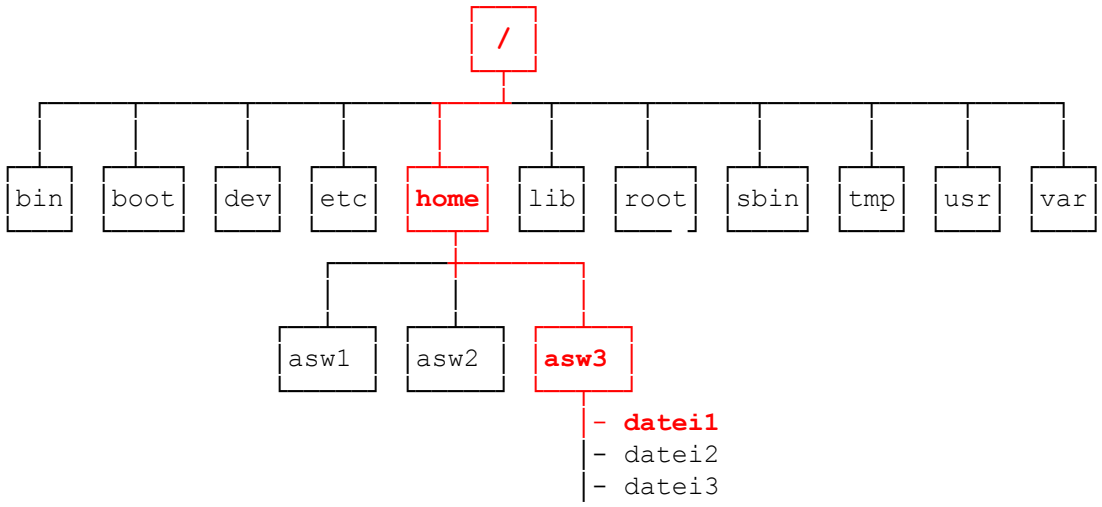


Abbildung: Absolute Pfadangabe

Beispiel:

```
cat /home/asw3/datei1
```

Die Eingabe beginnt mit dem Wurzelverzeichnis "**/**", führt über das Verzeichnis *home* und das Unterverzeichnis *asw3* bis hin zur Datei *datei1*.

Bei dieser Form der Pfadangabe spielt der eigene Standort im Dateissystem keine Rolle, denn der beschriebene Weg von ganz oben zum Ziel ist immer der gleiche, egal, wo der Benutzer sich selbst befindet.

Merke:

Der absolute Pfad beginnt **immer** mit einem "/".

4.5.1.2 Relativer Pfad

Relative Pfadangaben dagegen gehen immer von der Stelle im Dateibaum aus, an der sich der Benutzer gerade befindet. Diese aktuelle Verzeichnis wird auch "current directory" genannt. Vom aktuellen Verzeichnis wird der relative Pfad zurückverfolgt. Steht der Anwender beim Aufruf Eines relativen Pfadnamens nicht im richtigen Verzeichnis, ist der Befehl nicht erfolgreich.

Beispiel:

```
cat ../asw3/datei1
```

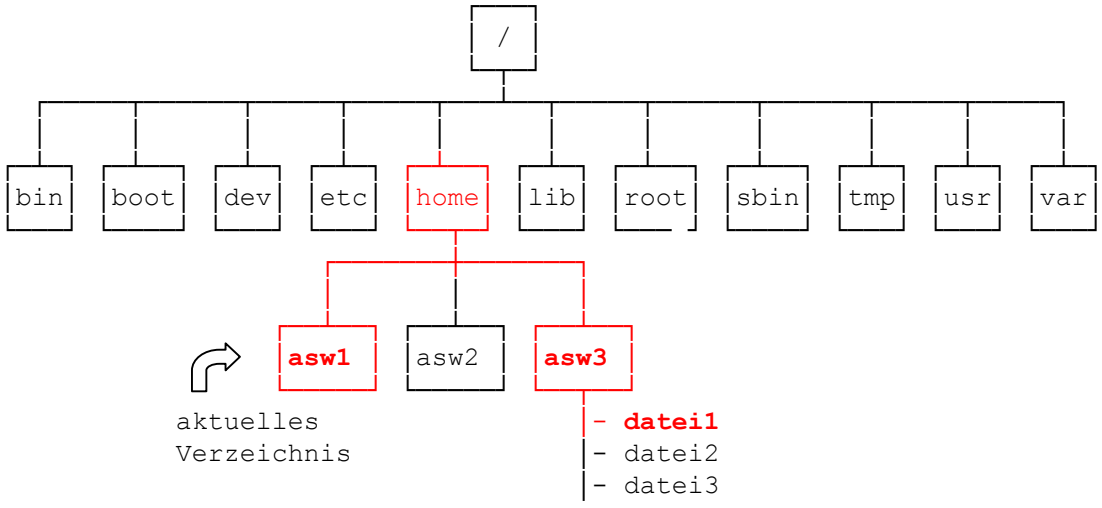


Abbildung: Realtiver Pfad, Wechsel in der Hierarchie nach oben

Das bedeutet, wenn sich der Anwender im Verzeichnis **`asw1`** befindet, kann er sich auf diese Weise die Datei **`datei1`** im Verzeichnis **`asw3`** anzeigen lassen, wobei

".." dem aktuellen Verzeichnis übergeordnete Verzeichnis darstellt.

Hinweis:

Jedes Verzeichnis unter Linux enthält automatisch zwei Unterverzeichnisse: das Verzeichnis ".", das ein symbolischer Name auf das aktuelle Verzeichnis selbst ist, und as Verzeichnis "..", das auf das darüberliegende Elternverzeichnis (parent directory) verweist.

- `.` aktuelles Verzeichnis
- `..` dem aktuellen Verzeichnis übergeordnetes Verzeichnis

Merke:

Der relative Pfad beginnt **niemals** mit einem "`/`".

Vom Verzeichnis **`asw1`** ausgehend kann der Anwender auch mit dem Befehl

```
cd tmp
```

in das Verzeichnis `tmp` wechseln.

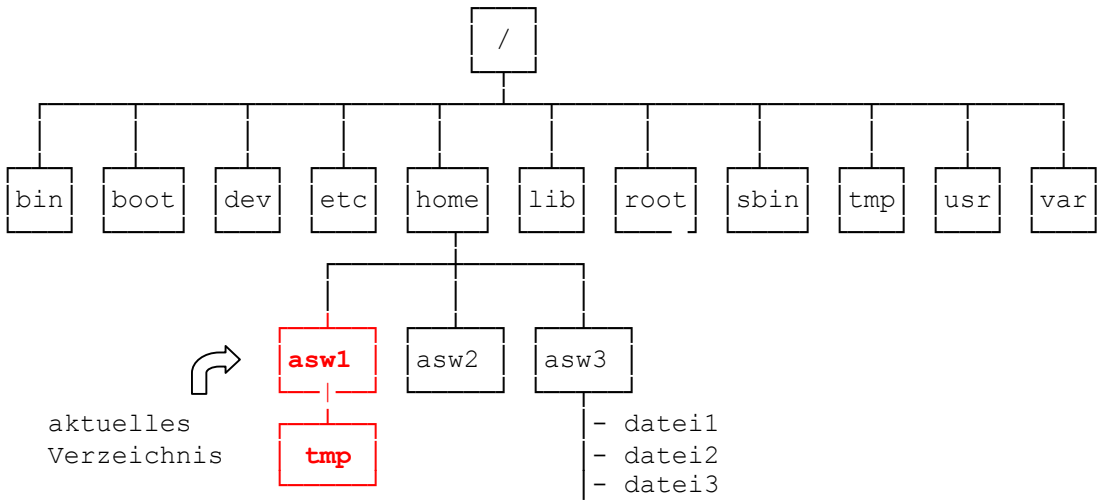


Abbildung: Realtiver Pfad, Wechsel in der Hierarchie nach unten

Bei dieser Form der Pfadangabe wird verwendet, wenn vom eigenen Standort ausgehend der Weg in der Hierarchie nach unten geht. Um in der Hierarchie nach oben zu wechseln, wird das Symbol für das übergeordnete Verzeichnis (..) verwendet (siehe oben).

4.6 Arbeiten mit Verzeichnissen

Nach dem Einloggen befinden Sie sich immer in Ihrem Heimatverzeichnis. Dies wird generell abgekürzt durch das Tildezeichen "**~**".

Beispiel:

> **~max** ist gleichbedeutend mit **/home/max**

4.6.1 pwd

Der Befehl **pwd** (*Print Working Directory*) zeigt das Verzeichnis an, in dem Sie gerade arbeiten.

```
user@linux> pwd
/home/user/asw/grundlagen
```

4.6.2 cd

Um in ein anderes Verzeichnis zu wechseln benutzt man das Kommando **cd** (*Change Directory*).

```
cd [DIRECTORY]
```

Der Aufruf des Kommandos `cd` **ohne** Parameter bringt Sie wieder zurück in Ihr Homeverzeichnis.

Zum Beispiel bringt Sie das Kommando

```
cd /
```

in das Wurzelverzeichnis.

Das Kommando

```
cd /dev
```

wechselt in das Verzeichnis /dev. Der Schrägstrich (Slash) bezeichnet dev vom Wurzelverzeichnis aus (siehe oben absoluter Pfad). Es können auch mehrere Verzeichnisnamen zu einem Pfadnamen verkettet werden. Um die einzelnen Namen voneinander zu trennen, werden weitere Slashes verwendet. Das Kommando

```
cd /usr/lib/emacs
```

bringt Sie in das Verzeichnis **emacs**. Um von dort aus in das Verzeichnis **lib** zu gelangen, können Sie entweder wieder den *absoluten* Pfad vom Wurzelverzeichnis aus angegeben

```
cd /usr/lib
```

oder Sie benutzen den *relativen* Pfad vom aktuellen Verzeichnis aus:

```
cd ..
```

4.6.3 mkdir

Der Befehl **mkdir** wird dazu benutzt neue Verzeichnisse zu erstellen.

```
mkdir [OPTIONEN] [VERZEICHNISPFADLISTE]
```

Wird die Option **-p** nicht mit angegeben, so müssen die Elternverzeichnisse für das neue Verzeichnis existieren.

Optionen

-p	Erzeugt auch die nötigen Elternverzeichnisse, wenn diese nicht existieren.

Beispiele

mkdir tex/linux

Dieses Kommando legt das Verzeichnis `tex/linux` an. Dabei **muß** das Verzeichnis `tex` existieren.

mkdir -p tex/linux/kurs/material

Dieser Befehl legt einen kompletten Verzeichnispfad an. Dabei werden alle Verzeichnisse erzeugt, wenn sie noch nicht existieren.

4.6.4 rmdir

Um **leere** Verzeichnisse wieder zu löschen, wird der Befehl `rmdir` verwendet.

```
rmdir [OPTIONEN] [VERZEICHNISPFADLISTE]
```

Dabei wird immer das letzte Verzeichnis in der angegebenen Verzeichnishierachie gelöscht. Der Schalter **-p** ermöglicht das Löschen ganzer Hierarchien.

Optionen

-p	Löschen aller leeren Verzeichnisse in der Verzeichnishierachie.

Beispiele:

Dieses Kommando löscht alle leeren Verzeichnisse des Elternverzeichnisses `linux`

rmdir tex/linux/*

Soll die ganze Hierachie gelöscht werden, also auch die Verzeichnisse `tex/linux` und `tex` gelöscht werden, so lautet der Befehl:

```
rmdir -p tex/linux/*
```

`rmdir` arbeitet nur, wenn die Verzeichnisse keine weiteren Verzeichnisse oder Dateien enthalten.

Für das Löschen von Verzeichnissen samt Inhalt, wird der Befehl

rm –rf [VERZEICHNISNAME]

verwendet. Das ist einer der berüchtigsten Befehle unter Linux, da ohne Nachfrage alles gelöscht wird. Bei einem Vertippen oder falscher Pfadangabe, kann das gesamte System zerstört werden.

Beispiel:

Sie wollen das Verzeichnis **/muell** samt Inhalt löschen und geben den Befehl

rm –rf / muell

ein, d.h. aus Versehen ist ein Leerzeichen dazugekommen. Dieser Befehl löscht Ihnen ohne Nachfrage das Wurzelverzeichnis und das System ist hinüber!!!

4.6.5 ls

Um den Inhalt eines Verzeichnisses anzuzeigen, gibt es das Kommando **`ls`** (list) .

ls [OPTIONEN] [DATEINAMEN/VERZEICHNISNAME]

Beispiel:

```
user@linux> ls /bin

arch      compress  echo      gzip      ls        ps        su
bash      cp        ed        hostname  mkdir     pwd       sync
cat       date      false     kill      mknod     rm        tar
chgrp     dd        free      killall   mv        rmdir     true
chmod     df        ftp       ln        netstat   sh        uname
chown     du        gunzip    login     ping      stty      zcat
…
```

zeigt alle Dateien des bin Verzeichnisses an.

Durch Optionen in der Kommandozeile können Sie das Ausgabeformat des **`ls`** Kommandos ändern. Nachfolgende Optionen sind zunächst ausreichend.

Optionen

`-l` (long) für ein ausführliches Listing.

`-a (all)` für die Anzeige aller Dateien, auch versteckter Dateien, sog. hidden files.

`-t` sortiert die Ausgabe nach datum, nicht nach Namen

Beispiel:

```
user@linux> ls -l /

total 302
drwxr-xr-x   2 ruth     bin          2048 Jun 10 12:39 bin
drwxr-xr-x   2 ruth     ruth         1024 Dec  2 16:46 boot
drwxr-xr-x   2 ruth     ruth         3072 Jul 23 12:33 dev
drwxr-xr-x   8 ruth     bin          2048 Jun 15 12:18 etc
....
-rw-r--r--   1 ruth     ruth       281092 Dec  2 16:54 vmlinuz
```

zeigt beispielsweise alle Dateien des Root Verzeichnisses ausführlich an.

4.7 Dateiverwaltung

Unter Linux wird **jede physikalische Einheit** (z.B. Festplatten, Drucker, Terminals) **oder logische Einheit** (u.a. Verzeichnisse, normale Dateien) **als Datei im Linux-Dateisystem** dargestellt!
Neben den reinen Nutzdaten gehören zu einer Datei auch Metadaten, die die Datei beschreiben. Im UNIX-Filesystem sind die Nutzdaten und die Metadaten getrennt. Die Metadaten werden in **Inodes (information nodes)** abgelegt. Diese verweisen dann auf eine Liste von Blöcken auf dem Massenspeicher, in denen sich die Nutzdaten befinden. Zusätzlich zu den o.a. Metadaten hat ein inode noch eine Nummer, über die dieser inode angesprochen werden kann. **Jedes Dateisystem** hat eine **eigene Inode-Tabelle** zur Verwaltung dieser Inodes.

4.7.1 Aufbau eines ext2-Filesystem

Das ext2-Dateisystem besteht aus dem Bootblock und den Blockgruppen. Die Blockgruppen sind wiederum aufgeteilt in einen Superblock, einer Liste der Blockgruppenbeschreibungen, Bitmaps für Blöcke und Inodes, Inode-Tabelle und den Datenblöcken.

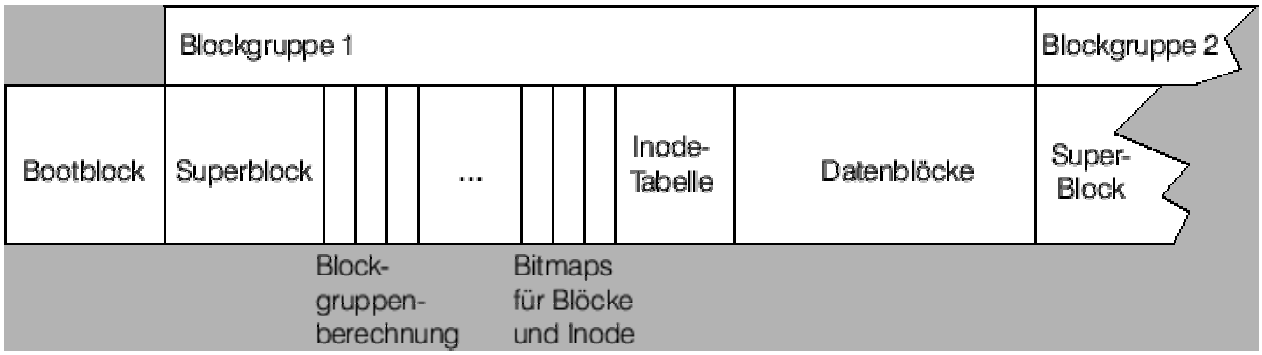


Abbildung: Aufbau ext2 Filesystem

Bootblock
Der Bootblock liegt im ersten Block (Block 0) des Dateisystems. Er enthält ein Programm zum Starten und Initialisieren des gesamten Systems. Meistens enthält nur das erste Dateisystem einen Bootblock. Beim ext2-Dateisystem ist der Bootblock 1024 Zeichen groß.

Blockgruppe
Jede Blockgruppe, wie in der Abbildung zu sehen, besteht aus sechs Komponenten:

- Superblock
- Liste der Blockgruppenbeschreibungen
- Bitmap für Blöcke
- Bitmap für Inodes
- Inode-Tabelle
- Datenblöcke

Die Standardgröße für eine Blockgruppe liegt bei 8192 Blöcken und 2048 Inodes, von denen acht reserviert sind.

Superblock
Die Informationen über das gesamte Dateisystem liegen im Superblock. Wenn dieser Superblock beschädigt wird, kann das Dateisytem nicht mehr ins Dateisystem eingebunden (gemountet) werden. Deswegen werden in regelmäßigen Intervallen Backup-Kopien dieses Blocks im Dateisystem angelegt. Im Normalfall erfolgt das alle 8192 Blöcke. Das heißt die Daten befinden sich in Block 8193, 16385, 24577 etc.

