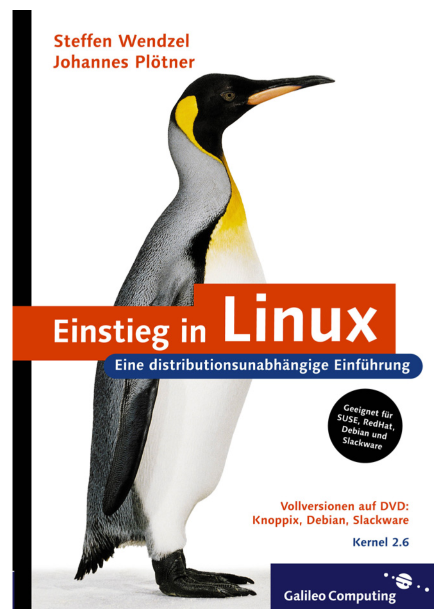
www.galileocomputing.de/725 - Einstieg in Linux - Leseprobe



Auf einen Blick

1	Einleitung	19
2	Installationsvorbereitung	35
3	Linux-Installation	49
4	Die grundlegende Funktionsweise von Linux	71
5	Der Bootstrap-Vorgang	101
6	Prozesse	117
7	Grundlegende Administration	143
8	Die Shell	207
9	Die Editoren	289
10	X11 – Die grafische Oberfläche	305
11	Netzwerkkonfiguration	337
12	Serverdienste	381
13	Drucken und Textverarbeitung	427
14	Speichermedien unter Linux	441
15	Multimedia unter Linux	451
A	Quellcode	469
B	Programmieren unter Linux	481
C	Was ist was? – DOS und Linux	491
D	Glossar	493
E	Literatur	497
	Nachwort	499
	Index	501

Inhalt

1 Einleitung 19

1.1 Was ist Linux? 19
1.2 Die Linux-Distributionen 21
1.3 Unix- und Linux-Geschichte 23
1.3.1 Unix 23
1.3.2 Die Geschichte vom kleinen Linux 26
1.3.3 Etwas Slackware-Geschichte 27
1.3.4 Die Kernelversionen 27
1.4 Die Anforderungen an Ihren Rechner 29
1.4.1 Hardwarekompatibilität 30
1.5 Über dieses Buch 30
1.5.1 Was Sie in diesem Buch erwartet 30
1.5.2 Wie Sie dieses Buch lesen sollten 32
1.5.3 Wo Sie weitere Informationen bekommen 33

2 Installationsvorbereitung 35

2.1 Die Anforderungen an Ihre Hardware 35
2.2 Hardwareunterstützung 36
2.2.1 Hardwarekompatibilitäts-Listen der Hersteller 36
2.2.2 XFree86 und Grafikkarten 37
2.2.3 Linux auf Laptops 38
2.2.4 Andere Geräte 38
2.3 Festplatten und Partitionen 39
2.3.1 Was tun, wenn schon ein anderes ... 41
2.3.2 Windows und Linux 42
2.3.3 Unix und Linux 43
2.3.4 Erstellen eines Backups 43
2.4 Installationsmedien 43
2.5 Erstellen der Bootdisketten 44
2.5.1 Auswahl der Bootdisk 45
2.5.2 Auswahl der Rootdisk 46
2.5.3 Diskettenerstellung unter Windows 47
2.5.4 Diskettenerstellung unter Linux 47

3 Linux-Installation 49

3.1 Slackware-Installation 49

3.1.1 Die Tastaturbelegung 49
3.1.2 Der erste Login 50
3.1.3 Partitionierung der Festplatte 50
3.1.4 Das Tool cfdisk 53
3.1.5 Setup – die eigentliche Installation 54
3.1.6 Test der Installation 62

3.2 SuSE-Installation 63
3.2.1 Installation per Mausklick 63
3.2.2 Partitionierung 63
3.2.3 Paketinstallation 63
3.2.4 Installation der Software 63
3.2.5 Konfiguration 64

3.3 RedHat-Installation 65
3.3.1 Sprache auswählen 65
3.3.2 Die Maus 65
3.3.3 Update oder Neuinstallation 65
3.3.4 Installationstypen und Partitionierung 65
3.3.5 Der Bootmanager GRUB 65
3.3.6 Netzwerkkonfiguration 66
3.3.7 Letzte Arbeiten vor dem Kopieren 66
3.3.8 Kopiervorgang 66
3.3.9 Bootdiskette erstellen 66
3.3.10 X11-Konfiguration 66

3.4 Debian-Installation 67
3.4.1 Die Installations-CD 67
3.4.2 Besonderheiten der Installation 68

3.5 Zusammenfassung 70
3.5.1 Mehrere Systeme 70

4 Die grundlegende Funktionsweise von Linux 71

4.1 Singleuser, Multiuser 72

4.2 Singletasking, Multitasking 72

4.3 Ressourcenverwaltung 72
4.3.1 Speicherverwaltung 72
4.3.2 Swapping 74
4.3.3 Speicherplatz der Festplatte 74
4.3.4 Verwaltung weiterer Ressourcen 74
4.3.5 Schnittstellenbezeichnung unter Linux 75
4.3.6 pseudo devices 76

4.4 Zugriffsrechte 76
4.4.1 Standardrechte 76

4.4.2 Erweiterte Zugriffsrechte 82
4.4.3 Access Control Lists 84

4.5 Das virtuelle Dateisystem 86
4.5.1 Die Verzeichnisstruktur 86
4.5.2 Dateinamen 88
4.5.3 Dateitypen 88
4.5.4 Einhängen von Dateisystemen 92

5 Der Bootstrap-Vorgang 99

5.1 Der MBR 99
5.1.1 Die Partitionstabelle 99
5.1.2 Ein Beispiel 101

5.2 Vom lilo bis zum init–Prozess 101
5.2.1 init 104

5.3 Runlevel-Skripte 105
5.3.1 Wechseln des Runlevels 106
5.3.2 Die Datei /etc/inittab 107
5.3.3 Die Rc–Skripte 109

5.4 getty und der Anmeldevorgang am System 110
5.4.1 (a)getty 110
5.4.2 login 111
5.4.3 Shellstart 111

5.5 Beenden einer Terminalsitzung 112

5.6 Herunterfahren und Neustarten 113
5.6.1 Die Auswahl 113
5.6.2 shutdown 113

6 Prozesse 115

6.1 Was ist ein Prozess? 115
6.1.1 Das Starten eines Programmes 115
6.1.2 Eltern- und Kind-Prozesse 116

6.2 Der Kernel und seine Prozesse 117
6.2.1 Die Prozesstabelle 117
6.2.2 Der Prozessstatus 118

6.3 Prozess-Environment 119

6.4 Sessions und Prozessgruppen 120
6.4.1 Ein Beispiel 121

6.5 Vorder- und Hintergrundprozesse 121
6.5.1 Wechseln zwischen Vorder- und Hintergrund 124

6.5.2 jobs – behalten Sie sie im Auge 125
6.5.3 Hintergrundprozesse und Fehlermeldungen 126
6.5.4 Wann ist es denn endlich vorbei? 128