Die wichtigsten Informationen sind:

- Gesamtgröße des Dateisystems in Blöcken bzw. Inodes
- Anzahl der freien Blöcke
- Anzahl der Blöcke für die Inodes
- Anzahl der freien Inodes
- Adresse des ersten Datenblocks
- Größe eines Datenblocks
- Größe eines Teildatenblocks
- Größenbestimmung für eine Blockgruppe
- Zeitpunkt des Einbinden des Dateisystems
- Zeit der letzten Änderung
- Anzahl der Einbindungen (*mount counts*)
- Maximale Anzahl der Einbindungen bevor ein Dateisystemcheck durchgeführt wird
- Versionsnummer des Dateisystems
- Magische Nummer: für ext2 lautet sie `0xEF53`.
- Nummer des einrichtenden Betriebssystems
- Status des Dateisystems: sauber oder mit Fehlern
- Verhalten bei Fehlererkennung
- Zeitpunkt der letzten Prüfung
- Maximaler Zeitraum zwischen zwei Prüfungen

Liste der Blockgruppenbeschreibungen
Hier werden die wichtigsten Kenndaten der Blockgruppe festgehalten. Dies sind

- Blocknummer für Block-Bitmap
- Blocknummer für Inode-Bitmap
- Blocknummer für Inode-Tabelle
- Anzahl der freien Blöcke
- Anzahl der freien Inodes
- Anzahl der Verzeichnisse

Block- und Inode-Bitmap
Die Bitmaps für die Verwaltung der Blöcke und Inodes sind auf einen logischen Block beschränkt. Für jeden Block wird ein Bit benötigt. Daher ist eine Blockgruppe auf 1024*8 = 8192 Blöcke beschränkt.
Inode-Liste
Die Inode-Liste besteht aus vielen aneinandergereihten Inodes, die alle exakt die gleiche Größe haben. Im Standardfall sind es 2048 Stück, die innerhalb einer Blockgruppe gespeichert werden. Die Inode ist der Dreh- und Angelpunkt des Dateizugriffs.

In der Inode sind die **folgenden Informationen** gespeichert:
- Dateityp (gewöhnliche Datei, Verzeichnis, Gerätedatei, ...)
- Rechtetabelle
- Anzahl der Links (bzw. Dateinamen)
- Dateibesitzer (uid)
- Besitzergruppe (gid)
- Größe in Byte (bei Geräteadressen statt dessen Major und Minor Device number)
- Datum der letzten Änderung
- Datum des letzten Zugriffs
- Datum der letzten Änderung dieser Verwaltungsinformationen
- Attribute (nur für das Dateisystem ext2)
- physikalischer Ort der Speicherung (Blöcke in denen die Datei abgelegt ist)

Adressierung des Speicherblocks
In der Inode befinden sich auch die Informationen über die für die Datei verwendeten Datenblöcke. Diese Tabelle besitzt 15 Einträge.
Die ersten zwölf Einträge verweisen direkt auf jeweils einen Datenblock. Im Normalfall ist dieser logische Block 1024 Bytes groß. Es können also direkt 12 KB adressiert werden. Der Zugriff ist aufgrund des Mechanik des Caches sehr schnell.
Reichen die in der Inode referenzierten Blöcke für eine Datei nicht aus, zeigt der Eintrag in der Inode auf einen weiteren Inode, welcher nun die eigentlichen Verweise zu den Datenblöcken beinhaltet.
So weißt der dreizehnte Eintrag auf einen weiteren Datenblock. Dieser Datenblock enthält aber wiederum bis zu 256 Adressen von Datenblöcken. Es sind also nun 256 KByte adressierbar. Man spricht von einem **einfach indirekten Verweis**.
Der vierzehnte Eintrag weißt auf einen Datenblock der 256 Einträge vom Typ des dreizehnten Eintrags enthält. Er adressiert hier also 256*256 Blöcke mit 64 MByte Daten, sog. **doppelt indirekter Verweis.**
Der fünfzehnte Eintrag weißt schließlich auf einen Datenblock mit 256 Einträgen vom Typ des vierzehnten Eintrags. Hiermit werden also 256*256*256 Blöcke mit ca. 16,6 GByte adressiert, **dreifach indirekter Verweis.**
Damit kann eine maximale Dateigröße von ca. 16 GByte erzielt werden.

Veranschaulichte Darstellung:

1. Adresse	Block mit 1024 Zeichen			
2. Adresse	Block mit 1024 Zeichen			
3. Adresse	Block mit 1024 Zeichen			
4. Adresse	Block mit 1024 Zeichen			
5. Adresse	Block mit 1024 Zeichen			
6. Adresse	Block mit 1024 Zeichen			
7. Adresse	Block mit 1024 Zeichen			
8. Adresse	Block mit 1024 Zeichen			
9. Adresse	Block mit 1024 Zeichen			
10. Adresse	Block mit 1024 Zeichen			
11. Adresse	Block mit 1024 Zeichen			
12. Adresse	Block mit 1024 Zeichen			
13. Adresse	**einfacher indirekter Verweis auf einen Block mit**	256 Adressen von Datenblöcken der Datei		
14. Adresse	**doppelt indirekter Verweis auf einen Block mit**	256 Adressen von Blöcken mit	jeweils 256 Adressen von Datenblöcken der Datei	
15. Adresse	**dreifach indirekter Verweis auf einen Block mit**	256 Adressen von Blöcken mit	jeweils 256 Adressen von Blöcken mit	jeweils 256 Adressen von Datenblöcken der Datei
Summe	**12 x 1024 Byte = 12 KB +**	**256 x 1 KB = 256 KB +**	**256 x 256 x 1 KB = 65.536 KB +**	**256 x 256 x 256 x 1 KB = 16.777.216 KB**

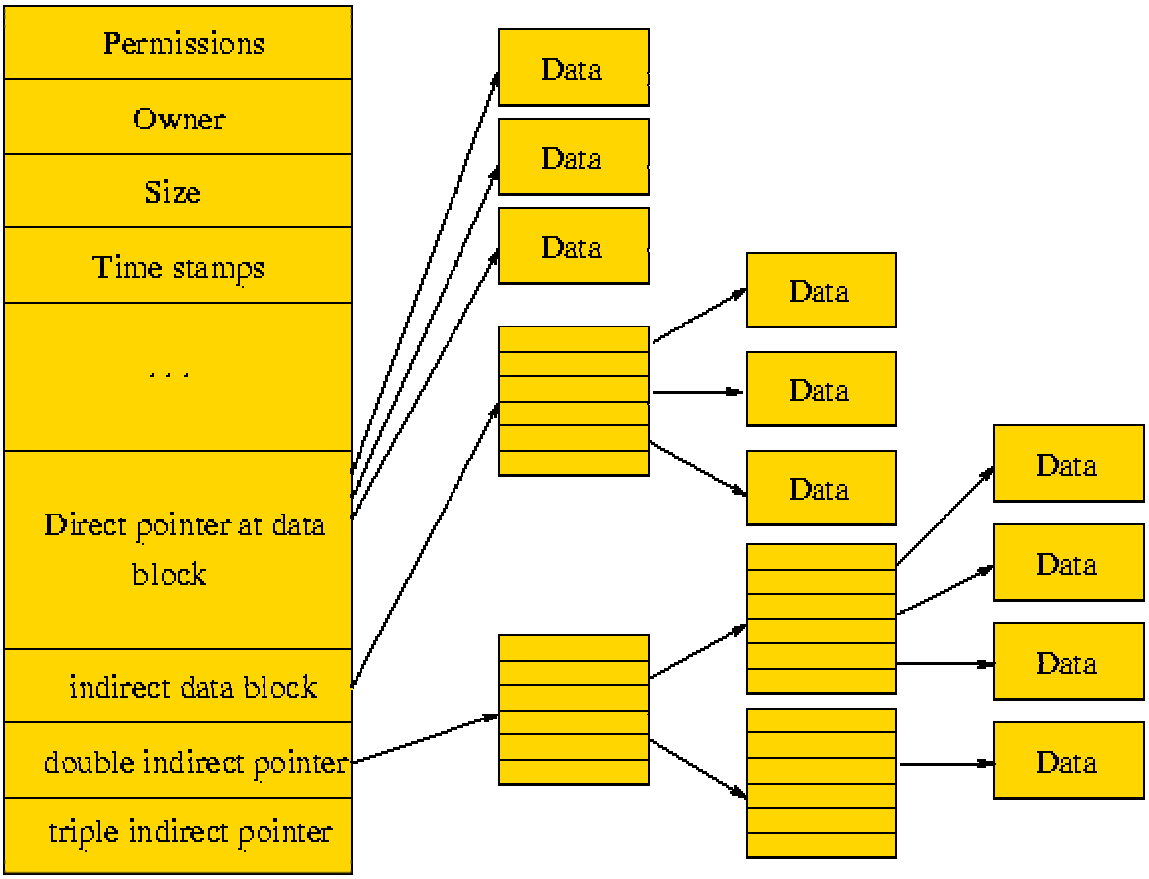


Abbildung: Speicherung der Dateien mit Hilfe von INodes

4.7.2 Wichtigste Merkmale des Linux-Filesystem ext2fs

- Dateinamen können bis zu 255 Zeichen lang sein
- Pfadangaben werden zum Unterschied von DOS mit einem normalen Schrägstrich gemacht.
- Wird als **erstes Zeichen** in einem Dateinamen der **Punkt** verwendet, so ist diese **Datei** "**hidden**", d.h. sie wird bei einem Dateilisting mit dem Kommando *ls* nicht angezeigt
- Groß- und Kleinschreibung wird unterschieden
- Alle Zeichen sind erlaubt, Format-Einschränkungen gibt es nicht. Es ist sogar möglich, einen aus Leerzeichen bestehenden Dateinamen zu bilden:

```
user@linux > touch "     "
user@linux > ls-l
-rw-r—r--   1  user   users   0  Nov  6 07:23
user@linux > rm "     "
```

4.7.3 Dateiarten unter Linux

Grundsätzlich unterscheidet Unix die folgenden Dateiarten:

- Reguläre Dateien (regular file)
- Verzeichnisse (directory)
- Gerätedateien
- Symbolische Links (symbolic link)
- Feststehende Programmverbindungen (named pipe oder FIFO)

Das erste Zeichen des Dateizugriffsstring (darstellbar mit **`ls -l`**) zeigt die Dateiart an. ein Bindestrich steht für eine normale Datei, ein d für ein Verzeichnis, ein l für einen symbolischen Link, ein b bzw. c für ein block- bzw. zeichenorientiertes Gerät, ein p für eine named pipe und ein s für einen Socket.

4.7.3.1 Reguläre Dateien

Reguläre Dateien sind alle "normalen Dateien", also das, was wir auch unter DOS als Dateien kennen. Dies können einfache Texte sein, die einfach als Zeichenfolge abgelegt werden; es können ausführbare Programme im Binärcode sei, Dateien aus einer Tabellenkalkulation oder z.B. auch ein Bild aus einer Grafikanwendung.

Dateityp: -

Beispiel:

```
-rw-r--r--   1   ukp22   wio22   6243   Nov 20 2001   test.txt
```

4.7.3.2 Verzeichnis

Verzeichnisse sind unter Unix auch nur Dateien, die allerdings eben den Typ **d** statt Bindestrich haben um sie von regulären Dateien zu unterscheiden und enthält die Namen der darin gespeicherten Dateien und Unterverzeichnisse mit ihrer Inode! Nur bei Symbolischen Links (siehe unten) ist zusätzlich der Pfad zu der Originaldatei im Verzeichniskatalog gespeichert!

Dateityp: d

Beispiel:

```
drwxr-xr-x   2   ukp22   wio22   4096   Nov 13 2001   tmp
```

4.7.3.3 Gerätedateien

Unix betrachtet auch jedes Stück Hardware als Datei. Dabei wird zwischen block- und zeichenorientierten Geräten unterschieden.

Die Gerätedateien (engl. special files), sowohl die block- wie auch die zeichenorientierten befinden sich im Verzeichnis /dev. Sie belegen physikalisch keinen Platz auf der Festplatte, sind also selbst nicht die Gerätetreiber, wie oft fälschlicherweise gedacht wird. Diese Dateien haben eigentlich nur zwei Nummern, die die Adresse des Gerätetreibers im Kernel symbolisieren. Die erste Nummer, auch major number genannt zeigt, um welchen Gerätetreiber es sich handelt, die zweite (minor number) beschreibt, das wievielte Gerät es ist, das diesen Treiber benutzt.

(a) Blockorientierte Geräte

Als blockorientiertes Gerät gilt jedes Gerät, das nicht einzelne Zeichen verarbeitet, sondern ganze Blocks. Dabei handelt es sich z.B. typischerweise um Festplatten, Disketten und andere Laufwerke. Diese Laufwerke werden dann als Dateien verwaltet, für den Anwender bleibt der Unterschied zwischen Gerät und Datei nahezu transparent.

Dateityp: b

Beispiel:

```
                                    major     minor
                                    number    number
                                      |         |
brw-rw----   1 root     disk         3,        0       Apr  5 01:12 hda
brw-rw----   1 root     disk         3,        1       Apr  5 01:12 hda1
brw-rw----   1 root     disk         3,        2       Apr  5 01:12 hda2
brw-rw----   1 root     disk         3,        3       Apr  5 01:12 hda3
brw-rw----   1 root     disk         3,        4       Apr  5 01:12 hda4
…
```

Typische blockorientierte Geräte sind hier also der Arbeitsspeicher (ram1-4 Major-Number 1), die Diskettenlaufwerke (fd0, fd1 Major Number 2) und die Festplatten. (Major Number 3 für IDE-Platten) Bei den Festplatten wird unterschieden zwischen den ganzen Platten (hda, hdb, hdc, hdd) und den darauf befindlichen Partitionen (hda1, hda2, hdb1, hdb2,...)
Auf diese Gerätedateien kann zugegriffen werden, wie auf jede normale Datei, sie haben einen Zugriffsmodus (rwx...) wie normale Dateien und Eigentümer und Gruppenzugehörigkeit.

Damit ist es z.B. möglich den **copy**-Befehl mit dem Befehl **cat** wie folgt zu realisieren:

```
cat Datei > /dev/fd0
```

(b) Zeichenorientierte Geräte

Grundsätzlich trifft alles, was wir für die blockorientierten Geräte oben gesagt haben auch auf die zeichenorientierten Geräte zu, nur daß es sich bei ihnen um Geräte handelt, die nicht blockweise angesteuert werden sondern durch einzelne Bytes. **Zeichenorientierte Geräte** verarbeiten die Ein-/Ausgaben als Zeichenstrom, der sequentiell - also unstrukturiert - läuft. Bei diesen Geräten erfolgt keine Zwischenpufferung der Daten. Typischerweise handelt es sich hier um serielle oder parallele Schnittstellen, auch und gerade die Terminalleitungen (auch die der virtuellen Terminals) aber auch etwa der Soundanschluß.

Dateityp: c

Beispiel:

Terminalleitungen für die virtuellen Terminals

```
crw-rw-rw-   1 root     root       5,   0 Apr  5 01:12 tty
crw--w--w-   1 root     tty        4,   0 Apr  5 01:12 tty0
crw-------   1 root     root       4,   1 Aug  8 11:39 tty1
```

Serielle Leitungen (COM1 und COM2)

```
crw-rw----   1 root     uucp       4,  64 Apr  5 01:12 ttyS0
crw-rw----   1 root     uucp       4,  65 Apr  5 01:12 ttyS1
```

Wichtige Gerätedateien sind:

Festplattenlaufwerke:

- /dev/hda — erste IDE Festplatte
- /dev/hda1 bis /dev/hda15 — Partitionen der ersten IDE Festplatte
- /dev/sda — erste SCSI Festplatte
- /dev/sda1 bis /dev/sda15 — Partitionen der ersten SCSI Festplatte
- /dev/sdb — zweite SCSI Festplatte
- /dev/sdc — dritte SCSI Festplatte

Diskettenlaufwerke:

- /dev/fd0 — erstes Floppylaufwerk
- /dev/fd1 — zweites Floppylaufwerk

CD-ROM Laufwerke:

- /dev/cdrom — **Link** auf das verwendete CD-ROM Laufwerk wie z.B.
- /dev/mcd — für Mitsumi CD-ROM oder
- /dev/scd0 bis /dev/scd1 — SCSI CD-ROM Laufwerke

Bandlaufwerke:

- /dev/rmt0 — erster SCSI Streamer "rewinding"
- /dev/ftape — Floppy-Streamer "rewinding"

Mäuse:

- /dev/mouse — **Link** auf die verwendete

Modem:

- /dev/modem — **Link** auf den com-Port, an den das Modem angeschlossen ist

Serielle Schnittstellen:

- /dev/ttyS0 bis /dev/ttyS3 Serielle Schnittstellen 0 bis 3

Parallele Schnittstellen:

- /dev/lp0 bis /dev/lp2 Parallele Schnittstellen **lpt1 bis lpt3**

Spezielle Devices:

- /dev/tty1 bis /dev/tty8 Virtuelle Konsolen
- /dev/null "Datenpapierkorb"

4.7.3.4 Links

Im Prinzip ist ein Link nur ein weiterer Name für die Datei. Bei den Links wird zwischen dem harten Link (hard link) und dem symbolischen (weichen) Link (soft link) unterschieden.

(a) Hard Link

Darunter ist tatsächlich nichts anderes zu verstehen, als ein zweiter Name für ein und dieselbe Datei. Wenn eine Datei zwei Namen hat, so kann nicht zwischen Original und Link unterschieden werden. Es ist möglich, eine der beiden Dateien zu löschen und trotzdem ist die andere noch vorhanden. Physikalisch handelt es sich aber immer um die selbe Datei, d.h., sie nimmt nur einmal den Speicherplatz auf der Platte in Anspruch. Physikalisch werden die Daten daher erst gelöscht, wenn der letzte Hard-Link für die Datei gelöscht wird.

Diese Tatsache erklärt aber auch die Einschränkung, der ein Hardlink unterworfen ist:

Der Link muss sich auf der selben Partition befinden, auf der sich die Originaldatei befindet. Das liegt in der Natur des Links, weil er ja nur ein weiterer Name für die Datei ist, der im Inhaltsverzeichnis der Platte bzw. der Partition gespeichert ist.
Hardlinks werden mit dem Befehl `ln` angelegt.

Ein **Hard-Link** ordnet einer Inode einen weiteren logischen Dateinamen zu. Der **Referenzzähler** (**Link-Counter**) **in der Inode** gibt an, wieviele logische Verweise - d.h. unterschiedliche Dateinamen - auf die betreffenden Datei existieren!

> **Merke:**
>
> Jeder Hard-Link hat dieselben Berechtigungen auf die Datei, da alle Links auf dieselbe Inode verweisen!!