6.6 Das Kill-Kommando und Signale 128
6.6.1 Welche Signale gibt es? 129
6.6.2 Beispiel: Anhalten und Fortsetzen eines Prozesses 130

6.7 Prozessadministration 131
6.7.1 Prozesspriorität 132
6.7.2 pstree 133
6.7.3 Prozessauflistung mit Detail mit ps 135
6.7.4 top 137
6.7.5 Timing für Prozesse 139
6.7.6 Dateideskriptoren von Prozessen 139

7 Grundlegende Administration 141

7.1 Benutzerverwaltung 141
7.1.1 Linux und Multiuser 141
7.1.2 Das Verwalten der Benutzerkonten 143
7.1.3 Benutzer und Gruppen 146

7.2 Installation neuer Software 148
7.2.1 Das Debian-Paketsystem 149
7.2.2 Das RedHat-Paketsystem 153
7.2.3 Das Slackware-Paketsystem 154
7.2.4 Paketesysteme ohne Grenzen 157
7.2.5 Softwareinstallation ohne Pakete 158

7.3 Backups erstellen 161
7.3.1 Die Sinnfrage 162
7.3.2 Backup eines ganzen Datenträgers 163
7.3.3 Backup ausgewählter Daten 164

7.4 Logdateien und dmesg 168
7.4.1 /var/log/messages 168
7.4.2 /var/log/wtmp 170
7.4.3 /var/log/XFree86.log 170
7.4.4 syslogd 170
7.4.5 logrotate 171
7.4.6 tail und head 171

7.5 Kernelkonfiguration 173
7.5.1 Die Kernelsourcen 173
7.5.2 Los geht's! 173
7.5.3 Start der Konfiguration 175
7.5.4 Kernel-Erstellung 179

7.5.5 LILO **180**
7.5.6 Ladbare Kernel-Module (LKMs) **181**

7.6 Weitere nützliche Programme 184
7.6.1 Speicherverwaltung **184**
7.6.2 Festplatten verwalten **186**
7.6.3 Benutzer überwachen **188**
7.6.4 Der Systemstatus **192**
7.6.5 Offene Dateideskriptoren mit lsof **192**

7.7 Grundlegende Systemdienste 193
7.7.1 cron **193**
7.7.2 at **194**

7.8 Manpages 195

7.9 Dateien finden mit find 197
7.9.1 Festlegung eines Auswahlkriteriums **197**
7.9.2 Festlegung einer Aktion **200**
7.9.3 Fehlermeldungen vermeiden **200**

7.10 Der Midnight Commander 201
7.10.1 Die Bedienung **202**
7.10.2 Verschiedene Ansichten **202**

8 Die Shell 205

8.1 Grundlegendes 205
8.1.1 Was ist eine Shell? **205**
8.1.2 Welche Shells gibt es? **206**
8.1.3 Die Shell als Programm **207**
8.1.4 Die Login-Shell wechseln **207**
8.1.5 Der Prompt **208**
8.1.6 shellintern vs. Programm **210**
8.1.7 Kommandos aneinander reihen **211**
8.1.8 Multi-Line-Kommandos **212**

8.2 Arbeiten mit Verzeichnissen 213
8.2.1 Pfade **213**
8.2.2 Das aktuelle Verzeichnis **214**
8.2.3 Verzeichniswechsel **214**
8.2.4 Und das Ganze mit Pfaden ... **215**

8.3 Die elementaren Programme 216
8.3.1 echo und Kommandosubstitution **216**
8.3.2 sleep **217**
8.3.3 type – intern oder extern? **218**
8.3.4 Erstellen eines alias **219**
8.3.5 cat **220**

8.4 Programme für das Dateisystem 220
8.4.1 mkdir – Erstellen eines Verzeichnisses 220
8.4.2 rmdir – Löschen von Verzeichnissen 221
8.4.3 cp – Kopieren von Dateien 222
8.4.4 mv – Verschieben einer Datei 222
8.4.5 rm – Löschen von Dateien 223
8.4.6 touch – Setzen der Zugriffszeiten von Dateien 223
8.4.7 cut – Abschneiden von Dateiinhalten 224
8.4.8 paste – Zusammenfügen von Dateien 225
8.4.9 tac – den Dateiinhalt umdrehen 225
8.4.10 nl – Zeilennummern für Dateien 226
8.4.11 wc – Zählen von Zeichen, Zeilen und Wörtern 226
8.4.12 od – Dateien zur Zahlenbasis x ausgeben 227
8.4.13 Mehr oder weniger, das ist hier die Frage! 227
8.4.14 head und tail 229
8.4.15 sort und uniq 229
8.4.16 Dateien aufspalten 231
8.4.17 Zeichenvertauschung 232
8.5 Linux und DOS 232
8.5.1 Die mtools 233
8.5.2 dos2unix und unix2dos 234
8.6 Startskripte 235
8.7 Ein- und Ausgabeumlenkung 236
8.7.1 Fehlerausgabe und Verknüpfung von Ausgaben 238
8.7.2 Anhängen von Ausgaben 238
8.7.3 Gruppierung der Umlenkung 239
8.8 Pipes 239
8.8.1 Tee kochen mit tee 240
8.8.2 Named Pipes (FIFOs) 240
8.9 Grundlagen der Shell-Skript-Programmierung 241
8.9.1 Doch was ist ein Shellskript denn genau? 241
8.9.2 Wie legt man los? 242
8.9.3 Das erste Shellskript 242
8.9.4 Kommentare 243
8.9.5 Variablen 243
8.9.6 Rechnen mit Variablen 244
8.9.7 Benutzereingaben für Variablen 246
8.9.8 Arrays 246
8.9.9 Kommandosubstitution und Schreibweisen 247
8.9.10 Argumentübergabe 248
8.9.11 Funktionen 249
8.9.12 Bedingungen 252
8.9.13 Bedingte Anweisungen – Teil 2 255

8.9.14 Die while-Schleife 256
8.9.15 Die for-Schleife 258
8.9.16 Menüs bilden mit select 259
8.9.17 Das Auge isst mit: der Schreibstil 260

8.10 Reguläre Ausdrücke: awk und sed 261
8.10.1 awk – Basics und reguläre Ausdrücke 263
8.10.2 Arbeitsweise von awk 264
8.10.3 Reguläre Ausdrücke anwenden 265
8.10.4 awk – etwas detailierter 267
8.10.5 awk und Variablen 270
8.10.6 Bedingte Anweisungen 273
8.10.7 Funktionen in awk 276
8.10.8 Builtin-Funktionen 277
8.10.9 Arrays und String-Operationen 280
8.10.10 Was noch fehlt 281
8.10.11 sed 281
8.10.12 grep 284