Beispiel

Auf die existierende Datei `max` wird ein harter Link namens `moritz` angelegt. Wird nun `moritz` durch z.B. den Editor `VI` editiert, so finden sich die Änderungen auch in `max` wieder.

```
mustermann@linux:~/links > ls -il
insgesamt 2
111099 -rw-r--r--   2 mustermann   users      46 Jan 14 13:41 max
111099 -rw-r--r--   2 mustermann   users      46 Jan 14 13:41 moritz
```

In der dritten Spalte die Anzahl der harten Links (Link Counter) zu sehen, die auf die Datei zeigen. Löscht man nun einen der beiden Links, so bleibt die Datei trotzdem bestehen. Erst wenn alle Links auf die Datei gelöscht sind, die Zahl also auf Null steht, wird die Datei auch wirklich gelöscht.

(b) Soft Links

Im Gegensatz zu Hardlinks sind symbolische Links wesentlich flexibler. Hier handelt es sich tatsächlich um Verweise auf eine bestehende Datei, d.h., der symbolische Link ist in Wahrheit eine Datei mit eigener Inode, **zu der im Verzeichniskatalog der Pfad auf die Originaldatei gespeichert ist.** Damit das System diese Datei nicht als Textdatei mißinterpretiert wird sie als SymLink gekennzeichnet, als Dateityp steht ein `l` statt eines Bindestrichs.
Durch diese Tatsache kann man beim SymLink immer zwischen Orginal und Link unterscheiden, im Gegensatz zum Hardlink, bei dem das nicht möglich ist. Symbolische Links erlauben es auch, Verweise über die Partitionsgrenzen hinweg zu erstellen. Allerdings ist nicht sichergestellt, daß der Link auch auf etwas zeigt, was existiert. Wird z.B. ein SymLink auf eine Datei angelegt und anschließend die Originaldatei gelöscht, so zeigt der Link ins Leere.

Als weiterer Vorteil von SymLinks ist noch die Tatsache zu nennen, daß sie auch auf Verzeichnisse zeigen können. Damit ist es also etwa möglich, ein Verzeichnis, das nicht mehr auf die Partition passt, in der es erwartet wird, in einer anderen Partition abzuspeichern und in der ursprünglichen Partition einfach einen SymLink auf die neue Position zu setzen.

Merke:

Symbolische Links haben **nicht** dieselben Berechtigungen wie die Originaldatei!

Dateityp: `l`

Beispiel:

```
lrwxrwxrwx   1   mueller  users   2   Okt 16 2001   slink -> file.orig
                                                        |          |
                                                    Linkname    Original
```

Kommando ln

Das Kommando `ln` legt einen harten oder einen symbolischen Link an.

```
ln [OPTIONEN] DATEINAME NEUERLINK
```

Symbolische Links werden wie Hardlinks mit dem Befehl **`ln`** angelegt. Zur Unterscheidung wird dort einfach der Parameter **`-s`** (für symbolischen Link) benutzt.

Optionen

`-s` Legt einen symbolischen Link an

Einen harten oder symbolischen Link können Sie durch den Befehl **`rm`** entfernen.

4.7.3.5 Feststehende Programmverbindungen (named pipes oder FIFOs)

In seltenen Fällen kann es nötig sein, feststehende Verbindungsleitungen zwischen verschiedenen Programmen zu nutzen. Dazu können named pipes benutzt werden. Es handelt sich um einfache FIFO Buffer (First In First Out), in die geschrieben werden kann und aus denen gelesen werden kann. Damit können z.B. auch Prozesse, die auf verschiedenen Terminals laufen via Pipe verbunden werden.

Dateityp: p

4.8 Zugriffsberechtigungen auf Dateien und Verzeichnisse

Als grundlegende Zugriffsrechte können für die drei Benutzerklassen **user** (Eigentümer), **group** (Gruppe) und **others** (alle Personen, die nicht Eigentümer der Datei und nicht Mitglied in der Gruppe des Eigentümers sind) die **Rechte**

r	**Leserecht**
w	**Schreibrecht**
x	**Ausführungsrecht**

vergeben werden.

Diese **Zugriffsrechte** (r,w,x) **unterscheiden sich für Dateien und Verzeichnisse** wie in der Tabelle dargestellt:

Zugriffsrecht	Datei	Verzeichnis
read (r)	lesen	Dateinamen listen
write (w)	Inhalt verändern	Inhalt des Verzeichnisses verändern (= Erzeugen oder Löschen von Dateien)
execute (x)	Inhalt als Programm ausführen	Verzeichnis durchsuchen oder den Befehl c**d** anwenden, um in das Verzeichnis zu wechseln

4.8.1 Anzeige der Zugriffsberechtigungen

Die Zugriffsberechtigungen können mit dem Kommando **`ls`** und der Option **`-l`** angezeigt werden.

```
ls –l DATEI
```

Mit diesem Befehl wird in der ersten Spalte des Inhaltsverzeichnisses eine 10-stellige Anzeige des Dateityps und der Zugriffsrechte erzeugt.

- Das **erste Zeichen** gibt dabei den **Typ der Datei** an:
  - `-` normale Datei
  - `d` Verzeichnis
  - `b` blockorientierte Gerätedatei
  - `c` zeichenorientierte Gerätedatei
  - `l` Link-Datei
  - `p` Pipe oder FIFO special file
- Die **Stellen 2-4** zeigen die **Zugriffsrechte des Besitzers** der Datei.
- Die **Stellen 5-7** zeigen die **Zugriffsrechte der Besitzergruppe**.
- Die **Stellen 8-10** zeigen die **Zugriffsrechte aller übrigen Benutzer**.

Es existieren also drei Gruppen zu je drei Spalten, die die Rechte

`-`**`rwx`**`rwxrwx` **des Eigentümers**
`-rwx`**`rwx`**`rwx` **der Gruppe und**
`-rwxrwx`**`rwx`** **der anderen**

bezeichnen.

Beispiel:

```
-rw-r--r--   1   mustermann   users   24   Okt 30 2001   datei.txt
Besitzer (user)
      Gruppe (group)
            alle anderen Benutzer (others)
```

Hinweis:

Die angegebenen Zugrifssrechte gelten nicht für den Superuser. Dieser hat immer alle Rechte auf jede Datei!

4.8.2 Kommandos zum Ändern/Setzen von Zugriffsrechten

4.8.2.1 chmod

Der Befehl chmod erlaubt das Ändern der Rechte für eine Datei oder Verzeichnis. Es kann nur vom Besitzer der Datei/Verzeichnis oder vom Super-User ausgeführt werden.

```
chmod [OPTIONEN] RECHTE DATEILISTE
```

Mit der **Option –R** können die Zugriffsrechte **rekursiv**, d.h. für das angegebene Verzeichnis und alle untergeordneten Verzeichnisse/Dateien geändert werden.

Die Zugriffsrechte können auf zwei verschiedene Arten gesetzt werden:

- Symbolischer oder relativer Modus und
- numerischer oder absoluter Modus

(a) Symbolischer oder relativer Modus

Symbolisch lassen sich die Rechte setzen durch eine Kombination aus der betreffenden Rechtegruppe

Symbol	Rechtegruppe
u	Eigentümer (user)
g	Gruppe (group)
o	Andere (others)
a	Alle (all)

und den entsprechenden Rechten:

Symbol	Recht
r	Leserecht
w	Schreibrecht
x	Ausführungsrecht
s	s-Bit setzen
t	t-Bit setzen

Kombiniert werden die Rechtegruppen mit den entsprechenden Rechten mit Hilfe von Operatoren.

Operator	Funktion
+	Recht(e) hinzufügen
-	Recht(e) entfernen
=	Genau diese Recht(e) setzen, d.haufgeführte Rechte setzen, nicht aufgeführte Rechte wegnehmen

In Form einer Zuweisung wird beim Setzen der Rechte auf der linken Seite des Operators die Zielgruppe genannt. Auf der rechten Seite stehen die Rechte.

```
chmod  <Zielgruppe> <Operator> <Rechte>  Datei/Verzeichnis
             u          +          r
             g          -          w
             o          =          x

             a
```

Beispiele:

```
user@linux> ls -l test.txt
-rw-r--r--   1  user    users    108 Apr  7 12:10   test.txt

user@linux> chmod a+w test.txt
user@linux> ls -l test.txt
-rw-rw-rw-   1  user    users    108 Apr  7 12:10   test.txt

user@linux> chmod g-rw,u+xs test.txt
user@linux> ls -l test.txt
-rws---rw-   1  user    users    108 Apr  7 12:10   test.txt
```

(b) Numerischer oder absoluter Modus

Im numerischen Modus werden die Rechte mit einem Wert dem Kommando chmod mitgegeben.

Abbildung: Nummerischer Modus von chmod

Die Rechte werden in diesem Modus bitweise dargestellt, für Eigentümer, Gruppe und Andere.

Recht	**Bitdarstellung**	**Wert**
Read	0000 0100	4
Write	0000 0010	2
E**x**ecute	0000 0001	1

Jede Rechtegruppe wird durch einen numerischen Wert repräsentiert, die Sonderrechte werden durch einen eigenen Wert dargestellt. Somit werden dem Kommando chmod maximal vier Werte übergeben (Abbildung). Fehlen Werte, werden diese von links her (!) als Null (keine Rechte) angenommen, d.h.

```
chmod 1 <datei>
```

ist gleichbedeutend mit

```
chmod 0001 <datei>
```

Für das Kommando wird also ein Wet benötigt, der die Rechtevergabe widerspiegelt. Dazu nun eine kleine Übersicht mit den 3 Nutzern, den 3 Rechten und dem dazugehörigen Wert:

	Besitzer (user)	Gruppe (group)	Andere (others)
Lesen	4	4	4
Schreiben	2	2	2
Ausführen	1	1	1

Durch die Addition der Werte für den jeweiligen Nutzer ergibt sich dann entsprechend der Wert, der dem Kommando zum Setzen der Rechte mitgegeben wird.

Beispiel:

Eine Datei test.txt soll für alle lesbar und für den Besitzer "schreibbar" sein.

Besitzer: 4 (lesen) + 2 (schreiben) = **6**
Gruppe: 4 (lesen) = **4**
Andere: 4 (lesen) = **4**

Im Beispiel ergibt die Kombination der Rechte den Wert **644**, also

```
user@linux> chmod 644 test.txt
```

Für die Sonderrechte gelten folgende Bitdarstellungen:

Recht	**Bitdarstellung**	**Wert**
s-Flag des Eigentümers, **Set UserID Bit (SUID)**	0000 0100	4
s-flag der Gruppe, **Set GroupID Bit (SGID)**	0000 0010	2
t-Flag, **Sticky Bit**	0000 0001	1

Set UserID Bit (SUID)

Dieses Recht gilt ausschließlich für ausführbare Dateien. Hat eine ausführbare Datei dieses Recht gesetzt, so erscheint in der Darstellung durch das ls -l Kommando statt dem x beim Eigentümerrecht ein s. Jeder User, der dieses Programm ausführt, tut dies unter der effektiven UserID des Users, dem die Datei gehört.
So hat z.B. das Programm /usr/bin/passwd die Aufgabe, daß auch normale User damit ihr eigenes Passwort ändern können. Dieses Passwort wird aber gespeichert in einer Datei, die nur von root beschrieben werden darf. Das Programm /usr/bin/passwd hat als Eigentümer root und hat das Substitute UserID Bit gesetzt. Jeder User, der dieses Programm ausführt tut dies also unter der effektiven UserID von root und hat daher die Rechte von root während der Ausführung. Das ermöglicht es dem Programm, das veränderte Passwort zu speichern.

```
-rwsr-xr-x  1  root  shadow  27920  Jul 29 2000
```

Set GroupID Bit (SGID)

Dieses Recht gilt einerseits für ausführbare Dateien und andererseits für Verzeichnisse. Hat ein ausführbares Programm dieses Recht gesetzt, so gilt der gleiche Mechanismus, wie beim Substitute UserID Bit, nur diesmal eben die Gruppenmitgliedschaft betreffend. Ein User, der dieses Programm ausführt, tut dies als Gruppenmitglied der Gruppe, der das Programm gehört, statt seiner eigentlichen Gruppenkennung. Er hat also die Rechte eines Gruppenmitglieds dieser Gruppe, auch wenn er selbst nicht Mitglied dieser Gruppe ist. Statt dem x beim Gruppenrecht stellt das ls -l Kommando hier ein s dar.
Hat ein Verzeichnis dieses Recht gesetzt, dann liegt der Fall etwas anders. Legt ein User, der Schreibrecht auf ein Verzeichnis hat, in diesem Verzeichnis eine Datei an, so erhält diese Datei normalerweise die Gruppenmitgliedschaft der primären Gruppe des Users, der sie eben angelegt hat. Das führt schnell zum Chaos, wenn z.B. mehrere User zusammen an einem Projekt arbeiten. Alle diese User sind Gruppenmitglieder einer bestimmten Gruppe, aber sie haben diese Gruppe nicht als primäre Gruppe. Das heißt, es kann sein, daß die einzelnen User die Dateien der jeweils anderen Projektmitarbeiter nicht lesen können. Wenn dieses Verzeichnis aber der gemeinsamen Gruppe gehört und eben das Substitute GroupID Bit darauf gesetzt ist, dann werden Dateien in diesem Verzeichnis grundsätzlich unter der GruppenID abgespeichert, der das Verzeichnis gehört. Mit diesem Mechanismus sind Arbeitsverzeichnisse für bestimmte Projektgruppen realisierbar.

Sticky Bit

Das Sticky Bit hat nur eine Bedeutung für Verzeichnisse. Oben wurde schon das Problem erwähnt, daß ein User, der Schreibrecht auf ein Verzeichnis hat, innerhalb dieses Verzeichnisses alle Dateien löschen kann, auch wenn sie ihm nicht gehören und er keinerlei Rechte auf diese Dateien besitzt. Das wird durch das Sticky Bit verhindert.

Ein Verzeichnis in dem jeder User Schreibrecht haben muß, wie etwa das /tmp-Verzeichnis, sollte unbedingt dieses Bit gesetzt haben. Ansonsten kann ein böswilliger User dort die Dateien anderer User löschen.
Das gesetzte Sticky-Bit wird vom ls -l Kommando durch ein t statt des x in der Kategorie "others" dargestellt.

Beispiel:

```
drwxrwxr-t  2  mustermann  users  4096   Okt 16 2001   test.txt
```

Um diese speziellen Rechte auch numerisch darzustellen, wird vor den oben genannten "normalen" Rechten noch eine Oktalziffer angefügt, so daß sich letztlich das folgende Bild ergibt:

Sonderrechte			**user**			**group**			**others**		
SUID	SGID	Sticky	r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1	4	2	1

Ein Recht von 4755 bedeutet also, daß das Set UserID Bit gesetzt ist (4), der Eigentümer Lese-, Schreib- und Ausführungsrechte (4+2+1=7) hat während sowohl Gruppenmitglieder, als auch der Rest der Welt nur Lese- und Ausführungsrecht (4+1=5) besitzen.

```
user@linux> ls -l hello
-rw-r--r--   1  user    users    108 Apr  7 12:10   hello

user@linux> chmod 4755 hello
-rws---rw-   1  user    users    108 Apr  7 12:10   hello
```

4.8.3 Standardeinstellung der Dateirechte

Das Kommando **umask** (*User's creation MASK*) vereinbart eine Standardeinstellung der Dateirechte für alle Dateien, die nach Aufruf von **umask** erzeugt werden. Auf bestehende Dateien hat das Kommando keinen Einfluss.

umask [MASKE]

Die Eingabe von `umask` ohne Parameter gibt die aktuell eingestellte Maske wieder. Die Maske selber ist eine dreistellige oktale Zahl. Die Bedeutung der Zahlen ist identisch mit derer für die Rechtevergabe. Allerdings gibt `umask` nicht an, welche Rechte gegeben werden, sondern welche Rechte entzogen werden. Oft wird die `umask` in der Datei `/etc/profile` auf den Wert `022` voreingestellt. Jeder Benutzer kann jedoch für sich explizit seine persönliche Maske in der Datei `.profile` in seinem Homeverzeichnis setzten.

Um den Wert der `umask` zu ermitteln, zieht man am einfachsten von der Maximalmaske den oktalen Wert der gewünschten Rechte ab (siehe oben Numerischer oder absoluter Modus) und erhält so den Wert der `umask`. Dabei ist zu beachten, das sich der Wert der Maximalmaske für Verzeichnisse und Dateien unterscheidet.

	Verzeichnisse	Dateien
Maximalmaske	777	666
Rechte	755	644
umask	022	022

Umgekehrt lässt sich anhand des oktalen Wertes der `umask` auch die Rechte, die bei Erzeugung von Verzeichnissen und Dateien vergeben werden, ermitteln. Die bei der Erzeugung gesetzten Rechte entstehen nun, indem von der Maximalmaske der durch `umask` vorgegebene Wert subtrahiert wird:

	Verzeichnisse	Dateien
Maximalmaske	777	666
umask	022	022
Rechte	755	644

Wird die Maske vom Promptzeichen aus mit `umask` festgelegt, dann erlischt diese wieder beim beim Ausloggen.

4.8.4 Ändern der Besitzergruppe einer Datei

Das Kommando **chgrp** ändert für eine Datei die zugehörige Gruppe.

```
chgrp [OPTIONEN] GRUPPE DATEILISTE
```

Die Änderung der Gruppe ist für den Besitzer und für den Superuser erlaubt. Der Besitzer darf allerdings seiner Datei nur eine Gruppe zuordnen, in der er selber Mitglied ist.

Optionen

`-c`	Informationen über alle geänderten Dateien werden angezeigt
`-v`	Informationen über alle Aktionen werden ausgegeben
`-f`	Fehlermeldungen werden nicht ausgegeben
`-R`	Änderungen werden rekursiv den Verzeichnisbaum herunter durchgeführt

Beispiel:

```
user@linux> chgrp users test.txt
```

ändert die Gruppe der Datei `test.txt` auf **users**.

4.8.5 Ändern des Besitzers

Mit dem Befehl `chown` ist es möglich der Datei einen neuen Besitzer und gleichzeitig eine neue Gruppe zu geben.

```
chown [OPTIONEN] [BENUTZER][:GRUPPE] DATEILISTE
```

Allerdings ist es nur dem Superuser gestattet, den Eigentümer einer Datei zu ändern. Die Änderung der Gruppe ist für den Besitzer und für den Superuser erlaubt. Der Besitzer darf allerdings seiner Datei nur eine Gruppe zuordnen, in der er selber Mitglied ist.

Optionen

`-c`	Informationen über alle geänderten Dateien werden angezeigt
`-v`	Informationen über alle Aktionen werden ausgegeben
`-f`	Fehlermeldungen werden nicht ausgegeben
`-R`	Änderungen werden rekursiv den Verzeichnisbaum herunter durchgeführt

Beispiele:

`user@linux>` **`chown moritz test.txt`**
ändert den Besitzer der Datei `test.txt` auf **moritz**.

`user@linux>` **`chown -R moritz:users /home/max`**
ändert den Besitzer auf **moritz** und die Gruppe auf **users** für das Verzeichnis `/home/max` und seinem Inhalt.

`user@linux>` **`chown :users test*`**
ändert die Gruppe für alle Dateien, die mit "test" beginnen auf **users**.

4.8.6 Arbeiten mit Dateien

4.8.6.1 touch

Das Kommando `touch` ändert die Zeit des letzten Zugriffs und der letzten Änderung auf die aktuelle Zeit. Existiert die Datei nicht, so wird eine neue Dateien der Größe 0 Byte erstellt.

```
touch [OPTIONEN] [DATEILISTE]
```

4.8.6.2 cat

`cat` (`concatenate file`) wird vor allem als Tool zum Anzeigen und Zusammenfügen von Dateien verwendet.

```
cat [OPTIONEN] [DATEILISTE]
```

Optionen

-b	Nummeriert alle nichtleeren Zeilen durch
-e	Gleich mit -vE
-n	Nummeriert alle Zeilen durch (-b hat Vorrang)
-E	Fügt am Ende jeder Zeile ein $ ein
-s	Faßt aufeinanderfolgende Leerzeilen zu einer Leerzeile zusammen
-v	Zeigt alle nichtdruckbaren Zeichen durch Metazeichen an
-T	Zeigt Tabulatoren als ^I an
-A	Zeigt alle nichtdruckbaren Zeichen, Tabulatoren als ^I und das Zeilenende als $ an (vgl. -vET)

Beispiel

Fügt die Dateien t1 und t2 zusammen und schreibt sie in die Datei t3.

```
cat t1 t2 > t3
```

4.8.6.3 cp

Um Kopien von einer Datei oder einem Verzeichnis zu erstellen wird das Kommando cp verwendet.

```
cp [OPTIONEN] QUELLDATEI ZIELDATEI
cp [OPTIONEN] QUELLDATEILISTE ZIELVERZEICHNIS
```

Wird ein Dateiname für die Quelle angegeben, so kopiert cp die Datei in eine zweite Datei (ZIELDATEI). Ist der letzte Parameter ein Verzeichnis, dann wird die Datei in das Verzeichnis kopiert. Listen von Quelldateien können nur in ein Verzeichnis kopiert werden. Allerdings kann cp in der Normaleinstellung keine Verzeichnisse kopieren.

Optionen

-a	Kopiert die Dateien unter Beibehaltung von Struktur und Attributen, das gleiche wie -dpR
-b	Erzeugt von jeder überschriebenen Datei eine Sicherheitskopie
-f	Überschreiben von vorhandenen Zieldateien
-i	Nachfragen vorm Überschreiben von vorhandenen Zieldateien
-p	Überträgt Besitzer, Gruppe, Rechte und Zeitmarken an die neue Datei

-P	Kopieren der Dateien inklusiver ihrer Verzeichnisstruktur
-r	Kopiert rekursiv Dateien aus Unterverzeichnissen mit
-R	Recursiv
-u	Kopiert nur Dateien, die jünger sind als die Zieldateien
-v	Zeigt die Namen der kopierten Dateien an

4.8.6.4 mv

Das Kommando `mv` (move) verschiebt ein oder mehrere Dateie(en) bzw. Verzeichnis(se) oder benennt sie um.

```
mv [OPTIONEN] ALTERDATEINAME NEUERDATEINAME
mv [OPTIONEN] QUELLDATEILISTE ZIELVERZEICHNIS
```

Werden zwei Dateinamen als Parameter vergeben, dann wird die Datei umbenannt. Ist der letzte Parameter ein Verzeichnis, dann werden die angegebenen Dateien in dieses Verzeichnis verschoben.

Optionen

-b	Erzeugt von jeder überschriebene Datei eine Sicherheitskopie
-f	Überschreiben von vorhandenen Zieldateien
-i	Nachfragen vorm Überschreiben von vorhandenen Zieldateien
-u	Verschiebt nur Dateien, die jünger sind als die Zieldateien

Beispiel:

Die folgende Kommandosequenz benennt die Datei `test.txt` in die Datei `TEST.txt` um:

```
mv test.txt TEST.txt
```

Mit dem folgenden Kommando werden alle Dateien aus dem aktuellen Verzeichnis in das Verzeichnis `/home/moritz` verschoben und eventuell vorhandene Zieldateien überschrieben:

```
mv -f * /home/moritz
```

4.8.6.5 rm

Der Befehl `rm` löscht die angegebenen Dateien. In seiner Standardeinstellung löscht er keine Verzeichnisse.

```
rm [OPTIONEN] DATEILISTE
```

Um eine Datei zu löschen, benötigt man das Schreibrecht (`w`) auf das Verzeichnis, aber nicht auf die Datei. Sollte ein Datei nicht das Schreibrecht besitzen, so erbittet `rm` nur eine Bestätigung des Löschbefehls, wenn kein `-f` oder wenn ein `-i` gesetzt ist.

Optionen

`-d`	Löscht ein Verzeichnis durch Entfernen des Hardlinks. Volle Verzeichnisse werden auch gelöscht. Da die enthaltenen Dateien nicht mehr referenziert werden, ist es ratsam ein `fsck` danach auszuführen. (Nur Superuser)
`-f`	Löscht alle Dateien ohne explizites Nachfragen. Überlagert die Option `-i`
`-i`	Fragt vor jedem Löschen einer Datei um Bestätigung
`-r`	Löscht Verzeichnisse und deren Inhalt rekursiv

Beispiel:

Der folgende Befehl löscht alle Dateien, die mit `.txt` enden.

```
rm *.txt
```

Dieser Befehl löscht das Verzeichnis `/home/moritz/tmp` ohne Nachfrage.

```
rm -rf /home/moritz/tmp
```

4.8.6.6 more

Der Befehl `more` gehört zur Gruppe der Befehle, mit denen man sich den Inhalt von Dateien anschauen kann (*pager*). Im Gegensatz zu `cat` zeigt er die Daten aber seitenweise an.

```
more [OPTIONEN] DATEILISTE
```

Befehle

`<LEERTASTE>`	Eine Seite nach unten scrollen
`<f>`	Eine Seite nach unten scrollen
`<b>`	Eine Seite nach oben scrollen
`<RETURN>`	Eine Zeile nach unten scrollen
`<s>`	Eine Zeile nach unten scrollen

</>MUSTER	Durchsuchen des Textes nach dem regulären Ausdruck MUSTER
<n>	Weitersuchen nach unten
<q>	Beenden

4.8.6.7 less

Der Befehl `less` ist die Weiterentwicklung von `more`.

```
less [OPTIONEN] DATEILISTE
```

Neben den Möglichkeiten von `more` bietet `less` die Möglichkeit mit den Richtungstasten zu scrollen und mit Lesezeichen, Zeilennummern und prozentualen Textpositionen zu arbeiten. Auch wird `less` nicht am Ende der Datei beendet.

Zusätzliche Befehle

<j>	Eine Zeile nach unten scrollen
<k>	Eine Zeile nach oben scrollen
<g>	An den Anfang der Datei scrollen
<G>	Ans Ende der Datei scrollen
</>MUSTER	Durchsuchen des Textes nach dem regulären Ausdruck MUSTER nach unten
<?>MUSTER	Durchsuchen des Textes nach dem regulären Ausdruck MUSTER nach oben
<n>	Weitersuchen nach unten
<N>	Weitersuchen nach oben

4.8.6.8 ls

Eine Reihe von Befehlen ermöglicht es, sich den Inhalt eines Verzeichnisses anzusehen. Der am häufigsten benutzte Befehl ist `ls` (LiSt).

```
ls [OPTIONEN] [DATEINAME]
```

Für DATEINAME können Namen von Dateien oder Verzeichnissen verwendet werden. Dabei kann, um eine Menge von Dateinamen zu bilden, Metazeichen (Joker) verwendet werden (s.u. Exkurs Jokerzeichen). Die bekanntesten Joker sind dabei das Fragezeichen ?, das für ein einzelnes beliebiges Zeichen steht, und der Asterisk *, der für eine beliebige Anzahl beliebiger Zeichen steht.

Optionen

-a	Anzeige aller Dateien, auch derjenigen, die mit . beginnen.

-A Anzeige aller Dateien außer . und ..

-c Sortiert zusammen mit -t Dateien nach dem Datum der letzten Änderung der Inode (Verwaltungsinformationen). Standardsortierfolge ist nach ASCII-Zeichen. Die Anzeige des entsprechenden Datums erfolgt zusammen mit -l. (ls -ctl)

-d Verzeichnisse in der Liste der Argumente werden wie andere Dateien behandelt. Unterdrückung der Durchsuchung des Inhalts von Verzeichnissen.

-l Anzeige ausführlicher Dateiinformationen. Viele Schalter haben nur im Zusammenhang mit -l eine Bedeutung, da sie nicht automatisch die Information anzeigen sondern nur vorbereiten (z. B. -e oder -k).

-r Ausgabe in umgekehrter Sortierreihenfolge

-t Sortiert Dateien nach dem Datum der letzten Änderung. Standardsortierfolge ist nach ASCII-Zeichen. Die Anzeige des Datums erfolgt zusammen mit -l.

-x Spaltenweise Ausgabe der Dateinamen, im Gegensatz zu -C jedoch waagerecht geordnet.

-m Ausgabe der Dateinamen als durch Komma getrennte Liste.

-F Kennzeichnung diverser Dateitypen durch Anhängen von Sonderzeichen: / für Verzeichnisse, * für ausführbare Dateien, @ für Links, | für FIFOs und = für Sockets.

-R Rekursive Anzeige, d. h. es werden nicht nur Verzeichnisse nach ihrem Inhalt durchsucht, sondern auch darin enthaltene (Unter-) Verzeichnisse. Nicht zusammen mit -d anwendbar.

Beispiele:

```
user@linux > ls -lF
-rw-r--r--    1   ukp22       wio22 6243  Nov 20 2001 sortbliste
-rwxr-xr-x    1   ukp22       wio22 6243  Nov 20 2001 testprogramm*
drwxr-xr-t    2   ukp22       wio21 4096  Okt 16 2001 testverzeichnis/
-rw-r--r--    1   ukp22       wio22   24  Okt 30 2001 text1
(1)(2)        (3)    (4)      (5)    (6)     (7)         (8)
```

Erläuterung des Listings:

(1) erstes Zeichen kennzeichnet Dateityp
(2) Zugriffsrechte der drei Rechtegruppen
(3) Link-Counter
(4) Besitzer der Datei
(5) Gruppe, die Rechte auf die Datei besitzt
(6) Speicherplatzbelegung in Byte
(7) Datum/Uhrzeit der Erstellung/letzten Modifikation
(8) Dateiname

```
user@linux > ls -il
352234      -rw-r--r-- ..... sortbliste
352232      -rwxr-xr-x ..... testprogramm
352432      drwxr-xr-t ..... testverzeichnis
  |
Inode
Number
```

```
user@linux > ls -sl
0     lrwxrwxrwx ..... slink -> s1
8     -rw-r--r-- ..... sortbliste
4     -rwxr-xr-x ..... testprogramm
4     drwxr-xr-t ..... testverzeichnis
|
Belegte
Datenblöcke
```

Exkurs Jokerzeichen

Um sich nur Dateien mit einem bestimmten Namensmuster anzuschauen, verwendet man wie bereits erwähnt so genannte Jokerzeichen (Wildcards). Nachfolgende Tabelle zeigt eine Übersicht der wichtigsten Jokerzeichen, die von der Shell interpretiert werden.

Jokerzeichen	Bedeutung
?	genau ein beliebiges Zeichen
*	null oder beliebig viele Zeichen
[abc]	genau eines der angegebenen Zeichen

[a-f]	genau ein Zeichen aus dem angegebenen Bereich
[!abc] oder [^abc]	keines der angegebenen Zeichen
~	Alias für das Heimatverzeichnis
abc{1,2,3}	liefert abc1, abc2 und abc3

Tabelle: Übersicht der Jokerzeichen

Beispiele:

```
user@linux> ls /boot/*.map
/boot/System.map

user@linux> ls -d /?[abc]*
/sbin   /var

user@linux> ls /?[abc]*
/sbin:
SuSEconfig   fsck.minix     ldconfig   rmmod
YaST         genksyms   lilo       route ...

/var:
X11R6   cron    lock   named     spool   texfonts
adm     games   log    openwin   squid   tmp
...
```

Aufgaben:

Was machen nachfolgende Kommandoaufrufe:

- user@linux> **rm main.[!ch]**
- user@linux> **rm main.{x,y,z}**
- user@linux> **rm main.?**

4.8.6.9 find

`find` durchsucht das aktuelle oder ein angegebenes Verzeichnis rekursiv nach Dateien und zeigt diese mit vollem Pfadnamen an.

```
find [PFAD] [SUCHKRITERIEN]
```

Wenn keine Suchkriterien angegeben wurden, werden alle Dateien angezeigt. Die Ausgabe lässt sich (u.a.) mit den folgenden Kriterien auf bestimmte Dateien beschränken:

Kriterien	Aktion
`-name MUSTER`	Findet Dateien, auf deren Dateiname das `MUSTER` zutrifft.
`-type dateityp`	Findet Dateien vom gegebenen Typ: b = Blockgerät, c = Zeichengerät, d = Verzeichnis, l = Link oder f = normale Datei
`-user BENUTZER`	Findet die Dateien des angegebenen Benutzers
`-group GRUPPE`	Findet Dateien die zu einer bestimmten `GRUPPE` (GID) gehören
`-perm NNN`	Findet Dateien, auf die Rechtemaske `NNN` zutrifft.
`-inum INODENUMMER`	Findet Dateien mit der Inode `N`
`-ok BEFEHL`	Ausführen eines `BEFEHL` für jede gefundene Datei mit vorheriger Sicherheitsabfrage
`-exec BEFEHL`	Ausführen eines `BEFEHL` für jede gefundene Datei
`-print`	Zeigt die gefundenen Dateien mit ihrem Pfad an. (Standard)

Tabelle: Suchkriterien des `find`-Befehls

> **Hinweis:**
>
> Werden **im gesuchten Dateinamen Wild Cards** verwendet, so muss der **Name in Anführungsstriche** gesetzt werden.

`find -name "test*" –print` sucht alle Dateien, die mit `test` anfangen

Es können zwei Optionen mit Blank getrennt angegeben werden. In diesem Fall werden beide Optionen als logische **UND**-Verknüpfung betrachtet!

`find / -size +10k -user moritz -print` zeigt alle Dateien die größer als 10 kB sind **UND** dem Benutzer **moritz** gehören.

Es können zwei Optionen mit **-o** getrennt in der Klammerung \(\) angegeben werden. In diesem Fall werden beide Optionen als logische **ODER**-Verknüpfung betrachtet!

find / \(-perm 644 -o **-perm 664 \)**	ab dem Root-Directory nach Dateien mit den Zugriffsrechten 644 **ODER** 664.

Die Negation eines Kriteriums wird durch das ! erreicht.

find / -name !'*~' -print	findet alle Dateien, die *nicht* mit einer Tilde ~ enden.

Beispiel:

```
user@linux > find / -user 406 -ok rm {} \;
```

sucht im gesamten Dateisystem nach Dateien des Benutzers mit der uid 406. Gefundene Dateien werden bei positiver Beantwortung einer Sicherheitsabfrage gelöscht.

Kapitel 5

Kommandoverarbeitung

5.1 Kommandos

Unter Linux existieren 4 Arten von Kommandos:

- Aliasse
- Funktionen (nicht (T)csh)
- Builtins
- Ausführbare Programme

5.1.1 Aliasse

Ein Alias ist einfach nur ein anderer Name für ein Kommando oder eine Kommandofolge.

Die allgemeine Syntax für den Befehl `alias` lautet:

```
alias [NAME[=KOMMANDO]]
```

Dabei gibt das Kommando `alias` unter Angabe des Aliasnamen das dazugehörige Kommando aus. Der Aufruf ohne Parameter gibt eine Liste aller definierten Aliase mit ihren Kommandosequenzen aus.

Wird ein Alias mit dem Namen eines vorhandenen Kommandos definiert, so wäre das originale Kommando nur noch durch Angabe seines vollständigen Zugriffspfades erreichbar:

Beispiel:

```
user@linux> alias date="echo 'date ist weg'"
user@linux> date
date ist weg
user@linux> /bin/date
Son Okt 13 20:38:41 CEST 2002
user@linux> unalias date
user@linux> date
Son Okt 13 20:39:03 CEST 2002
```

Mit **`unalias NAME`** werden definierte Aliasse wieder gelöscht

5.1.2 Funktionen

Funktionen gruppieren Kommandos als eine separate Routine.

5.1.3 Builtins

Builtins sind Kommandos, die direkt in der Shell implementiert sind, d.h. diese Kommandos existieren nicht als Programme oder Shellskripte, z.B. das Kommando `cd`.

5.1.4 Ausführbare Programme

Ausführbare **Programme** liegen irgendwo auf der Festplatte. Zum Start eines Programms erzeugt die Shell einen neuen Prozess, alle anderen Kommandos werden innerhalb des Prozesses der Shell selbst ausgeführt.

5.2 Verknüpfen von Kommandos

Linux ermöglicht es, mehrere Kommandos in einer Zeile einzugeben. Dabei werden die einzelnen Kommandos durch das Zeichen ";" getrennt.

Beispiel:

```
user@linux> date; echo "Verzeichnisinhalt:"; ls
Die Nov 19 23:55:00 MEST 2002
Verzeichnisinhalt:
Changelog    einwahlzeiten         filelist.mandrake  Sources
```

```
cron.src     features.txt          hijack
bin  dead.letter  filelist.grundsystem  Makefile
```

5.3 Bedingte Ausführung von Kommandos

Die Zeichen **&&** und **||** verknüpfen zwei Befehle miteinander, wobei der zweite Befehl in Abhängigkeit vom Erfolg des ersten Befehls ausgeführt wird.