8.11 Ein paar Tipps zum Schluß 286

8.12 Was man sonst noch mit der Shell anstellen kann 286

9 Die Editoren 287

9.1 Anforderungen an Editoren 287
9.1.1 Zeilenorientiert versus bildschirmorientiert 288

9.2 vi 289
9.2.1 Den vi starten 289
9.2.2 Kommando- und Eingabemodus 290
9.2.3 Dateien speichern 290
9.2.4 Arbeiten mit dem Eingabemodus 290
9.2.5 Navigation 291
9.2.6 Löschen von Textstellen 292
9.2.7 Textbereiche ersetzen 293
9.2.8 Kopieren von Textbereichen 293
9.2.9 Shiften 294
9.2.10 Die Suchfunktion 294
9.2.11 Konfiguration 295

9.3 vim 296
9.3.1 gvim 296

9.4 Emacs 297
9.4.1 Konzepte 297
9.4.2 Grundlegende Kommandos 300
9.4.3 Arbeiten mit Puffern und Fenstern 300

9.4.4 Arbeiten mit Mark und Region 301
9.4.5 Das Menü nutzen 301
9.4.6 Den Emacs konfigurieren 301
9.5 Editoren in der Shell 302
9.5.1 Mausunterstützung 302

10 X11 – Die grafische Oberfläche 303

10.1 Funktionsweise 303
10.1.1 Geschichte 303
10.1.2 Client und Server 304
10.1.3 Das Display 305
10.1.4 XFree86 306
10.2 Die Konfiguration 306
10.2.1 Die /etc/X11R6/XF86Config–4 306
10.2.2 xf86config 311
10.2.3 X -configure 311
10.2.4 Tipps und Tricks 311
10.2.5 Testen der Konfiguration 312
10.3 Window-Manager 312
10.3.1 Warum Window-Manager? 312
10.3.2 Klassische Window-Manager 314
10.3.3 Desktop-Umgebungen 316
10.4 X11 starten 321
10.4.1 Aus dem Textmodus 322
10.4.2 Grafische Login-Manager 323
10.4.3 Startskripte für xdm 324
10.5 Die wichtigsten Programme 324
10.5.1 Eterm, xterm und Co. 324
10.5.2 Mozilla 325
10.5.3 The GIMP 327
10.5.4 xchat 328
10.6 Tuning 330
10.6.1 Xinerama und DualHead 330
10.6.2 Mehrere X-Sessions 332
10.6.3 X11 in einem Fenster 333

11 Netzwerkkonfiguration 335

11.1 Etwas Theorie 335
11.1.1 TCP/IP 335
11.1.2 Ihr Heimnetzwerk 337

11.2 Konfiguration einer Netzwerkschnittstelle 339
11.2.1 Konfiguration von Netzwerkkarten mit ifconfig 339
11.2.2 DHCP 342
11.3 Routing 343
11.3.1 Was ist Routing 343
11.3.2 route 344
11.3.3 iproute2 346
11.4 Netzwerke benutzerfreundlich – DNS 346
11.4.1 DNS 346
11.4.2 DNS und Linux 348
11.4.3 Windows und Namensauflösung 350
11.5 Mit Linux ins Internet 351
11.5.1 Das Point-to-Point Protocol 351
11.5.2 Einwahl mit einem Modem 354
11.5.3 Einwahl mit ISDN (Kurztipp) 359
11.5.4 Einwahl mit DSL 363
11.6 Firewalling und NAT 365
11.6.1 Network Address Translation 365
11.6.2 Firewalling mit iptables 367
11.6.3 Firewalling mit dem TCP-Wrapper 369
11.7 Nützliche Netzwerktools 372
11.7.1 ping 372
11.7.2 netstat 373
11.7.3 nmap 375
11.7.4 tcpdump 377

12 Serverdienste 379

12.1 Grundlegende Konzepte 379
12.1.1 Peer-to-Peer-Netzwerke 379
12.1.2 Das Client/Server-Prinzip 380
12.1.3 Und das Ganze mit TCP/IP 381
12.2 inetd 382
12.2.1 Die /etc/inetd.conf 383
12.2.2 TCP-Wrapper 384
12.2.3 update-inetd 384
12.3 Standarddienste 385
12.3.1 finger 386
12.3.2 telnet 386
12.3.3 Die r-Tools 387
12.3.4 Weitere kleine Server 388
12.4 Secure Shell 388

12.4.1 Das SSH-Protokoll 389
12.4.2 Secure Shell nutzen 391
12.4.3 Der Secure Shell-Server 395

12.5 World Wide Web 396
12.5.1 Das HTTP-Protokoll 396
12.5.2 Einrichten eines Apache Webservers 400
12.5.3 Den Apache verwalten 401

12.6 Samba 403
12.6.1 Windows-Freigaben mounten 403
12.6.2 Dateien freigeben 404

12.7 Dateien tauschen mit FTP 404
12.7.1 Das FTP-Protokoll 405
12.7.2 FTP nutzen 406
12.7.3 Einen Server konfigurieren 409

12.8 E-Mail unter Linux 410
12.8.1 Grundlegende Begriffe 410
12.8.2 mail 411
12.8.3 Mails löschen und weitere Aktionen 413
12.8.4 elm, pine, mutt und Co. 413
12.8.5 fetchmail 417
12.8.6 procmail 419
12.8.7 MTAs 420

12.9 Das Usenet 421
12.9.1 Newsgroups 421
12.9.2 Regeln 421
12.9.3 Clients 422

12.10 NFS 423
12.10.1 Dateisysteme exportieren 423
12.10.2 Die Dienste 423
12.10.3 NFS Nutzen 424
12.10.4 NFS unter Unix und Windows 424

13 Drucken und Textverarbeitung 425

13.1 Druckerkonfiguration 425
13.1.1 CUPS – Common Unix Printing System 426
13.1.2 Den Drucker benutzen 429

13.2 TEX 429
13.2.1 So fängt man an 430
13.2.2 Erstellen eines fertigen Dokumentes 432
13.2.3 Das Ergebnis betrachten 432

13.3 groff 433

13.4 **Textverarbeitungsprogramme** 434
13.4.1 OpenOffice.org 434
13.4.2 KOffice 436
13.4.3 Abiword 437