Beispiel:

```
user@linux> less config.txt || less test.txt
```

bewirkt, dass nur wenn der erste Befehl keinen Erfolg hat, weil z.B. die Datei nicht existiert, der zweite Befehl ausgeführt wird.

```
user@linux> less config.txt && cp config.txt /backup
```

bewirkt, dass nur wenn der erste Befehle erfolgreich war, der zweite Befehl ausgeführt wird.

5.4 Substituierung von Kommandos

Oft ist es sinnvoll die Ausgabe eines Kommandos in einem anderen Kommando zu verwerten. Das Einschließen eines Kommandos in `$(...)` oder in `` `...` `` bewirkt, daß die Ausgabe des Kommandos verwendet wird. Mit ***Kommandosubstitution*** ist also gemeint, ein Kommando innerhalb eines Kommandos auszuführen.

Beispiel:

```
user@linux> touch `date +"%d-%m-%y"`-config.txt

user@linux> touch $(date +"%d-%m-%y")-config.txt
```

In beiden Beispielen wird eine Datei erstellt, die jeweils das aktuelle Datum vorangestellt hat.

5.5 Umleitung der Eingabe und Ausgabe

Jedes Programm erhält automatisch beim Start drei offene "`Datenkanäle"': die Standardeingabe (**STDIN**), die Standardausgabe (**STDOUT**) und die Standardfehlerausgabe (**STDERR**).

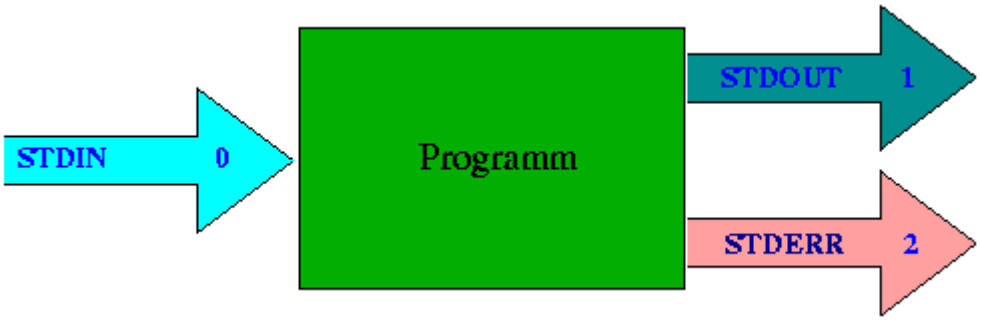


Abbildung: die drei Standardkanäle

Standard-Eingabe (STDIN)
Der Kanal, von dem das Programm seine Eingabe erwartet. Dieser Kanal besitzt die Kanalnummer 0 und ist normalerweise mit der Tastatur verbunden.

Standard-Ausgabe (STDOUT)
Der Kanal, auf den das Programm seine Ausgaben schreibt. Dieser Kanal besitzt die Nummer 1 und ist normalerweise mit dem Monitor verbunden.

Standard-Fehlerausgabe (STDERR)
Der Kanal, auf den das Programm seine Fehlerausgaben schreibt. Dieser Kanal besitzt die Kanalummer 2 und ist wie STDOUT normalerweise mit dem Monitor verbunden.

Standardmäßig kommen die Eingaben von der Tastatur und die Ausgabe des Kommandos und seiner Fehlermeldungen landen auf dem Bildschirm. Diese Ausgaben können unter Linux aber in Dateien oder Geräte umgeleitet werden, bevor eine Kommandozeile ausgeführt wird. Auf diese Weise können Dateien im Dateisystem zum Lesen bzw. Schreiben für das Kommando geöffnet werden, die nach dessen Beendigung automatisch wieder geschlossen werden. Beispielsweise kann so die Ausgabe des *ls*-Kommandos zur weiteren Bearbeitung in eine Datei geschrieben werden (s. unten 5.5.2 Ausgabe)

5.5.1 Eingabe

Mit dem Zeichen "<" liest das Programm seine Eingabe aus der Datei statt von der Standardeingabe (Tastatur).
Beispiel:

```
user@linux > cat < meinText
```

5.5.2 Ausgabe

Mit dem Zeichen "**>**" leitet das Programm seine Standard Ausgabe in die Datei um, statt sie auf den Bildschirm zu schreiben. Existiert die Datei schon, so wird sie überschrieben, existiert sie noch nicht, so wird sie neu angelegt.

Beispiel:

```
user@linux > ls > verzeichnis
```

Dabei wird jedesmal eine neue Datei namens `Verzeichnis` erstellt und der Inhalt der alten Datei gelöscht.

Soll die Ausgabe aber an eine bestehende Datei angehängt werden, so verwendet man "**>>**", existiert sie noch nicht, so wird sie neu angelegt.

Beispiel:

```
user@linux > echo $UID $USER >> logdatei
```

5.5.3 Fehlerausgabe

Manchmal ist es auch sinnvoll oder gewünscht, den Fehlerausgabekanal umzuleiten. Das ist über die Nennung seiner Kanalnummer vor dem Umleitungssymbol, also mit "**2>**" möglich. So würde die Anweisung

```
Programm 2> Fehlerprotokoll.txt
```

die Fehlerausgabe des Programms in die Datei `Fehlerprotokoll`.txt umleiten. Selbstverständlich ist das auch mit 2 Umleitungssymbolen (anhängend) möglich.

Beispiel:

```
user@linux > touch /etc/passwd 2> fehler
```

5.5.4 Fehlerausgabe und Programmausgaben

Und in einigen Fällen sollen beide Ausgabekanäle (STDOUT und STDERR) in eine Datei umgeleitet werden. Dazu kann man beide Kanäle zu einem zusammenfassen, der dann umgeleitet wird. Das geschieht mit der kryptischen Anweisung "**2>&1**". Um wirklich beide Kanäle in eine Datei umzuleiten muß die Bündelung nach der eigentlichen Umleitung vorgenommen werden:

```
Programm > Datei 2>&1
```

Diese Anweisung bündelt die Kanäle 2 und 1 (STDERR und STDOUT) und leitet beide in die Datei um.

Um beide Ausgaben (Fehler und Programm) gemeinsam in eine Datei zu leiten, wird auch "**&>**" und "**>&**" verwendet:

```
Programm &> Datei
Programm >& Datei
```

Praxisbeispiel:

Die E/A Umleitung ist z.B. bei der Protokollierung der Onlinezeiten mit Hilfe der beiden Skripte *ip-up* und *ip-down nützlich*. Diese beiden Skripte werden von Linux beim Auf- bzw. Abbau einer (z.B. ISDN) Verbindung aufgerufen. Durch eine Erweiterung dieser Dateien um paar echo-Zeilen lässt sich die Online-Zeit auf einem Terminal und in einer Datei mitschreiben. *ip-up* ist dabei wie folgt zu erweitern:

```
echo -n "$(date +"%d.%m.%Y %H:%M:%S") - " ..>>/etc/ppp/online-time
echo -n "$(date +"%d.%m.%Y %H:%M:%S") - "   > /dev/tty6 2>&1
```

Für *ip-down* gilt analog dazu:

```
echo "$(date +"%d.%m.%Y %H:%M:%S")"  >>/etc/ppp/online-time
echo "$(date +"%d.%m.%Y %H:%M:%S")"   > /dev/tty6 2>&1
```

Bei jeder Einwahl wird nun ein String in das Terminal */dev/tty6* und in die Datei */etc/ppp/online-time* geschrieben, der die Form 13.12.1999 18:37:36 -" hat. Das Newline-Zeichen wurde absichtlich weggelassen, so dass beim Auflegen die Offline-Zeit hinten drangehängt werden kann, ein Eintrag sieht dann beispielsweise so aus:

```
13.12.1999 18:37:36 - 13.12.1999 19:25:30
```

Die so protokollierten Zeiten lassen sich danach mit einem einfachen Skript auswerten.

5.5.5 Pipes

Wenn wir jetzt zwei Programme haben, von denen das zweite die Ausgabe des ersten weiterverarbeiten sollte, so könnten wir jetzt die Ausgabe des ersten Programms in eine Datei umleiten und dann das zweite Programm aufrufen und seine Eingabe aus der Datei lesen lassen:

```
Programm1 > Datei
Programm2 < Datei
```

Das funktioniert auch ohne den Umweg über die Datei, indem wir zwei Programme direkt miteinander verbinden. Genauer gesagt wird die Standard-Ausgabe des ersten Programms direkt mit der Standard-Eingabe des zweiten verbunden.

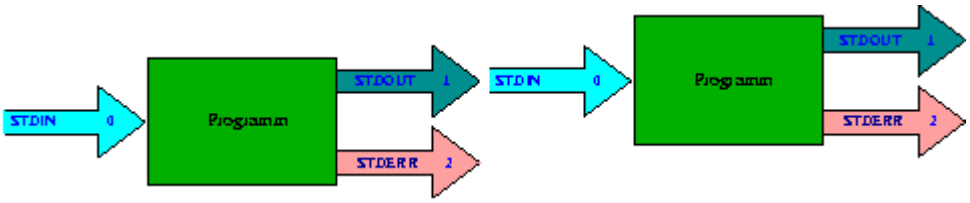


Abbildung: Funktionsweise einer Pipe

Realisiert wird dies durch ein zwischengeschaltetes Pipe-Zeichen "**|**"

Beispiel:

```
user@linux > ls -Al | less
user@linux > less /etc/passwd | grep root
```

Das läßt sich beliebig fortsetzen, es ist also möglich, auch mehrere Programme hintereinander zu hängen.

```
user@linux > less /var/log/messages | less | grep isdn
```

5.5.6 tee

Das Kommando `tee` sendet gleichzeitig die Ausgabe an eine Datei und an die Standardausgabe, die der Bildschirm oder eine weitere Pipeline sein kann.

```
tee [OPTIONEN] DATEILISTE
```

Optionen

`-a`	Die Daten werden an eine vorhandene Datei angehängt

Beispiele

```
user@linux > ls -al | tee myDir
```

Diese Kommandosequenz sendet den Inhalt des Verzeichnisses in die Datei `myDir` und auf den Bildschirm.

```
user@linux > cat /var/log/messages | tee -a ausgabe.txt | less
```

Bei dieser Sequenz wird der Inhalt der Datei `messages` an die Datei `ausgabe.txt` angefügt und gleichzeitig mit `less` angezeigt.

Kapitel 6

Prozessverarbeitung

6.1 Prozesse unter Linux

Alle Aktionen des Betriebssystems Linux - z.B. Ausgabe von Daten, Kopieren von Daten - erfolgen in Form von Prozessen.

> **Merke**:
> Ein Prozeß ist ein in der Ausführung befindliches Programm mit seiner Ausführungsumgebung.

Jeder Prozess arbeitet in einem separaten, virtuellen Adressraum, so kann kein Prozess einen anderen wissentlich (z. B. durch Viren) oder unwissentlich stören. Den einzelnen Prozesses wird nacheinander der Zugriff auf die CPU erlaubt. Wie lange und in welcher Reihenfolge dies geschieht, wird durch den Prozess Scheduler bestimmt. Wenn ein Programm "hängt" kann es nicht, wie bei manch primitiven Betriebsystemen, den ganzen Rechner lahm legen.

Ein Prozess kann weitere Prozesse starten. In diesem Fall werden der **startende Prozess** als **Vaterprozess** und die **gestarteten Prozesse** als **Sohnprozesse** dieses Startprozesses bezeichnet. Ein Sohnprozess kann wiederum weitere Prozesse starten. Aus dieser Vater-Sohn-Prozessbeziehung ist ersichtlich, dass Prozesse hierarchisch strukturiert sind. Jeder Prozess - mit Ausnahme des Ursprungsprozesses **init**[5] - besitzt also einen Vaterprozess, wobei ein Vaterprozess durchaus mehrere Sohnprozesse besitzen kann. Ein Vaterprozess bleibt normalerweise mindestens so lange aktiv, wie seine Sohnprozesse aktiv sind.
Alle Prozesse werden über die zentrale Prozesstabelle verwaltet. Die Prozesse stehen in einer hierarchischen Ordnung, da jeder Prozeß eine eindeutige Nummer (Prozessnummer, **PID = process identification number**) und die Nummer des Elternprozesses mit sich führt. Sohnprozesse haben zusätzlich auch die Nummer ihres Vaterprozesses - die **PPID** (= **parent process identification number**) - gespeichert.

Das Kommando `pstree` gibt einen Baum von allen Prozessen aus. Dadurch wird deutlich, welcher Prozess von welchem anderen Prozess gestartet wurde.

```
pstree [OPTIONEN] [PID]
```

Optionen

`-a`	Zeigt die Kommandozeilenargumente an
`-c`	Verhindert die kompakte Darstellung von identischen Zweigen

[5] `init` ist der Vater aller Prozesse. Er hat die Prozessnummer 1 und alle weiteren Prozesse sind Kinder von `init`

`-l`	Zeigt lange Zeilen an, sonst werden die Zeilen für die Bildschirmbreite umgebrochen
`-n`	Sortiert die Prozesse nach PID und nicht nach Name
`-p`	Zeigt die PIDs zu den Prozessen an
`-u`	Zeigt an, wenn die UID von Kinds- und Elternprozeß sich unterscheidet

Ohne weitere Parameter gibt `pstree` den Baum ab dem Prozess `init` aus.

Beispielhafte Ausgabe des Befehls `pstree`:

```
[root@linux /root]# pstree
init-+-adsl-connect---pppd---pppoe
     |-authdaemond.pla---5*[authdaemond.pla]
     |-2*[couriertcpd]
     |-crond
     |-ddclient
     |-gpm
     |-httpd---8*[httpd]
     |-klogd
     |-6*[kreiserfsd]
     |-2*[logger]
     |-lpd
     |-6*[mingetty]
     |-named---named---3*[named]
     |-nmbd
     |-proftpd
     |-qmail-send-+-qmail-clean
     |            |-qmail-lspawn
     |            |-qmail-rspawn
     |            `-splogger
     |-safe_mysqld---mysqld---mysqld---mysqld
     |-sh---tcpserver
     |-slapd
     |-smbd
     |-snmpd
     |-splogger
     |-squid-+-5*[ncsa_auth]
     |       |-pinger
     |       `-unlinkd
     |-sshd---sshd---bash---pstree
     |-syslogd
     |-tcpserver
     `-xfs
[root@linux /root]#
```

Die Verwaltungsinformationen von Prozessen sind im Verzeichnis `/proc` (= virtuelles Dateisystem) abgebildet. In diesem Verzeichnis existiert für jeden Prozess ein eigenes Unterverzeichnis, das als Name die PID des betreffenden Prozesses erhält. Das Verzeichnis `/proc` beansprucht jedoch keinen Speicherplatz auf der Festplatte, es ist nur ein Abbild betriebssysteminterner Daten.

6.1.1 Ausführen von Prozessen im Vorder- und Hintergrund

Normalerweise kann der Benutzer erst einen neuen Befehl starten, wenn das aktuelle Kommando beendet ist, da auch der Befehlsprompt nicht angezeigt wird. Die Shell wartet dann bis der Prozess beendet ist und kehrt erst danach zur Eingabeaufforderung zurück. Der Ablauf solcher Programme wird als ***Vordergrundprozess*** bezeichnet.

Damit die Eingabeaufforderung sofort wieder zur Verfügung steht, kann ein Prozess im Hintergrund gestartet werden, indem der Kommandozeile einfach ein Ampersand (**&**) angehängt wird.
Dadurch kehrt die Shell sofort wieder zur Eingabeaufforderung zurück, der gestartete Prozess läuft jetzt im Hintergrund (***Hintergrundprozess***) und meldet nur kurz seine PID und die Jobnummer. Dies ist sinnvoll bei längeren Programmabläufen, wenn keine Nutzereingaben mehr notwendig sind.

Beispiel:

```
root@linux > find / -type d -print &
[3] 12304
```

Die Zahl in eckigen Klammern gibt an, wie viel Jobs nun im Hintergrund laufen. Die zweite Zahl ist die PID.

Zu beachten ist, dass Jobs, die im Hintergrund arbeiten, weiterhin auf den Bildschirm schreiben, und dabei die Arbeit im Vordergrund erheblich stören können. Man kann aber die Möglichkeit der Datenumleitung mittels > nutzen, um die Ausgabe in eine Datei schreiben. Fehlermeldungen lassen sich mit 2> umleiten (s.o.)

```
root@linux > find / -type d -print 2> /dev/null > ↵
/tmp/ergebnis.txt &
```

6.1.2 Wechsel zwischen Vorder- und Hintergrundausführung

Die Shell bietet die Möglichkeiten, Prozesse, die im Vordergrund gestartet wurden (also ohne &) aus dem Vordergrund in den Hintergrund zu schieben und umgekehrt.

Hierzu ist zunächst erforderlich, dass der Vordergrundprozess angehalten wird. Mit der Tastenkombination **Strg-z** wird der Vordergrundprozess gestoppt. Dadurch wird dem Job jegliche Rechenzeit entzogen und die Shell kehrt zur Eingabeaufforderung zurück. Das Kommando **bg** setzt einen unterbrochenen Prozess im Hintergrund fort. Ohne Parameter versetzt **bg** den zuletzt unterbrochenen Prozess in den Hintergrund und der Job bekommt wieder Rechenzeit. Anderenfalls muss der Prozessname oder die Jobnummer angegeben werden. Diese wird beim Stoppen des Prozesses in eckigen Klammern angezeigt oder kann mit dem Kommando **jobs**[6] abgefragt werden.

```
bg [PROZESS]
```

[6] Zeigt eine Liste aller Jobs, Vordergrund- Hintergrund und gestoppte Jobs samt ihren JobIDs

Merke:
Soll ein Hintergrundprozess auch nach Abmeldung des Benutzers weiterlaufen, so muss das Kommando mit

nohup *kommando* **&**

gestartet werden

Um einen Prozess wieder in den Vordergrund zu schieben wird das Kommando **fg** verwendet. **fg** setzt einen Prozess im Vordergrund fort. Als Parameter wird die Jobnummer angeben. Beim alleinigen Aufruf von **fg** wird der zuletzt gestoppte Prozess wieder aufgenommen. Die Shell kehrt *nicht* mehr zur Eingabeaufforderung zurück.

fg [PROZESS]

6.1.3 Überwachung von Prozessen

Laufende Prozesse können überwacht werden mit den Programmen **ps** und **top**.