14 Speichermedien unter Linux 439

14.1 **ZIP-Laufwerke** 439
14.1.1 Medien mounten 439
14.2 **Neue Festplatten integrieren** 440
14.2.1 Formatieren 440
14.2.2 Mountpoint festlegen 440
14.3 **Eine Datei als Dateisystem: Loop Device** 441
14.3.1 Und das ganze mit dem RAM 443
14.4 **DVD-Laufwerke** 443
14.4.1 Treiber? 443
14.4.2 Mounting 443
14.5 **CDs brennen** 444
14.5.1 ISO-Dateien erzeugen 444
14.5.2 cdrecord 445
14.5.3 Die benutzerfreundliche Variante: k3b 445
14.6 **USB-Sticks und Co.** 446
14.6.1 USB-Treiber 447
14.6.2 Das Device ansprechen 447
14.7 **SoftRAID und LVM** 448

15 Multimedia unter Linux 449

15.1 **Multimedia unter den Distributionen** 449
15.1.1 Der distributionsunabhängige Weg 449
15.1.2 Debian 450
15.1.3 SuSE 451
15.2 **Konfiguration der Soundkarte** 451
15.2.1 Bis Kernel 2.6 – OSS 451
15.2.2 Ab Kernel 2.6 – ALSA 453
15.3 **Audiowiedergabe** 454
15.3.1 Ausgabemöglichkeiten 454
15.3.2 MP3-Player und Co. 455
15.3.3 Text-to-Speech 456
15.4 **Videos und DVDs** 456
15.4.1 DVDs, DivX und Co. 456

15.4.2 MPlayer 457
15.4.3 XINE 459
15.5 Installation einer TV–Karte 460
15.6 Webcams und Webcam-Software 461
15.6.1 Beispiel: USB IBM Cam einrichten 461
15.6.2 Webcamsoftware 462

A Quellcode 467
A.1 MBR 467
A.2 Samba-Konfiguration 468
A.3 ProFTPd-Konfiguration 472
A.4 Apache Beispiel-Konfiguration 475
A.5 ISDN-Konfigurationsskript 477

B Programmieren unter Linux 479
B.1 Vorraussetzungen 479
B.2 Der GNU Debugger 480
B.3 strace 481
B.4 Makefiles und make 483
B.5 Shared Librarys 484
B.5.1 Vorteile der Shared Librarys 484
B.5.2 Statisches Linken 485
B.5.3 Die Dateien 485
B.6 X11-Programmierung 485
B.7 Integrierte Entwicklungsumgebungen 486

C Was ist was? – DOS und Linux 489
C.1 DOS-interne Befehle 489
C.2 DOS–externe Befehle 489

D Die Buch-DVD 491

E Glossar 493

F Literatur 497

Nachwort 499

Index 501

4 Die grundlegende Funktionsweise von Linux

»Immer nur lernen ohne dabei nachzudenken, das führt zur Verwirrung. Immer nur nachdenken ohne dabei zu lernen, das führt zur Erschöpfung.«
Konfuzius

Dieses Kapitel wendet sich der Funktionsweise des Linux-Kernels und dem virtuellen Dateisystem zu. Sie werden lernen, erste Arbeiten mit dem Dateisystem durchzuführen.

Wie Sie bereits wissen, handelt es sich bei Linux um ein Betriebssystem. Doch welche Aufgaben übernimmt der Kernel denn nun genau und was bedeutet das für den Anwender?

Aufbau des Systems

Nun, ein Betriebssystem ist die Software, die die Verwendung des Computers im Sinne des Anwenders ermöglicht. Das Linux-Betriebssystem besteht aus der Kernkomponente (dem Kernel, der als das eigentliche *Linux* bezeichnet wird) und zugehöriger Software, welche, wie bereits erläutert, in Form von Distributionen vertrieben wird.

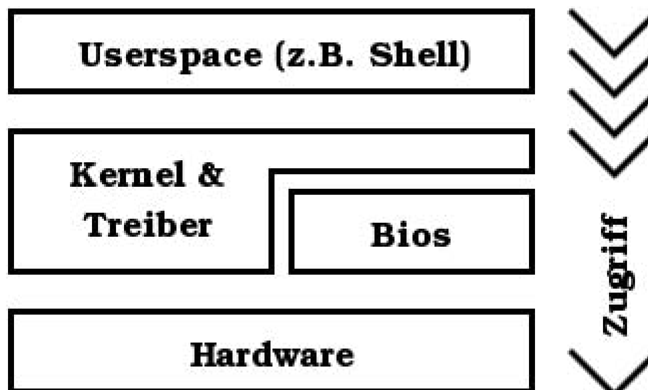


Abbildung 4.1 Aufbau

Der Kernel hat die Aufgabe, die Hardware zu verwalten und Softwareapplikationen die Nutzung dieser zu ermöglichen. Außerdem muss der Kernel das problemlose Nebeneinanderlaufen aller Applikationen gewähr-

leisten, er kümmert sich aus diesem Grund auch um die Prozess- und Speicherverwaltung.

4.1 Singleuser, Multiuser

Unix wurde realtiv früh in der Entwicklungszeit als ein Betriebssystem mit Multiuserfähigkeiten konzipiert. Betriebssysteme die multiuser- bzw. mehrbenutzerfähig sind, untersützen das gleichzeitige Arbeiten mehrerer Benutzer am System. Anders als Betriebssysteme wie Windows 98, OS/2 oder MS-DOS – welche als Singleuser-Systeme bezeichnet werden – verfügen die Benutzer unter Unix-Systemen (und damit auch unter Linux) über so genannte Accounts. Hat man also so einen Benutzeraccount, kann eine Anmeldung am System erfolgen und die Arbeit aufgenommen werden.

4.2 Singletasking, Multitasking

Eine Voraussetzung für den Multiuserbetrieb ist das Multitasking. Ein System, welches Multitasking unterstützt, erlaubt es, mehrere Prozesse und damit mehrere Programme gleichzeitig auszuführen. Auf Systemen mit mehreren CPUs ist dies sogar tatsächlich möglich. Verfügt ein System jedoch nur über eine CPU, muss Multitasking emuliert werden. Dabei wird jedem Prozess vom so genannten *Scheduler* nur ein sehr kleiner Teil der CPU-Zeit zugewiesen, anschließend wird der nächste Prozess in die Verarbeitung geschickt. Dadurch entsteht der Eindruck, dass alle Prozesse gleichzeitig ablaufen.

Prioritäten

Ein Prozess ist dabei *ein Programm in Ausführung*, er verfügt unter Unix-Systemen über eine Reihe von Eigeschaften. Eine wichtige Eigenschaft ist die Priorität eines Prozesses, die mit allen anderen Attributen in der vor allem vom Scheduler genutzten Prozesstabelle des Kernels steht. Über Prozessprioritäten ist es beispielsweise möglich, systemkritischen Anwendungen im Verhältnis mehr Rechenleistung zukommen zu lassen als anderen.