6.1.3.1 ps

Das Programm **ps** bietet einen Schnappschuss der gerade laufenden Prozesse. Es zeigt verschiedene Daten dieser Prozesse an, das wichtigste ist dabei die ProzessID (PID) des jeweiligen Prozesses. Ohne Parameter aufgerufen zeigt ps nur die eigenen Prozesse.

ps [OPTIONEN]

Die Optionen bei ps werden ohne führenden Bindestrich geschrieben, weil der Befehl keine Parameter kennt.

Optionen

a	Zeigt alle Prozesse auf einem Terminal an
e	Zeigt die dazugehörigen Umgebungsvariablen
f	Zeigt die Ausgabe als Baum (Eltern- und Kindsprozesse)
l	Ausführliche Ansicht
u	Zeigt Benutzer und Startzeit an
x	Wählt Prozesse ohne kontrollierende ttys
w	breite Ausgabe

Häufige Kombinationen von Parametern sind z.B.

ps auxw

Zeigt eine Liste aller (a) Prozesse des Systems in breiter Ausgabe (w) inklusive der Angabe der User (u) denen die Prozesse gehören, es werden auch daemon-Prozesse ohne eigenes Terminal (x) angezeigt.

```
ps fax
```

Zeigt eine Liste aller (a) Prozesse des Systems wieder mit den daemon-Prozessen (x) an. Die Prozesse werden als Baumstruktur (f) angezeigt, die klar macht, welche Prozesse von welchen Eltern-Prozessen abstammen.

6.1.3.2 top

Im Gegensatz zu **ps** zeigt der Befehl **top** eine ständig aktualisierte Liste der Prozesse mit der höchsten CPU-Auslastung. Er erlaubt daneben auch gleichzeitig Statistiken über den Speicher und die Swap-Datei. Systemlaufzeit, CPU-Status und Prozeßgröße sind weitere Angaben, die das Tool liefert.

Top ist ein interaktives Programm, das auch Modifikationen der Prozesse zulässt. Mit der Taste **<h>** oder **<?>** wird ein Hilfe-Bildschirm angezeigt. **top** läuft solange, bis es mit **<q>** (quit) beendet wird.

Beispiel für eine Ausgabe des Befehls **top**:

```
  8:38pm  up 29 min,  3 users,  load average: 0.11, 0.08, 0.09
59 processes: 51 sleeping, 2 running, 6 zombie, 0 stopped
CPU states:  0.7% user,  0.7% system,  0.0% nice, 98.4% idle
Mem:    62844K av,   61308K used,    1536K free,   46324K shrd,   15880K
buff
Swap:  128516K av,     380K used,  128136K free                   17252K
cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU %MEM   TIME COMMAND
  724 moritz     9   0  1060 1060   876 R       0  0.9  1.6   0:00 top
  207 root      10   0 11160  10M  2008 R       0  0.3 17.7   0:16 X
  569 moritz     2   0  4272 4272  3100 S       0  0.1  6.7   0:01 kvt
    1 root       0   0   200  200   172 S       0  0.0  0.3   0:05 init
    2 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kflushd
    3 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kupdate
    4 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kpiod
    5 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kswapd
   63 bin        0   0   396  396   320 S       0  0.0  0.6   0:00 portmap
   75 root       0   0   556  556   468 S       0  0.0  0.8   0:00 syslogd
   79 root       0   0   784  784   328 S       0  0.0  1.2   0:00 klogd
  108 root       0   0   500  484   396 S       0  0.0  0.7   0:00 rpc.nfsd
  115 root       0   0  1536 1516  1400 S       0  0.0  2.4   0:00 httpd
  116 wwwrun     0   0  1056  912   800 S       0  0.0  1.4   0:00 httpd
  122 at         0   0   540  540   460 S       0  0.0  0.8   0:00 atd
```

6.1.4 Ansprechen und Beenden von Prozessen

Unter Linux können laufende Prozesse angesprochen werden, genauer gesagt können ihnen Signale geschickt werden. Wie die jeweiligen Prozesse auf Signale reagieren hängt davon ab, was der Programmierer des jeweiligen Programms eingestellt hat.
Die wichtigste Aufgabe von Signalen ist es, Prozesse zu beenden. Das Programm zum Versenden von Signalen heißt daher folgerichtig **`kill`**.

Es gibt aber auch viele andere Aufgaben für Signale, so kann etwa ein Prozess durch ein Signal dazu aufgefordert werden, seine Konfigurationsdatei neu einzulesen. Damit kann ein laufender Prozess verändert werden, ohne ihn neu starten zu müssen.

```
kill [-Signal] PID [PID ...]
```

Das Programm kill erwartet eine oder mehrere ProzessIDs als Parameter, die die Prozesse beschreiben, denen das Signal geschickt werden soll. Ein Parameter, mit führendem Bindestrich wird als Signal interpretiert, alle anderen als PIDs. Signale können entweder in ihrer numerischen Form oder über ihren Namen angegeben werden - also z.B. entweder -9 oder -KILL.

> **Hinweis**:
> Nur der Superuser oder der Besitzer des Prozesses dürfen Signale senden.

Wenn kein spezielles Signal angegeben wird, so schickt kill den Prozessen das Signal Nr. 15 (SIGTERM), das einen Prozess auffordert, sich selbst zu beenden. Man spricht in diesem Fall auch von Terminieren. Es gibt ein Signal, das immer "tödlich" ist, es hat die Nummer 9 (SIGKILL).

Nr.	Langname	Kurzname	Bedeutung
1	SIGHUP	HUP	Hangup: Reinitialisierung des Prozesses
2	SIGINT	INT	Interrupt (wie <STRG>+<C>)
3	SIGQUIT	QUIT	Quit: Beenden
9	SIGKILL	KILL	Sofortiges Beenden des Prozesses (wird nicht ignoriert)
15	SIGTERM	TERM	Sofortiges Beenden des Prozesses (kann ignoriert werden)

Tabelle: Signale und ihre Bedeutung

Eine Liste aller unterstützten Signale bekommt man mit **`kill -l`**

Beispiel:

```
root@linux > kill -9 %netscape
```
fordert den Job "netscape'' auf, sich bedingungslos zu beenden

```
root@linux > kill –HUP $(cat /var/run/httpd.pid)
```
führt zum Neustart des Apache-Webservers.

Genau wie der Befehl **kill** leitet der Befehl **killall** Signale an Prozesse weiter. Allerdings wird bei diesem Befehl nicht die PID sondern der Name des laufenden Programm angegeben. Alle Prozesse, die diesem Namen zugeordnet sind, werden dann beendet.

```
killall [OPTIONS] ID
```

Die sonstige Arbeitsweise ist identisch mit `kill`.

6.1.5 Prozesspriorität

6.1.5.1 nice

Wie oft und schnell ein Programm CPU-Zeit bekommt hängt u. a. von seiner Prozesspriorität ab. Dies Prozesspriorität kann mit dem Befehl **nice** verringert werden. Nur der Superuser ist in der Lage einem Prozess eine höhere Priorität zuzuordnen.

```
nice [OPTION] KOMMANDO
```

Das Kommando ist ein Befehl oder eine Kette von Befehlen, die mit der angegebenen Priorität ausgeführt werden können. Im Normalfall ist die Prozesspriorität bei Verwendung von **nice** auf den Wert 10 festgelegt. Mit dem Schalter `-n ZAHL` kann eine Priorität zwischen -20 und 19 angegeben werden. Dabei bedeutet eine kleinere Zahl eine höhere Priorität und eine große Zahl eine niedrige Priorität. Nur der Superuser ist in der Lage einen negativen Wert, und damit eine höhere Priorität als normal, einzustellen.

Beispiel:

```
root@linux > nice -n 19 find / -name urmel* -print > liste.txt
```

Dieser langwierige Prozeß bekommt eine sehr niedrige Ausführungspriorität zugewiesen. Er wird praktisch nur ausgeführt, wenn das System genug Zeit hat.

6.1.5.2 renice

Der Befehl **renice** erlaubt eine Änderung der Prozesspriorität im Gegensatz zu **nice** für laufende Programme.

```
renice PRIORITÄT PROZESS
```

Dabei kann der Zielprozess durch Angabe der PID bestimmt werden. Mit den Schaltern `-u` und `-g` kann sich auf alle Prozesse eines Benutzers oder einer Gruppe bezogen werden und mit `-p` weitere PIDs angegeben werden.

Beispiel:

```
root@linux > renice +5 4711 -u apache moritz -p 42
```

Das Kommando setzt die Prozesspriorität der Prozesse 42 und 4711, sowie alle Prozesse von den Benutzern *apache* und *moritz*. Normale Benutzer können nur auf die Priorität Ihrer eigenen Prozesse einwirken und dabei die Priorität, wie schon aus `nice` bekannt, verringern.

6.1.6 Der Init-Daemon

Das Letzte, was der Kernel macht, ist den Init-Daemon (`/sbin/initd`) zu starten. Dieser bleibt so lange aktiv, bis das System heruntergefahren wird. Damit ist **`init`** wie bereits schon erwähnt der erste laufende Prozess. Alle weiteren Prozesse werden entweder vom Init-Daemon direkt oder indirekt durch Subprozesse von **`init`** gestartet. Dieser Daemon ist also für die Erstellung der Prozesse verantwortlich, die der Rest des Systems nutzt. Als Beispiel ist hier die Login-Shell zu nennen. Daneben sorgt er dafür, das bestimmte Prozesse nach ihrem Ende wieder neu gestartet werden. Er sorgt z.B. dafür, dass nach dem Logout die Konsole neu gestartet wird, um eine erneutes Einloggen zu ermöglichen. Bei einem Shutdown ist `init` der letzte noch laufende Prozess, der sich um das korrekte Beenden aller anderen Prozesse kümmert.
Der Init-Daemon und seine Aktionen werden durch die Konfigurationsdatei `/etc/inittab` gesteuert.

6.1.6.1 /etc/inittab

Diese Datei hat im Wesentliche vier durch Doppelpunkte getrennte Spalten der Form:

`Kennung:Runlevel:Aktion:Befehlszeile`

Beispielsweiser Auszug einer /etc/inittab eines RedHat -Systemes:

```
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Author:       Miquel van Smoorenburg,
<miquels@drinkel.nl.mugnet.org>
#               Modified for RHS Linux by Marc Ewing and Donnie Barnes
#
# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have
networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:
# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
l0:0:wait:/etc/rc.d/rc 0
```

```
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
# Things to run in every runlevel.
ud::once:/sbin/update
# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
# When our UPS tells us power has failed, assume we have a few minutes
# of power left.  Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting
Down"
# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown
Cancelled"
# Run gettys in standard runlevels
1:12345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
# Run xdm in runlevel 5
x:5:respawn:/usr/bin/X11/xdm -nodaemon
```

(a) Kennung

Die Kennung besteht aus zwei Zeichen und muss in der Datei eindeutig sein, um eine Identifizierung des Eintrags zu ermöglichen.

(b) Runlevel

Linux bietet verschiedene Betriebszustände an, sogenannte. Im Wesentlichen geht es dabei um die Frage, welche Dienste zur Verfügung gestellt werden und welche nicht.
Ein Runlevel definiert also einen Satz von Prozessen, der als Teil der Bootsequenz gestartet wird. Das kann von einem minimalen Satz von Prozessen zur Systemadministration bis zu einer Vielzahl von Prozessen für alle eingerichteten Geräte reichen.

Grundsätzlich stehen die Runlevel 0 bis 6 und S (oder s) zur Verfügung. Dabei haben die Runlevel 0, 6 und S eine festgelegte Bedeutung, alle anderen stehen frei zur Verfügung. Verschiedene Linux-Distributionen belegen diese übriggebliebenen Runlevel mit verschiedenen Bedeutungen, es existiert kein Standard. Meist werden mindestens drei Runlevel definiert, einer für Multiuser ohne Netz, einer für Multiuser mit Netz und einer für Multiuser mit Netz und graphischem Login (xdm). Die drei festgelegten Runlevel sind:

Runlevel	Bedeutung
0	System herunterfahren ohne anschließenden Neustart. (**halt**)

6	System herunterfahren und neu starten. (**reboot**)
S oder **s**, bei RedHat auch **1**	System im **Single User Mode** betreiben Der Single User Modus ist ein Modus, der in der Systemverwaltung immer dann benötigt wird, wenn Arbeiten durchgeführt werden sollen, bei denen Aktivitäten anderer Benutzer stören könnten. Zum Beispiel die Reparatur eines Dateisystems oder die Spiegelung einer Platte.

Der folgende Abschnitt definiert die Skripte, die für den jeweiligen Runlevel ausgeführt werden sollen. Sie stehen im Verzeichnis `/etc/rc.d` bzw. bei S.u.S.E. im Verzeichnis `/sbin/init.d`.

```
l0:0:wait:/sbin/init.d/rc 0
l1:1:wait:/sbin/init.d/rc 1
l2:2:wait:/sbin/init.d/rc 2
l3:3:wait:/sbin/init.d/rc 3
l6:6:wait:/sbin/init.d/rc 6
```

Das **rc**-Skript wird mit einem Parameter, der den Runlevel angibt, aufgerufen. Dabei führt es Skripte aus dem Verzeichnis **rc*n*.d** aus, wobei *n* für die Nummer des Runlevels steht.
Daneben kann **rc** auch Kernel-Module aufrufen. Dies sind Teile des Kernels, die nicht fest einkompiliert wurden. Sie können bei Bedarf nachgeladen und wieder entladen werden. Meistens handelt es sich hierbei um Gerätetreiber, die nicht oft benötig werden.

Der Befehl **init** ermöglich das Wechseln des Runlevels.

```
init [OPTIONEN] [RUNLEVEL]
```

Beim Aufruf des Befehl sendet **init** ein Stopsignal an alle Prozesse, die nicht für den neuen Runlevel definiert sind. Danach werden die Prozesse "gekilled'' und die Prozesse für den Runlevel, die noch nicht laufen, werden gestartet.

Mit Hilfe des Kommandos **runlevel** wird der Runlevel angezeigt, indem man sich gerade befindet.

(c) Aktion

Mit der Aktion wird angegeben, wie mit dem Eintrag verfahren werden soll. So kann z.B. festgelegt werden, dass der Prozess automatisch neu gestartet wird, wenn er beendet wurde. Hier gibt es die unter Anderem folgende Möglichkeiten:

Eintrag	**Aktion**
`boot`	Startet, wenn die `/etc/inittab` zum ersten Mal gelesen wird

bootwait	Startet nach den Einträgen für boot
ctrlaltdel	Startet nach der Tasteneingabe <Strg>+<Alt>+<Entf>
initdefault	Definiert den Standard-Runlevel
once	Startet beim Wechsel eines Runlevels
ondemand	Hält Prozesse in bestimmten Runlevels am Laufen
powerfail	Startet, wenn ein Stromausfallsignal empfangen wurde
sysinit	Startet, bevor die Einträge boot und bootwait ausgeführt werden.
respawn	Nachdem der hier gestartete Prozess zu einem Ende gekommen ist, wird sofort ein neuer gestartet.
wait	Startet den Prozeß einmal beim Erreichen des Runlevels und init wartet auf sein Ende

Eine Ausnahme stellen hier die Zeilen mit den Einträgen initdefault und sysinit dar. Der initdefault-Eintrag gibt an, welcher Runlevel anzustreben ist, wenn beim Systemstart nichts anderes verlangt wurde. Der beim sysinit-Eintrag angegebene Befehl wird unabhängig vom Runlevel beim allerersten Start von /sbin/init. ausgeführt, nicht aber bei evtl. folgenden Runlevelwechseln.

(d) Befehlszeile

Im letzten Feld wird das Linux-Kommando oder das Programm aufgeführt, daß in der Situation gestartet werden soll.

6.1.7 Ablaufsteuerung von Prozessen mit at, crontab und batch

6.1.7.1 at

Wenn ein Job einmal zu einer bestimmten Zeit gestartet werden soll, hilft das Kommando at.

```
at [OPTIONEN] ZEIT
```

Optionen

-b	Startet den Job, wenn das System nicht ausgelastet ist (*batch*)
-d	Löscht einen Job (*delete*)
-f DATEI	Liest den Job aus der angegebenen Datei (*file*)
-l	Zeigt alle Jobs des Benutzers an (*list*)
-m	Sendet nach dem Abschluß des Jobs eine eMail (*mail*)

Die Benutzung des `at`-Kommandos ist ganz einfach:

- Geben Sie `at` zusammen mit dem Ausführungszeitpunkt an.
- Geben Sie die Kommandos ein, die ausgeführt werden sollen.
- Drücken Sie <Strg>+<d> um den Job abzuschicken.

Beispiel:

```
user@linux:~ > at +1 minutes
warning: commands will be executed using /bin/sh
at> who > ~/who.txt
at> <EOT>
job 9 at 2001-02-10 18:25
user@linux:~ >
```

Bei der Zeitangabe stehen Ihnen mehrere Möglichkeiten zur Verfügung:

- Ein Zeitpunkt als Tag, Stunde und Minute.
- Ein Zeitpunkt von jetzt an gemessen.
- Der Wochentag als Zahl oder als Name.
- Die Verwendung von Worten wie `midnight`, `noon`, `teatime`, `today`, `tommorrow` und `now`.

Beispiele:

```
at 15:30
at 18.03.01 15:30
at 03/18/01 15:30
at 031801 15:30
at +3 hours
at +5 minutes
at now
```

Der Befehl `atrm` löscht einen wartenden `at`-Job.

`atrm JOBNUMMERN`

Die Funktion des Befehls ist identisch zu `at -d JOBNUMMER`

6.1.7.2 crontab

Das Kommando `at` erlaubt nur die zeitlich gesteuerte einmalige Ausführung eines Jobs. Allerdings ist es manchmal angebracht ein Job regelmäßig auszuführen. Dazu gehört z.B. die Datensicherung oder das Abgleichen von zwei Datenbanken. Für diese Aufgaben steht der Befehl `crontab` zur Verfügung.

`crontab [OPTIONEN] [DATEI]`

Optionen

`-e`	Erstellt oder bearbeitet die `crontab`-Datei des Benutzers
`-l`	Zeigt die `crontab`-Datei des Benutzers an
`-r`	Löscht die `crontab`-Datei des Benutzers
`-u BENUTZER`	Legt fest, mit welcher `crontab`-Datei gearbeitet werden soll (nur Superuser)

Ein Eintrag innerhalb einer `crontab` besteht aus einer Zeile mit sechs Feldern ein. Diese sechs Felder sind von links nach rechts: `Minute`, `Stunde`, `Tag des Monats`, `Monat`, `Wochentag` und `Kommando`.

Feld	Werte
`Minute`	0 - 59
`Stunde`	0 - 23
`Tag des Monats`	0 - 31
`Monat`	0 - 11 oder Namen
`Wochentag`	0 - 7 oder Namen (0 oder 7 ist Sonntag)

Tabelle: Erlaubte Werte für einen Crontab-Job

Ein Asterisk "`*`" in einem Feld repräsentiert jeden möglichen Wert für das Feld.

Das `crontab`-Kommando erstellt eine `crontab`-Datei an und speichert eine Kopie unter dem Benutzernamen in dem Verzeichnis `/usr/lib/crontab` ab.

> **Hinweis**:
> Das Anlegen der Datei bzw. das Editieren der Datei erfolgt standardmäßig mit dem Editor VI.

Beispiele:

Um täglich festzuhalten, wer um 10 Uhr morgens eingeloggt ist, kann folgender Cronjob angelegt werden.

```
00 10 * * * who >> /var/log/benutzerliste.log
```

Zeitbereich können mit einem Bindestrich `-' angegeben werden. Bei einer Liste von Werten werden die Einzelwerte durch Kommata voneinander getrennt. Ein Job, der jeden Dienstag und Donnerstag um 12 Uhr ausgeführt werden soll, kann dann so angegeben werden.

```
00 12 * * 2,4 myjob
```

6.1.7.1 batch

Das Kommando `batch` bewirkt das Gleiche wie `at` mit dem Schalter `-b` und legt einen Job an, wenn das System weniger belastet ist, d.h. bei diesem Befehl bestimmt nicht der Benutzer den Startzeitpunkt des Programmes, sondern das System.

`batch [OPTIONEN] [ZEIT]`

Im Gegensatz zu `at` hat der Job eine sehr niedrige Priorität. Der Befehl untersucht die virtuelle Datei `/proc/loadavg`. Wenn die durchschnittliche Belastung (*average load*) unter 1.5 sinkt, dann wird der Job ausgeführt.

Kapitel 7

Drucken unter Linux

7.1 Die Druckerwarteschlange

Traditionell wird eine Datei gedruckt, indem ihr Inhalt einfach über die Druckerschnittstelle (in der Regel ein Parallelport) ausgegeben wird; unter MS-DOS war das auch genau die Vorgehensweise. Linux ist aber ein echtes Multi-User-Multi-Tasking-Betriebssystem, d. h. es muss damit rechnen, dass mehrere Benutzer aus verschiedenen Programmen heraus gleichzeitig drucken wollen. Würden nun alle Prozesse einfach auf die parallele Schnittstelle schreiben, würde ein Chaos entstehen. Daher haben unter Linux normale Anwender keine Schreibrechte für die Parallelports (`/dev/lp*`), sondern müssen ihre Druckaufträge (= Jobs) mit dem Hilfsprogramm `lpr` in eine Warteschlange (=Queue) einstellen. Einstellen in die Warteschlange bedeutet standardmäßig, daß eine Kopie der zu druckenden Datei in ein Verzeichnis, den sogenannten Spool-Bereich, geschrieben wird. Dabei wird für jeden verfügbaren Drucker mindestens ein eigener Spool-Bereich angelegt, der ein Unterverzeichnis von `/var/spool/` (oft `/var/spool/lpd/` oder bei CUPS `/var/spool/cups/`) ist.
Ein Hintergrund-Prozeß, der Druck-Dämon `lpd` überprüft in regelmäßigen Abständen diese Spool-Bereiche und sorgt für den Druck von Druckaufträgen aus der Warteschlange auf dem angegebenen Drucker. Die Abarbeitung der Druckwarteschlange erfolgt dabei nach dem FIFO-Prinzip.

Der Syntax des Kommandos lpr zum Einstellen eines neuen Druckauftrages ist denkbar einfach:

```
lpr [OPTIONEN] DATEI(EN)
```

Optionen

`-m`	sendet eine Mail nach Abarbeitung des Druckauftrages in die Mailbox des Benutzers
`-h`	unterdrückt den Ausdruck der Header-Seite
`-#anzahl`	druckt den Druckauftrag mit *anzahl* Kopien
`-P`	Ausgabe auf einem anderem Gerät
`-r`	Löscht die zu druckende Datei, wenn der Druckauftrag abgeschlossen ist.

Prinzipiell kann `lpr` auf zwei Arten aufgerufen werden:

```
user@linux:~ > lpr datei.ps
```

Mit diesem Kommando wird die Datei `datei.ps` auf dem Standard-Drucker ausgedruckt.

Welcher Drucker als Default verwendet wird, entnimmt das Kommando der Umgebungsvariablen `$PRINTER`. Ist diese nicht gesetzt, dient der erste Warteschlangeneintrag in der `/etc/printcap` als Standard.

Sind mehrere Drucker eingerichtet so gibt die Datei /etc/printcap darüber Überblick, welche Druckerwarteschlangen eingerichtet sind. Der Name der Warteschlange kann dann mit der Option `-P[NAME]` (z.B. `lpr -Plp2`) übergeben werden. Wird diese Option nicht verwendet, geht Linux vom Standarddrucker aus.

```
user@linux:~ > cat /etc/printcap | grep ^lp
lp:\
lp2:\
```

In diesem Beispiel heißen die beiden vorhandenen Warteschlangen also `lp` und `lp2`.

Es können auch mehrere Dateinamen einer Warteschlange übergeben werden, und je nach Konfiguration des Drucksystems dürfen dies auch Dateien unterschiedlichster Formate sein:

```
user@linux:~ > lpr *.jpg test.ps README /var/log/messages
```

Die zweite und häufig interessantere Anwendungsmöglichkeit ist die, bei der `lpr` die Standardeingabe `STDIN` liest: So kann `lpr` das letzte Element einer Shell-Pipe-Kette sein. Ein Beispiel dafür wäre

```
user@linux:~ > dvips test.dvi | psselect 1-10 | psnup -2 | lpr
```

7.2 Die Datei /etc/printcap

Die Datei /etc/printcap ist die Konfigurationsdatei für angeschlossene Drucker. In ihr werden sowohl sämtliche lokale als auch Netzwerkdrucker definiert.

Beispielsweise könnte die Datei wie folgt aufgebaut sein:

```
# This file was generated by /usr/sbin/magicfilterconfig.
#
lp|hplj4l|HP Laserjet 4L:\
        :lp=/dev/lp0:sd=/var/spool/lpd/hplj4l:\
        :sh:pw#80:pl#72:px#1440:mx#0:\
        :if=/etc/magicfilter/ljet4ldith-filter:\
        :af=/var/log/lp-acct:lf=/var/log/lp-errs:
lp1|color|HP Deskjet 610:\
        :lp=/dev/lp1:sd=/var/spool/lpd/color:\
        :sh:pw#80:pl#72:px#1440:mx#0:\
        :if=/etc/magicfilter/dj550c-filter:\
        :af=/var/log/lp-acct:lf=/var/log/lp-errs:
```

Wie üblich sind Zeilen, die mit einem **#** beginnen, Kommentarzeilen. Als nächstes sieht man eine Zeile, in der ein Drucker definiert wird

Die durch **|** getrennten Einträge sind gleichwertige Namen (Aliase) , unter denen der Drucker angesprochen werden kann. Das abschließende **** dient dazu, den Zeilenumbruch zu unterdrücken, denn eigentlich sollten alle Einträge für einen Drucker in einer Zeile stehen.In den weiteren Zeilen stehen die Informationen zu den Druckern, dabei ist jede Information durch einen **:** vom andern getrennt.

7.3 Kommandos

7.3.1 line printer queue - lpq

Eine Übersicht über den aktuellen Inhalt der Warteschlange gibt es jederzeit mit dem Kommando `lpq` (bzw. `lpq -PDRUCKER`);

`lpq [OPTIONEN]`

Option	**Beschreibung**
-a	Zeigt die Warteschlangen aller Drucker an.
-l	Ausgabe im Long-Format.
-P Drucker	Zeigt die Warteschlange des angegebenen Drucker an.
Benutzer	Zeigt die Aufträge des/der angegebenen Benutzer(s) an.
Jobnummer	Zeigt die Druckaufträge mit der/den angegebenen Druckauftragsnummer(n) an.

`lpq` zeigt alle Druck-Jobs in der Reihenfolge an, in der sie an den Drucker geschickt werden. Wichtig ist dabei die Job-Nummer, denn über diese ist es möglich, einen Job auch wieder abzubrechen, ohne dafür die restlichen Druckaufträge auch zu stoppen. Im Mehrbenutzerbetrieb ist auch die Spalte *Owner* von Wert, denn über diese erfährt man sofort sofort, wer den Drucker über Stunden mit Handbuch-Ausdrucken quält.

```
lp is ready and printing
Rank       Owner      Job   Files      TotalSize
active     ukp12      104   testc      119bytes
1st        ukp12      117   uaktiv     419 bytes
2nd        root       118   ende       18bytes
3rd        ukp22      119   vliste     332 bytes

Position in der Warteschlange
           Besitzer des Druckjobs
                     Jobnummer
                         Dateiname
                                       Dateigröße
```

7.3.2 Löschen eines Druckauftrages

Wurde eine Datei zu früh an den Drucker geschickt, kann ein Job jederzeit mit dem Kommando lprm wieder aus der Druckerwarteschlange entfernt werden - das gilt sogar dann, wenn er sich bereits im Ausdruck befindet. Alles, was dazu benötigt wird, ist die Druckjob-Nummer, die mit Hilfe von `lpq` ermittelt werden kann. Sollte der Druckauftrag nicht in der Standard-Warteschlange liegen, ist auch noch der Warteschlangen-Name anzugeben.

`lprm [OPTIONEN]`

Option	Beschreibung
-P Drucker	Gibt den Drucker an, aus dessen Warteschlange ein Auftrag gelöscht werden soll. Geben Sie keinen Drucker an, gilt der Standarddrucker.
Benutzer	Gibt den Benutzer an, dessen Auftrag/Aufträge gelöscht werden sollen.
Jobnummer	Spezifiert den zu löschenden Auftrag anhand der Druckauftragsnummer.

Beispiel:

```
user@linux:~ > lprm -Plp2 42
```

Dabei darf jeder Benutzer übrigens nur die Jobs löschen, die er auch selbst in Auftrag gegeben hat. Um alle eigenen Jobs zu löschen, kann man auch kurz `lprm` eingeben. Besondere Privilegien hat wie üblich `root`: Der Administrator kann beliebige Jobs aus der Queue nehmen, und hier führt ein `lprm` zum Löschen wirklich sämtlicher Einträge.

7.3.3 lp control -lpc

Das Kommando `lpc` steht dem Super-User mit verschiedenen Optionen zur Steuerung der Druckwarteschlangen zur Verfügung. Im Wesentlichen geht es um das (De-) Aktivieren des Druckbetriebs (also Anhalten und Wieder-Anwerfen des Druckers) und die Sperrung und Freigabe der Druckerwarteschlange. Der Unterschied ist schnell erklärt: Eine gesperrte Warteschlange verweigert die Annahme neuer Druckjobs, während eine inaktive Warteschlange zwar neue annimmt aber nicht weiter druckt.

Die vier beschrieben Funktionen erreicht man über Befehle der Form

`lpc stop [all | Drucker]`
: Bendet das Drucken von Aufträgen an dem/den angegebenen Drucker, wenn der laufende Auftrag abgeschlossen ist.

lpc start [all | Drucker]
Aktiviert den Spooldämon für den/die angegebenen Drucker.

lpc disable [all | Drucker]
stoppt die Aufnahme neuer Druckaufträge in die Warteschlange

lpc enable [all | Drucker]
Aktivieren einer Druckwarteschlange

Mit Hilfe des Kommandos lpc ist es auch möglich, den angegebenen Druckauftrag bzw. die angegebenen Druckaufträge, der/die durch die Druckauftragsnummer oder den /die Benutzernamen spezifierten, an den Anfang der Warteschlange zu stellen:

lpc topq Drucker [job.. | Benutzer]

Kapitel 8

Benutzerkommunikation

8.1 write

Mittels `write` lässt sich eine Nachricht auf das Terminal eines anderen Benutzers schreiben, indem eigene Eingaben auf das Terminal des Gegenübers kopiert werden. Die Nachricht wird von der Standardeingabe gelesen. Wenn von der Tastatur gelesen wird, muss die Eingabe mit `[Strg] + [d]` abgeschlossen werden.
Der Empfänger muss allerdings am selben Rechner eine Sitzung offen haben

write USER [TTYNAME]

Wird `TTYNAME` nicht angegeben, und der andere Benutzer ist mehrfach angemeldet, so wählt das Kommando das Terminal mit der geringsten Idle-Zeit (Stillstandszeit - das Terminal, an dem der Partner zuletzt eine Eingabe getätigt hat).

Um mit dem Benutzer namens `moritz`, der am Terminal `pts/3` angemeldet ist, zu kommunizieren, geben wir Folgendes ein:

```
user@linux> write moritz pts/3
Das ist eine Nachricht von user.
[Strg]+[D]
```

Auf dem Terminal des Users `moritz` schaut das Ganze so aus:

```
moritz@linux>
Message from user@sonne on pts/1 at 09:58 ...
```

```
Das ist eine Nachricht von user.
EOF
```

8.2 wall

Das Kommando `wall` sendet eine Nachricht auf das Terminal aller aktiven Benutzer
Der Nachrichtentext wird von der Standardeingabe gelesen, und mit einem Dateiendezeichen `[Strg] + [d]` abgeschlossen.
Das Kommando `wall` wird vor allem für Hinweise des Systemverwalters root, z. B. vor dem Abschalten des Rechners benutzt.

8.3 mesg

Allerdings kann sich jeder Benutzer vor unerwünschten Nachrichten schützen. Mit `mesg` kann eingestellt werden, ob andere Benutzer Meldungen auf das eigene Terminal ausgeben können. Es wird typischerweise verwendet um das Schreiben auf das eigene Terminal mit Hilfe des Kommandos `write` zu kontrollieren. Aber auch Nachrichten, die durch das Kommando `wall` geschickt werden, können blockiert werden.

`mesg [y|n]`

Optionen

`y`	Löscht auch das Heimatverzeichnis des Benutzers
`n`	der Schreibzugriff wird gesperrt

Wird `mesg` ohne Optionen aufgerufen, gibt das Kommando den aktuellen Status aus über den Schreibzugriff aus.

> **Hinweis:**
>
> Nur die Nachrichten des Superusers `root` können nicht abgeschaltet werden.

8.4 mail

`mail` ein sehr einfacher Kommandozeilenclient zum Versenden und Lesen von Nachrichten. Sein Vorteil liegt ganz klar in der Eignung zum Einsatz in Shellskripten, wo eine automatische Mailgenerierung erwünscht wird. Aber auch auf der Kommandozeile ist es eine effiziente Möglichkeit, um schnell eine Email zu schicken. `mail` ist allerdings nicht in der Lage, Attachements zu behandeln.

`mail [-iInv] [-s subject] [-c cc-addr] [-b bcc-addr] to-addr...`

Zum **Senden einer Nachricht** ist das Kommando mit der eMail-Adresse des Empfängers aufzurufen. Gleichzeitig kann der Grund der Mail ("Subject") mit **-s Subject** und weitere Empfänger **-c Adresse[, Adresse]** angegeben werden. Bei Verzicht auf die Option **-s** wird das Programm zur Eingabe des Subjects auffordern:

```
user@linux> mail jemand@irgendwo.de
Subject: Antwort
Alle was nun eingegeben wird, bildet den Inhalt der Mail, bis zum Ende
mit [Ctrl]+[D] oder der Eingabe eines einzelnen Punktes auf einer
neuen Zeile.
.
Cc:
user@linux>
```

Ist neue Mail eingetroffen, ermöglicht der Aufruf von `mail` das Lesen dieser:

```
user@linux> mail
"/var/spool/mail/user": 2 messages 1 new 2 unread
 U  1 jemand@irgendwo.de   Tue Jun 20 08:18  14/446   "Neuigkeiten"
>N  2 jemand@irgendwo.de   Tue Jun 20 08:19  11/411   "Testemail"
& p
Message 2:
From jemand@irgendwo.de Tue Jun 20 08:19:08 2000
Date: Tue, 20 Jun 2000 08:19:08 +0200
From: Jemand von irgendwo <jemand@irgendwo.de>
To: user@linux.de
Subject: Testemail

Alles bleibt beim Alten.
& q
Saved 2 messages in mbox
user@linux>
```

Um die erste **neue Nachricht zu lesen,** ist **p** einzugeben, mit **+** gelangt man zur nächsten und mit **–** zur vorhergehenden (neuen oder ungelesenen) Nachricht. Durch Eingabe einer Ziffer kann eine Nachricht gezielt gewählt werden.

Um eine **Nachricht zu löschen**, ist **d Nummer** einzugeben, wobei Nummer eine einzelne Ziffer, mehrere, durch Leerzeichen getrennte Ziffern oder ein Bereich (2-7) sein kann. Solange **mail** nicht beendet wurde, kann eine gelöschte Mail mit **u Nummer** gerettet werden. Zum **Antworten auf eine Mail** ist **r** einzugeben.

Beendet wird mail durch Eingabe von **q**.
Nach dem Ende von **mail** werden alle gelesenen Mails von der Mailbox in die Datei `~/mbox` verschoben, mit der Kommandozeilenoption **-f** liest **mail** deren Inhalt ein.

Kapitel 9

Softwareinstallation unter Linux

9.1 Der einfache Weg - RPM-Pakete

RPM steht für "Red Hat Paket Manager". Dieses Installationskonzept hat die Firma Red Hat entwickelt und als Open-Source der Linux-Gemeinde zur Verfügung gestellt. Die RPM-Technik erlaubt es Software zu installieren, zu deinstallieren, zu aktualisieren oder eine Installation zu prüfen. Bei RPM-Paketen handelt es sich um vorkompilierte Programmpakete. Dies bedeutet, dass diese nicht erst vom eigentlichen Quellcode in ausführbare Dateien übersetzt werden müssen. Außerdem enthalten diese Pakete z.B. Konfigurtionsdateien die direkt an die richtige Stelle Ihres Systems installiert werden. Bei bedarf werden z.B. auch solche Konfigurationsdateien geändert. Die Informationen zu jedem installiertem Paket werden in einer Datenbank abgespeichert. Solche RPM-Pakete können nur als root-Benutzer installiert werden.