4.3 Ressourcenverwaltung

4.3.1 Speicherverwaltung

gpMarginalieKernel- und Userspace Linux verwaltet den Hauptspeicher in zwei getrennten Bereichen: Es gibt den Kernel- und den Userspace. Der Userspace wird dabei von Anwendungen genutzt, wobei der Kernelspace dem Betriebssystemkern selbst sowie diversen Treibern vorbehalten bleibt. Beide Speicherbereiche sind natürlich physikalisch zusammen auf

dem Hauptspeicher des Rechners abgelegt, jedoch ist der Zugriff auf den Kernelspace nur dem Kernel selbst erlaubt. Für die vom Benutzer ausgeführten Programme wird der Zugriff jedoch verweigert, was vor allem sicherheitstechnische Gründe hat. Genauso wenig kann aber beispielsweise ein Kernelmodul mit einem Treiber auf den Userspace zugreifen, auch wenn dies prinzipiell über Umwege möglich ist. So soll unter anderem verhindert werden, dass Viren oder andere bösartige Programme, die versehentlich von Benutzern ausgeführt werden, das System zerstören.

virtueller Speicher

Denkt man diesen Gedanken weiter, erscheint es nur folgerichtig, dass auch die Speicherbereiche der einzelnen Programme[1] logisch voneinander getrennt werden. Linux nutzt dafür das Konzept des so genannten *virtual Memory*. Bei diesem Konzept werden die Programme überhaupt nicht mit dem realen Speicher konfrontiert, sondern arbeiten auf rein virtuellen Adressen, die dann erst beim Zugriff in reale Speicheradressen übersetzt werden. Damit »sehen« sich die unterschiedlichen Programme überhaupt nicht, und haben demzufolge auch keine Möglichkeiten sich gegenseitig negativ zu beeinflussen.

Die Programme besitzen also selbst keine Möglichkeit, aus ihrer *virtuellen* Umgebung auszubrechen, trotzdem wird ab und zu ein Programm wegen einer Speicherzugriffsverletzung beendet. Diese ist jedoch losgelöst vom Konzept des *virtual Memory* zu betrachten, da das Programm in solchen Fällen meist in seinem eigenen Speicherbereich Unfug treibt. Dort werden dann undefinierte Variablen benutzt, oder es wird über Bereichgrenzen im Speicher hinausgeschrieben, so dass andere wichtige Daten überschrieben werden.[2] Vielleicht fragen Sie sich jetzt, wieso das Konzept des virtuellen Speichers uns hier nicht schützt bzw. warum es keine effektiven Schutzmechanismen für diese Probleme gibt.

Natürlich gibt es zum Beispiel wie in der Programmiersprache Java entsprechende Schutzmechanismen. Die von Unix-Anwendern am häufigsten genutzte Programmiersprache C allerdings bietet diese Sicherheit nicht, erlaubt es dem Programmierer aber systemnaher und damit deutlich effizienter zu programmieren. Und da der virtuelle Speicher die Anwendungen ja schließlich voreinander schützt, ist es für das System als Ganzes egal, wenn einzelne Anwendnungen einmal abstürzen. Der Rest kann, ohne etwas zu »merken«, unbeeinflusst weiterlaufen.

1 Eigentlich spricht man bei Programmen in Ausführung von Prozessen.
2 An dieser Stelle wird der Prozess dann einfach beendet und als Hinweis für Programmierer ein Speicherabild, ein »core dump«, auf die Festplatte geschrieben.

Der Speicher eines Programmes selbst ist nun wiederum in verschiedene Bereiche aufgeteilt. So liegt im virtuellen Speicher eines Prozesses beispielsweise ein so genanntes Codesegment mit den Anweisungen für den Prozessor, ein Segment mit statischen Daten sowie der *Heap* und der *Stack*. Aus dem Heap wird einem Programm beispielsweise dynamisch angeforderter Speicher zugewiesen, und ein Stack ist eine Datenstruktur, die unter anderem Funktionsaufrufe verwalten kann.

4.3.2 Swapping

Beim täglichen Gebrauch gerade von Desktop-Systemen oder kurzzeitig sehr stark ausgelasteter Server kann es durchaus vorkommen, dass Programme mehr Daten in den RAM laden wollen als dieser Platz bietet. Um dieses Problem zu lösen, nutzt Linux das so genannte Swapping. Beim Swapping werden im Moment nicht genutzte Daten aus dem Hauptspeicher auf einen speziellen Teil der Festplatte verschoben, die so genannte Swap-Partition.

Dieses Auslagern schafft freien Hauptspeicher, kostet aber natürlich Zeit, da die Festplatte vielleicht um den Faktor 100 langsamer ist, als der RAM[3]. Die Möglichkeit, Daten aus dem Hauptspeicher einfach in eine Datei im Dateisystem auszulagern, wird unter Linux im Gegensatz zu anderen Betriebssystemen so gut wie nicht genutzt. Durch das virtuelle Dateisystem kann nämlich nicht sichergestellt werden, dass die Datei auch tatsächlich auf der lokalen Festplatte und nicht etwa auf einem über ein Netzwerk angeschlossenen, entfernten Rechner landet.

4.3.3 Speicherplatz der Festplatte

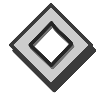
Was für den Hauptspeicher gilt, gilt in ähnlicher Weise auch für die Festplatten. Eine Festplatte hat eine Füllgrenze und ein `ext`-Dateisystem eine maximale Anzahl von Verzeichnissen und Dateien. Hinzu kommt, dass ein Dateisystem unter Linux hierarchisch aufgebaut und mit Zugriffsrechten und diversen Dateitypen versehen ist. Hiermit beschäftigen wir uns im weiteren Verlauf dieses Kapitels aber noch detaillierter.

4.3.4 Verwaltung weiterer Ressourcen

Um mehreren Benutzern den gleichzeitigen Zugriff auf verschiedene Ressourcen zu ermöglichen, werden oft Dämonprozesse eingesetzt. Diese im Hintergrund laufenden Programme haben den exklusiven Zugriff auf

3 Die Zugriffszeiten auf den RAM betragen viele Hundert MB/s, die auf Festplatten nur einige MB/s.

die Ressource und ermöglichen dann, beispielsweise mittels einer Warteschlange und verschiedenen Programmen, den Benutzern den Zugriff auf ihren Dienst.



Als Beispiel für diese Art Dämonprozess ist der Lineprinterdaemon `lpd` zu nennen. Der Dämon verwaltet unter Linux oft den Zugriff auf Drucker. Mittels verschiedener Programme wie `lpr` oder `lpq` können die User dann Dateien drucken und ihre Druckaufträge verwalten. Der `lpd` speichert alle Druckaufträge in einer Warteschlange und schickt dann einen nach dem anderen zum Drucker.