`rpm [OPTIONEN]`

Optionen

`-i`	Installation von Paketen (`--install`)
`-U`	Aktualisierung des Pakets auf eine höhere Versionsnummer (`--upgrade`)
`-e`	Entfernt installierte Pakete (`--erase`)
`-q`	Liefert Informationen über Pakete
`-h`	Zeigt einen Fortschrittsbalken von 50 Schweinegattern (`#`) während der Installaton an. (`--hash`)
`-v`	Ausführliche Informationen
`-vv`	Sehr ausführliche Informationen
`-a`	Zeigt eine Liste aller installierte Pakete (`--all`)(nur mit `-q`)
`-f DATEI`	Zeigt das Paket, das die angegebene Datei enthält (nur mit `-q`)
`-p PAKETDATEI`	Liefert Informationen über die angegebene Paketdatei (nur mit `-q`)
`-c`	Listet nur Konfigurationsdateien auf (`--configfiles`)(nur mit `-q`)
`-d`	Listet nur Dokumentationsdateien auf (`--docfiles`)(nur mit `-q`)
`-i PAKET`	Liefert Informationen über das angegebene Paket (nur mit `-q`)
`-l PAKET`	Listet die enthaltenen Dateien auf (`--list`)(nur mit `-q`)
`-R`	Zeigt die Abhängigkeiten an (`--requires`)(nur mit `-q`)
`-V`	Überprüfung von Dateien auf ihre Übereinstimmung mit den Angaben im Paket

`--force`	Erlaubt das Ersetzen von existierenden Paketen und Dateien
`--nodeps`	Führt keine Abhängigkeitsprüfung durch
`--quiet`	Beschränkt die Ausgabe auf Fehlermeldungen
`--test`	Testet nur und installiert nicht

9.1.1 Installieren von RPM-Paketen

Um ein neues Paket zu installieren muss nur der Befehl **`rpm -i`** zusammen mit dem Paketnamen verwendet werden. Wenn das Paket auf auf anderen Paketen basiert, die nicht installiert sind, gibt `rpm` eine passende Fehlermeldung aus. In diesem Falle müssen die fehlenden Pakete zuerst installiert werden.

Beispiel:

```
root@linux:~ > rpm -ivh VMware-workstation-3.2.0-2230.i386.rpm
Preparing...                ########################################### [100%]
   1:VMwareWorkstation      ########################################### [100%]
root@linux:~ >
```

Das Upgrade eines Pakets zu einer neuen Version wird durch die Option **`-U`** erledigt. Der Upgrade-Modus ist ein spezieller Installationsmodus. Existiert das zu aktualisierende Paket nicht, dann verhält sich **`-U`** wie der Schalter **`-i`**.

Beispiel:

```
root@linux:~ > rpm -Uvh VMware-workstation-3.2.0-2230.i386.rpm
Preparing...                ########################################### [100%]
   1:VMwareWorkstation      ########################################### [100%]
root@linux:~ >
```

9.1.2 Deinstallieren von Programmpaketen

Die Deinstallation von Paketen ist genauso einfach wie deren Installation. Hier ist jedoch Vorsicht geboten. Vor der Deinstallation wird keine Sicherheitsabfrage durchgeführt. D.h. wenn Sie den Befehl mit der Eingabetaste bestätigen wird die Deinstallation direkt ausgeführt. Dabei wird natürlich ein Paket nur dann entfernt, wenn kein anderes Paket auf dieses Paket angewiesen ist. Für die Deinstallition wird der Schalter **`-e`** verwendet

Beispiel:

```
root@linux:~ > rpm -e VMwareWorkstation
```

9.1.3 Informationen sammeln mit rpm

Installierte und nicht installierte Pakete können im Informationsmodus (*Query-Mode*) mit dem Befehl **`rpm -q`** befragt werden.

Ausführliche Informationen über ein installiertes Paket liefert die Schalterkombination **`-qi`**.

Beispiel:

```
root@linux:~ > rpm -qi apache
Name        : apache         Relocations: (not relocateable)
Version     : 1.3.19         Vendor: (none)
Release     : 5              Build Date: Don 03 Mai 2001 10:50:12 CEST
Install date: Mit 12 Sep 2001 20:33:32 CEST      Build Host: localhost
Group       : System Environment/Daemons    Source RPM: apache-1.3.19-
5.src.rpm
Size        : 1272468         License: Apache Group License
Summary     : The most widely used web server on the Internet.
Description :
Apache is a powerful, full-featured, efficient and freely-available
web server. Apache is also the most popular web server on the
Internet
root@linux:~ >
```

Wenn die Informationen über eine RPM-Datei gefragt sind, kommt der Schalter **`-p`** ins Spiel.

```
root@linux:~ > rpm -qpi VMware-workstation-3.2.0-2230.i386.rpm
Name        : VMwareWorkstation        Relocations: (not relocateable)
Version     : 3.2.0                    Vendor: VMware, Inc.
Release     : 2230           Build Date: Die 10 Sep 2002 05:28:44 CEST
Install date: (not installed)               Build Host:
cannes.vmware.com
Group       : Applications/Emulators        Source RPM:
VMwareWorkstation-3.2.0-2230.src.rpm
Size        : 25643758                         License: commercial
Summary     : VMware Workstation
Description :
VMware Workstation Virtual Platform is a thin software layer that
allows multiple guest
operating systems to run concurrently on a single standard PC, without
repartitioning or rebooting, and without significant loss of
performance.
```

Eine Liste der im Paket enthaltenen Dateien bekommt man durch den Schalter **`-l`**.

```
root@linux:~ >  rpm -qlp VMware-workstation-3.2.0-2230.i386.rpm
/etc/vmware
/etc/vmware/installer.sh
/usr/bin/vmnet-bridge
/usr/bin/vmnet-dhcpd
/usr/bin/vmnet-natd
/usr/bin/vmnet-netifup
```

```
/usr/bin/vmnet-sniffer
/usr/bin/vmware
....
```

Es ist sogar möglich herauszufinden, aus welchem Paket eine Datei stammt:

```
root@linux:~ >  rpm -qf /bin/cp
fileutils-4.1-51
```

Und natürlich können alle installierten Pakete angezeigt werden. Da es sich um einige hundert Pakete handelt, empfielt es sich die Ausgabe zu filtern:

Beispiele:

```
root@linux:~ >  rpm -qa | sort | less

root@linux:~ >  rpm -qa | wc –l
…..581

root@linux:~ >  rpm -qa | grep netscape
netscape-plugins-4.78-14
netscape-4.78-34
```

9.2 Programme im Quellcode - die .tar Archive

Software die als Quellcode ausgeliefert wird wird auf eine völlig andere Art und Weise installiert als RPM-Pakete. Der große Unterschied liegt darin, dass diese Programme wie schon zuvor erwähnt im Quellcode vorliegen. Dies hat den großen Vorteil, dass diese Anwendungen von erfahrenen Programmierern auf dessen Bedürfnisse angepasst werden können. Dies hat aber hingegen den Nachteil, dass diese Programmpakete erst aus einer für Menschen leserlichen Form, dem Quellcode, in eine für Computer lesbare Form übersetzt werden müssen. Diesen Vorgang bezeichnet man im Allgemeinen auch als "kompilieren". Manchmal lässt es sich leider nicht vermieden Software in Form von Quellcode-Paketen zu installieren, da diese oft aktueller sind als die RPM-Pakete.
Quellcode-Pakete beinhalten oft viele Dateien. Je nach Umfang der Software sind dies manchmal hunderte. Damit diese vielen Dateien nicht alle einzeln ausgeliefert werden müssen (z.B. durch einen Download aus dem Internet werden diese Dateien zu einem sog. Archiv zusammen gefasst. Oft werden diese Archive dann auch noch koprimiert, damit Speicherplatz gespart wird. Die oben angesprochenen Archive werden in sog. tar-ball-Dateien gespeichert. Diese tragen die Programmendung `.tar`. Sind diese Archive dann auch noch komprimiert, dann kommt zusätzlich die Programmendung `.gz` hinzu. So kommt auch die Programmendung `.tar.gz` zu stande.

Tragen die verwendeten Quellcode-Archive am Dateiende die Endung `.gz`, dann müssen diese zusätzlich "entpackt/dekomprimiert" werden. Dieser erste Schritt kann als normaler Benutzer ausgeführt werden. Der Befehl hierzu lautet wie folgt:

```
user@linux:~ >  tar xvzf programmname.tar.gz
```

Liegt das Archiv ohne die Endung .gz, also unkomprimiert vor, folgt das "auspacken" der Quellcode-Dateien wie folgt:

```
user@linux:~ >  tar xvf programmname.tar.gz
```

In den meisten Fällen wurde jetzt ein neues Verzeichnis angelegt, in dem sich die Quellcode-Dateien befinden. Dieses Verzeichnis trägt in der Regel den gleichen oder einen ähnlichen Namen wie das Programmpaket. Um weiter fortzufahren muß nun in das neu erstellte Verzeichnis gewechselt werden.

Was jetzt folgt wird als das eigentliche "kompilieren" bezeichnet. Dies geschieht meist in drei Phasen. Alle drei Vorgänge können je nach Umfang der zu installierenden Software mehrere Minuten bis hin zu einer Stunde Zeit in Anspruch zu nehmen. Wichtig für die Ausführung der folgenden Befehle ist, dass sie alle aus dem Verzeichnis des Quellcodes ausgeführt werden müssen. Als erstes wird ein sogenanntes Shell-Script ausgeführt, welches eine Dinge einstellt (konfiguriert). Der Name dieses Shell-Scriptes ist genormt und lautet `configure`. Dieser nächste Befehl kann auch mit normalen Benutzerrechten ausgeführt werden:

```
user@linux:~ >  ./configure
```

In nun folgenden Schritt wird der eigentliche Kompiliervorgang gestartet. Auch dieser Befehl kann mit normalen Benutzerrechten ausgeführt werden. Dieser Vorgang wird vermutlich am meisten Zeit in Anspruch nehmen. Der Befehl hierfür lautet wie folgt:

```
user@linux:~ >  make
```

Ist diese Prozedur überstanden und fehlerfrei durchlaufen dann liegt das Softwarepaket jetzt in computerleserlicher Form auf der Festplatte vor. Rein theoretisch kann es jetzt schon gestartet werden. Hierzu müsste man allerdings jedesmal in das Verzeichnis wechseln in dem sich das Programm zusammen mit dem Quellcode befinden. Aus diesem Grund wird das Programm jetzt noch installiert. Danach kann das Programm gestartet weden, gleich in welchem Verzeichnis man sich gerade aufhält. Dieser letzte Schritt muss als "root" ausgeführt werden. Der Befehl zum Installieren ist nun folgender:

```
root@linux:~ >  make install
```

Wurden alle Scripte fehlerfrei durchlaufen, so ist das Programm das zuvor als Quellcode vorlag auf dem System installiert. Einige solcher Programmpakete die im Quelltext vorliegen bieten außerdem die Möglichkeit, die Software wieder zu deinstallieren. Hierzu muss man in das Verzeichnis wechseln in dem sich der Quellcode befindet und folgendes eingeben:

```
root@linux:~ >  make deinstall
```

Dies funktioniert wie gesagt nicht bei allen Paketen. Hier muss man es einfach mal ausprobieren.

Kapitel 10

Weitere Befehle

10.1 grep

Der Befehl `grep` (*Global Regular Expression Print*) durchsucht eine Textdatei nach bestimmten Mustern und gibt die Zeilen, in denen das Muster vorkommt, auf der Standardausgabe aus.

`grep [OPTIONS] MUSTER [DATEILISTE]`

Optionen

`-G`	Interpretiert das `MUSTER` als regulären Ausdruck; Standarteinstellung. Nicht zusammen mit `-F` und `-E` verwenden
`-E`	Interpretiert das `MUSTER` als erweiterten regulären Ausdruck. Nicht zusammen mit `-F` und `-G` verwenden
`-F`	Interpretiert das `MUSTER` als einfache Zeichenkette. Nicht zusammen mit `-F` und `-E` verwenden
`-c`	Zeigt nur die Zeilennummern der gefundenen Zeilen an
`-n`	Zeigt zusätzlich zur Zeile auch die Zeilennummer an
`-v`	Zeigt die Zeilen an, die nicht dem `MUSTER` entsprechen
`-f DATEINAME`	Die Liste der zu bearbeitenden Dateien
`-h`	Unterdrückt die Ausgabe des Dateinamens bei Verwendung einer Dateiliste.
`-i`	Unterscheidet bei der Suche nicht nach Groß- und Kleinschreibung
`-w`	`MUSTER` wird als ganzes Wort und nicht als Teil des Wortes betrachtet
`-l`	Zeigt den Namen der Datei an, wenn die

	Zeile darin gefunden wurde
`-s`	Fehlermeldungen unterdrücken
`-r`	Durchsucht auch die Unterverzeichnisse

Beispiel:

```
user@linux:~ >  grep dialout /var/log/messages
```

Durchsucht die Datei `/var/log/messages` nach Zeilen mit dem Wort `isdn`.

```
user@linux:~ >  grep -il dial *
```

Durchsucht alle Dateien im Verzeichnis nach der Zeichenkette `dial` ohne Berücksichtigung der Groß- und Kleinschreibung und gibt die Namen der Dateien aus, die die Zeichenkette enthalten.

Das Kommando grep wird häufig innerhalb einer Pipe verwendet.

Beispiel:

```
root@linux:~ >  rpm -qa | grep netscape
netscape-plugins-4.78-14
netscape-4.78-34
```

Das Kommando liefert alle installierten RPM Pakete, die die Zeichenkette netscape im Namen haben.

10.2 head

Das Kommando `head` gibt den Anfang einer Datei (Voreinstellung 10 Zeilen) auf der Standardausgabe auf.

`head [OPTIONEN] [DATEILISTE]`

Werden mehrere Dateien angegeben, so fügt `head` den Dateinamen in der Form

```
==> DATEINAME <==
```

als Kopf vor dem Ausgabetext ein.

Optionen

`-c B`	Gibt anstatt der ersten 10 Zeilen, die ersten B Zeichen aus.
`-n N` `-N`	Gibt anstatt der ersten 10 Zeilen, die ersten N Zeilen aus.

`-q`	Unterdrückt die Ausgabe der Dateinamen als Kopfzeile
`-v`	Schreibt immer den Dateinamen als Kopfzeile vor der Ausgabe

```
user@linux:~ >  head -n 5 -v index.html
```

Diese Kommandosequenz gibt die ersten 5 Zeilen der Datei mit dem Dateinamen als Kopf aus.

10.3 tail

Das Kommando `tail` gibt das Ende einer Datei (Voreinstellung 10 Zeilen) auf der Standardausgabe auf.

```
tail [OPTIONEN] [DATEILISTE]
```

Werden mehrere Dateien angegeben, so fügt `tail` den Dateinamen in der Form

```
==> DATEINAME <==
```

als Kopf vor dem Ausgabetext ein.

Optionen

`-c B`	Gibt anstatt der letzten 10 Zeilen, die letzten B Zeichen aus.
`-n N` `-N`	Gibt anstatt der letzten 10 Zeilen, die letzten N Zeilen aus.
`-q`	Unterdrückt die Ausgabe der Dateinamen als Kopfzeile
`-v`	Schreibt immer den Dateinamen als Kopfzeile vor der Ausgabe
`-f`	Mit dieser Option überwacht `tail` kontinuierlich das Ende einer oder mehrerer Dateien. Wird an die Datei etwas angehängt, so werden die Änderungen ausgegeben. Die Überwachung wird mit STRG+C abgebrochen.

Beispiel:

```
user@linux:~ >  tail -n 5 -q seite1.html seite2.html seite3.html
```

Diese Kommandosequenz gibt die ersten 5 Zeilen der Datei mit dem Dateinamen als Kopf aus.

```
user@linux:~ >  tail -f /var/log/poplog
```

Mit dieser Sequenz wird die Datei /var/log/poplog auf Veränderungen überwacht.

10.4 clear

Dieser Befehl löscht den Bildschirm.

```
clear
```

10.5 cal

Das Kommando `cal` zeigt einen Kalender an.

```
cal [[MONAT] JAHR]
```

Für `JAHR` sind Werte zwischen 1 und 9999 erlaubt und für `MONAT` zwischen 1 und 12.

Optionen

`-m`	Zeigt Montag als ersten Tag der Woche an
`-j`	Benutzt das Julianische Datum
`-y`	Zeigt einen Kalender für das aktuelle Jahr

```
user@linux:~ >  cal -m
    Oktober 2002
Mo Di Mi Do Fr Sa So
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

10.6 date

Diese Befehl zeigt das aktuelle Datum und der Uhrzeit an.

```
date [+FORMATSTRING]
```

`FORMATSTRING` ist ein beliebiger Text, der, falls er Sonderzeichen enthält, in Anführungszeichen zu setzen ist. Sinn macht dieser Text allerdings erst, wenn er Platzhalter enthält, die Datum oder Zeit anzeigen. Einige Beispiele für solche Platzhalter sind:

`%m`	Monat	`%H`	Uhrzeit
`%d`	Tag	`%M`	Minuten
`%y`	Jahr (zweistellig)	`%S`	Sekunden
`%Y`	Jahr (vierstellig)	`%T`	Zeit (hh:mm:ss)

Beispiel:

```
user@linux:~ >  date +"Es ist der %d.%m.%y"
Es ist der 30.10.02
```

Der Systemverwalter kann auch Zeit und Datum über diesen Befehl ändern.

```
date MMTTSSmm[[HH]JJ][.ss]
```

Wobei `MM` für Monat, `TT` für Tag des Monats, `SS` für Stunde, `mm` für Minute, `HH` fürs Jahrhundert, `JJ` fürs Jahr und `ss` für die Sekunden stehen.
Für weitere Informationen zu Schaltern und Platzhaltern siehe `man date`.

```
root@linux:~ >  date 10221735
Mon Okt 22 17:35:00 CEST 2001
```