Nun verfügen aber nicht alle Schnittstellen über Dämonprozesse, und nicht jeder Dämonprozess verwaltet eine Ressource. Beispielsweise wird die Verwaltung der Netzwerkverbindungen ganz dem Kernel überlassen. Die Anwenderprogramme müssen hierbei selbst mit Hilfe von Syscalls[4] ihre Anforderungen (z.B. »Gib mir die für mich neu eingetroffenen IP-Pakete ...«) an den Kernel senden. Das Konzept der Dämonprozesse hat nicht nur für das Ressourcenmanagement Bedeutung, wie wir im Prozesskapitel noch sehen werden.

4.3.5 Schnittstellenbezeichnung unter Linux

Geräte werden unter Linux in Form von Dateien repräsentiert. Diese Gerätedateien sind Schnittstellen zu den reellen oder logischen Ressource die sie repräsentieren, und befinden sich unterhalb des */dev* Verzeichnisses.

Gerätedateien werden also von Userspace-Applikationen angesprochen und benutzt – sofern diese die entsprechenden Zugriffsberechtigungen besitzen. Hinter diesen Schnittstellen verbirgt sich oft der entweder direkt in den Kernel kompilierte oder in Form eines Kernelmodules geladene Treibercode.

Um so eine Schnittstelle zu benutzen, führt die jeweilige Applikation dazu mindestens 3 Syscalls durch. Zuerst wird die Gerätedatei geöffnet, dann von dieser gelesen bzw. auf diese geschrieben. Nachdem alle Daten gesendet bzw. gelesen wurden, wird die Schnittstelle wieder geschlossen. Tut dies die Anwendung nicht selbst, so zeigt der Kernel Erbarmen und schließt diese automatisch bei der Beendigung des Programmes.

Deskriptoren

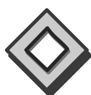
Intern werden geöffnete Dateien über so genannte Deskriptoren verwal-

4 Syscalls sind Systemaufrufe, mit denen wir die Unterstützung des Kernels in Anspruch nehmen.

tet, die dem Programmierer beim Öffnen von Dateien in Form von Variablen übergeben werden. Der Kernel verwaltet intern für jeden Prozess ebenfalls die geöffneten Dateien mittels dieser Deskriptoren und kann so beim Zugriff anhand der Deskriptoren feststellen, welche Datei nun eigentlich gemeint ist.

4.3.6 pseudo devices

Pseudogeräte (pseudo devices) sind Schnittstellen, welche nicht in Form von Hardware vorliegen und nur als Software implementiert wurden. Ein Beispiel dafür ist das so genannte »Datengrab« */dev/null*.



Die Schnittstelle */dev/null* wird als Datengrab bezeichnet, weil alle Daten, die an sie geschickt werden, einfach verschwinden. Diese Schnittstelle hat damit an sich keinen praktischen Nutzen, interessant wird sie erst im Zusammenhang mit anderen Anwendungen. Wie wir im Kapitel zur Shell noch lernen werden, ist es möglich, die Ausgabe von Programmen zum Beispiel in eine Datei umzuleiten. Leitet man so eine Ausgabe von Programmen nun einfach statt standardmäßig auf den Bildschirm nach */dev/null*um, so wird die Ausgabe einfach ignoriert und der Bildschrim bleibt wie gewünscht leer.

Listing 4.1 Wir wollen keine Ausgabe sehen

```
$ Befehl > /dev/null
```

4.4 Zugriffsrechte

UID und GID

Vor die Datei haben die Götter[5] das Recht gesetzt. Wir wollen Ihnen damit sagen, dass es unter Unix generell ein ausgefeiltes und strenges Rechtemanagement gibt. Jeder Benutzer wird dabei eindeutig über eine Nummer identifiziert, die so genannte UID (User ID). Außerdem ist jeder Benutzer noch in einer bis beliebig vielen Benutzergruppen vertreten, die über eine GID (Group ID) referenziert werden.

4.4.1 Standardrechte

Prinzipiell gibt es für eine Datei folgende Rechte: Lesen, Schreiben und Ausführen. Für Verzeichnisse ist der Zugriff ähnlich geregelt: Ob man in ein Verzeichnis schreiben, es lesen[6] oder in es wechseln darf, ist jeweils über ein einzelnes Attribut geregelt.

5 In diesem Fall die Programmierer ...
6 In diesem Zusammenhang ist das Anzeigen der vorhanden Dateien gemeint.

Für jede Datei können dabei die Rechte für den Eigentümer, dessen Gruppe und den Rest, unterschiedlich eingestellt werden. Ändern kann die Rechte nur der Eigentümer und `root`. Dabei ist `root`[7] der Systemadministrator, der von vornherein alles darf. Für ihn gelten keine Einschränkungen durch Rechte, entsprechend gefährlich und grenzenlos ist seine Macht[8].

der Superuser

Der `root`–Account auf einem Unixsystem wird und sollte ausschließlich zur Systemadministration benutzt werden. Wenn zum Beispiel ein neues Programm installiert werden soll oder ein Administrator an den Konfigurationsdateien Änderungen vornehmen will, kommt `root` ins Spiel. Damit kein anderer Benutzer das System kaputtspielt, hat der normale Benutzer nur Schreibrechte auf seine eigenen Dateien – und nicht auf wichtige Systemdateien, wie beispielsweise die installierten Programme. Arbeitet der Systemadministrator auch selbst am System, so sollte auch er mit einem ganz normalen Benutzeraccount arbeiten, und nur wenn nötig zu `root` werden.

Aber wieder von der angewandten Gesellschaftskunde der Informatik zurück zum Rechtesystem unter Linux.

Ich, meine Freunde und der Rest der Welt

Wie bereits angesprochen kann der Eigentümer[9] Rechte für sich, die Gruppe und den Rest der Welt vergeben.

Kontext	Bedeutung
Benutzer	Bei dem Benutzer handelt es sich um den Eigentümer der Datei.
Gruppe	Der zugreifende Benutzer ist in derselben Gruppe wie die Datei.
Rest der Welt	Der Benutzer ist weder Eigentümer noch in der Gruppe der Datei.

Tabelle 4.1 Rechteübersicht

Möchte nun ein Benutzer lesend, schreibend oder ausführend auf eine Datei zugreifen, prüft Linux zuerst, ob er der Eigentümer dieser Datei ist. Ist er das, wird in diesem Feld geprüft, ob er die entsprechenden Rechte

7 Manchmal wird `root` auch als Superuser bezeichnet.
8 Hat ein Hacker einmal Zugriff auf den Rootaccount eines geknackten Systems, ist also alles aus.
9 Und natürlich `root`, den wir an dieser Stelle nicht immer explizit noch mit erwähnen wollen.

hat. Aus der Natur der Dinge folgend, wird dieser Test sicher immer positiv verlaufen.[10] Ist der Benutzer nicht der Eigentümer, wird geschaut, ob er in der entsprechenden Gruppe der Datei ist. Wenn ja, werden die Gruppenrechte abgefragt und so geprüft, ob er die entsprechende Berechtigung besitzt. Ansonsten wird geschaut, welche Rechte für den Rest der Welt vergeben wurden und dann dementsprechend entschieden. Wie immer gilt: Wenn der Benutzer die UID 0 besitzt, also `root` ist, werden alle Rechte gewährt.

Verwaltung von Zugriffsrechten

Natürlich muss man diese Rechte halbwegs komfortabel verwalten können. Wenn Sie mit einer grafischen Oberfläche arbeiten, gibt es dazu wunderschöne und komfortable Dateimanager, wie beispielsweise den `konqueror`. Da sich diese Programme aber von selbst erklären und wir noch etwas ins Detail gehen wollen, werden wir die Veränderungen von Zugriffsrechten mit Hilfe von Shellkommandos beschreiben, auch wenn Sie an dieser Stelle des Buches noch keinen tieferen Einblick in die Shell haben.[11]

ls

Bevor man neue Rechte verteilt, will man vielleicht erst sehen, was für Rechte eine bestimmte Datei besitzt. Ein entsprechendes Shellkommando für dieses Problem ist `ls`. Ruft man `ls` ohne Parameter auf, zeigt es einfach alle Dateien im aktuellen Verzeichnis an.

Listing 4.2 Das Kommando ls

```
$ ls
hallo  test.txt
```

Mit einem Argument können wir `ls` aber überreden, etwas aussagekräftiger zu sein:

Listing 4.3 Ein langes Listing

```
$ ls -l
-rwxr-xr-x 1 hannes users 2344 Sep 13 23:07 hallo
-rw-r--r-- 1 hannes users   23 Sep 13 23:07 test.txt
```

r-w-x

In dieser Ausgabe finden wir schon alle Informationen, die wir bauchen. Beide Dateien gehören der Gruppe `users` und dem Benutzer `hannes`.

10 Und selbst wenn nicht – der Eigentümer kann sich selbst *jedes* Recht auf eine Datei geben.

11 An dieser Stelle soll die Beschreibung genügen, dass die bzw. eine Shell eine Art Kommandointerpreter ist, der getippte Befehle ausführt.

Ganz links finden wir die Rechte, von denen wir schon so viel gehört haben. Dabei stehen die Rechte in der Reihenfolge: Eigentümer, Gruppe und Andere. In unserem Beispiel kann der Eigentümer die Datei *hallo* lesen (r), schreiben (w) und ausführen (x). Alle anderen – Gruppenrechte und Andere sind gleich – dürfen nur lesen (r) und ausführen (x). Da die Ausgabe zwar gewiß gewöhnungsbedürftig, aber doch selbsterklärend und nicht sonderlich kompliziert ist, wollen wir gleich mit dem Modifizieren der Rechte fortfahren.

chmod

Der Befehl der Wahl ist in diesem Fall `chmod`. Bevor wir uns jedoch näher mit der Syntax dieses Befehls befassen, wollen wir noch kurz etwas über die Repräsentation der Rechte sagen. Dazu brauchen Sie nur etwas Binärarithmetik und müssen mit Oktalzahlen umgehen[12], aber keine Angst: wie immer wird nichts so heiß gegessen wie es gekocht wird.

oktales Zahlensystem

Es gibt genau zwei Modi, die ein Recht haben kann: entweder ist es gegeben oder es ist verweigert. Was liegt also näher, als die binäre Darstellung, repräsentiert durch 0 und 1, zu wählen? Bei drei Rechten hat man dann drei Bits, was genau eine Oktalzahl darstellt. Eine Oktalzahl ist eine Zahl zur Basis 8[13] (8 ist 2^3), also wie gesagt durch drei Bits mit jeweils zwei Zuständen darstellbar. Dabei werden die Bits in der Reihenfolge lesen, schreiben und ausführen gesetzt. Die Zahl 7 bedeutet also, da sie alle Bits gesetzt hat, volle Rechte. Die Zahl 6 dagegen hat nur die beiden höherwertigen Bits gesetzt, was in diesem Fall lesen und schreiben bedeutet – ausführen ist nicht erlaubt[14].

Oktalzahl	Übersetzung	Interpretation
777	`rwxrwxrwx`	Alle dürfen lesen, schreiben und ausführen.
644	`rw-r--r--`	Der Eigentümer darf lesen und schreiben, alle anderen nur lesen.
664	`rw-rw-r--`	Wie oben, nur darf jetzt auch die Gruppe schreiben.
600	`rw-------`	Der Eigentümer kann lesen und schreiben, sonst hat niemand Zugriff auf die Datei.

Tabelle 4.2 Rechenbeispiele für Oktalzahlen

12 Freakfaktor! :-)
13 Das heißt, es gibt keine 8 und keine 9, da wir bei 0 zu zählen anfangen, und nach der 7 eine neue Stelle brauchen. Man würde also zählen: 0, 1, 2, 3, 4, 5, 6, 7, 10 usw.
14 Keine Panik, die Rechtearithmetik ist nicht schwer, mit etwas Mathe und Gewöhnung sieht alles ganz einfach aus.

Nun kann man ja Rechte für den Eigentümer, die Gruppe und den Rest der Welt festlegen. Also hat man sich entschieden, die Rechte durch drei Oktalzahlen zu repräsentieren, ein typischer Akt der Rechtevergabe mit `chmod` würde dann zum Beispiel so aussehen:

Listing 4.4 Ändern der Zugriffsrechte

```
$ ls -l test.txt
-rw-r--r-- 1 hannes users 0 Sep 13 23:07 test.txt
$ chmod 664 test.txt
$ ls -l test.txt
-rw-rw-r-- 1 hannes users 0 Sep 13 23:07 test.txt
```

Dieser Aufruf setzt auf die Datei *test.txt* folgende Rechte: Der Eigentümer und die Gruppe dürfen lesen und schreiben, der Rest der Welt nur lesen.

In Kapitel 8 werden wir uns noch etwas näher mit diesem Kommando beschäftigen, und so soll diese Einführung zu diesem Zeitpunkt erst einmal genügen.

umask

Freakfaktor hin oder her, manche Leute finden das Herumrechnen mit binären Repräsentationen diverser Oktalzahlen nicht mal halb so spannend wie den Shoppingkanal. Und der ist schon die Hölle. Für diese Menschen gibt es den Befehl `umask`, mit ihm wird eine Art Voreinstellung für Rechte gemacht, so dass man `chmod` auch mit intuitiveren Parametern sinnvoll aufrufen kann.

Dabei ist zu beachten, dass die Voreinstellung nicht, wie man es vielleicht erwarten würde, alle Rechte gesetzt hat, die man haben möchte, sondern es ist genau anders herum. Man setzt mit `umask` also eine Einschränkung. Aber schauen wir uns das am Beispiel an:

Listing 4.5 umask in Aktion

```
$ umask 022            // umask setzen
$ umask                // umask anschauen
022
$ chmod +w test.txt
$ ls -l test.txt
-rw-r--r--  1 hannes  users  0 Sep 14 02:04 test.txt
```

In diesem Beispiel wird die `umask` auf 022 gesetzt. Nachdem ein `chmod` die Schreibrechte für diese Datei setzt, wird nur dort das Schreibrecht auch wirklich gesetzt, wo `umask` keine Einschränkung vorsieht. In diesem Fall bedeutet dies, dass nur der Eigentümer auch das Schreibrecht bekommt, da es in `umask` für die Gruppe und den Rest der Welt gesetzt ist.

Wäre die umask gleich 000, so gäbe es keine Einschränkungen, und `chmod +w` hätte das Schreibrecht für alle gesetzt. Diese Voreinstellung mittels der `umask` wird übrigens auch automatisch bei neu erstellten Dateien wirksam.

Um die `umask` zu umgehen, kann man natürlich bei `chmod` auch noch die Identifier für den Kontext explizit angeben, so dass `chmod g+x` der Gruppe das Ausführungsrecht gibt. So kann man mit Recht behaupten, dass der `umask`-Befehl vor allem als Schutz vor der eigenen Schusseligkeit gebraucht wird.

chown und chgrp

Nun kann es passieren, dass man eine Datei einem anderen User oder einer anderen Gruppe zuordnen möchte. Für diesen Fall gibt es die beiden Befehle `chown` und `chgrp`, deren Handhabung eigentlich keiner weiteren Erklärung bedarf.

Listing 4.6 Setzen der Eigentumsrechte

```
# chown steffen test.txt
# chgrp autoren test.txt
```

Jetzt gehört die Datei *test.txt* dem Benutzer `steffen` und der Gruppe `autoren`. Natürlich kann man den selben Effekt auch mit einem einzigen Aufruf erzielen, der Befehl `chgrp` ist also nur der Vollständigkeit halber vorhanden:

Listing 4.7 chown setzt die Gruppe

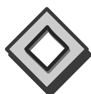
```
$ chown steffen:autoren test.txt
```

su und sudo

Dass man eine Datei dringend braucht und wieder mal keine Rechte für sie hat, kommt leider öfter vor als man denkt. Aber für diesen speziellen Fall gibt es das Programm `su`: Mit `su` kann man in die Identität jedes Benutzers schlüpfen, dessen Passwort man kennt:

Listing 4.8 su in Aktion

```
$ whoami                // Wer bin ich?
hannes
$ su steffen
Password: <tippsel>
$ whoami                // Abrakadabra...
steffen
```



Jetzt bin ich `steffen` und kann also die Datei bearbeiten oder am besten gleich die Rechte richtig setzen.

Ruft man `su` ohne Argument auf, wird automatisch angenommen, dass man `root` werden will. Es wird übrigens empfohlen, nur auf diese Weise als Systemadministrator zu arbeiten, ein extra Login als `root` ist meistens überflüssig.

Das Programm `sudo` öffnet im Gegensatz zu `su` keine Shell mit der Identität des Benutzers, sondern wird genutzt, um ein Programm mit den entsprechenden Rechten zu starten. Wie immer gilt: Ist kein Benutzer über die Option `-u` direkt angegeben, wird `root` als neue Identität genommen.

`$ sudo lilo` führt das Programm `lilo` als `root` aus. Damit man aber als Benutzer die Vorzüge von `sudo` genießen kann, muss man mit einem entsprechenden Eintrag in der `/etc/sudoers` eingetragen sein:

Listing 4.9 Die /etc/sudoers

```
...
# Den Benutzern der Gruppe users ohne Passwort alles
# erlauben
%users  ALL=(ALL)       NOPASSWD: ALL
# Dem Benutzer Test das mounten/unmounten des CDROM
# erlauben (hier wird der Befehl direkt angegeben)
test  ALL=/sbin/mount /cdrom,/sbin/umount /cdrom
...
```

Für die genaue Syntax sei Ihnen, wie so oft, die Manpage ans Herz gelegt.

4.4.2 Erweiterte Zugriffsrechte

Unter Unix-Systemen gibt es einige weitere Zugriffsrechte, auf die wir an dieser Stelle kurz eingehen möchten.

Sticky-Bit

Ist das Sticky-Bit (`chmod +t`) für ausführbare Dateien gesetzt, so werden diese beim Start in den Auslagerungsbereich kopiert. Dies kann sich unter Umständen positiv auf die Performance des Programmes auswirken. Wenn es nämlich relativ oft gestartet wird, muss es dann nicht jedes Mal von der Festplatte nachgeladen werden.

Sticky-Verzeichnisse

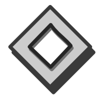
Ist das Sticky-Bit auf ein Verzeichnis gesetzt, so dürfen nur der Superuser und der Eigentümer des Verzeichnisses die darin enthaltenen Dateien löschen und einsehen. Dies wird beispielsweise beim Verzeichnis */tmp* angewandt, wo jeder Benutzer schreiben und Dateien anlegen kann, aber diese Daten trotzdem privat bleiben sollen. Ein gesetztes Sticky-Bit[15] ist am »t« in den Zugriffsrechten zu erkennen und lässt sich mit dem Kommando `chmod +t` setzen.

Listing 4.10 Das Sticky-Bit für Verzeichnisse

```
$ chmod +t dir
$ ls -ld dir
drwxr-xr-t  16 steffen users   1024 Sep 15 19:37 dir
```

Suid-/Sgid-Bit

Setzt man das Suid-Bit auf eine Programmdatei, so wird es zur Laufzeit mit den Rechten des Eigentümers ausgeführt, beim Sgid-Bit mit denen der Gruppe. Dies ist beispielsweise bei Programmen nötig, die direkten Hardwarezugriff erfordern, oder Zugriff auf Dateien wie */etc/shadow* benötigen. Mit diesem Bit kann man also erreichen, dass der Zugriff auf Dateien oder andere Ressourcen in vertrauenswürdigen Programmen gekapselt wird.



Betrachten wir nun einmal die Datei */etc/shadow*. Diese Datei enthält die verschlüsselten Passwörter aller User und darf nur von `root` gelesen und geschrieben werden. Nun macht es aber Sinn, dass Benutzer ihr Passwort selbst ändern können. Dazu gibt es nun das Programm `passwd`, welches ausführbar mit dem Sticky-Bit ist und `root` gehört. Führt ein Benutzer nun `passwd` aus, so kann das Programm mit Rootrechten Änderungen an der */etc/shadow* vornehmen. Der Benutzer hat jedoch keine Möglichkeit zur bösartigen Manipulation, und da er die Datei nicht mal lesen kann, kann er auch nicht daheim heimlich versuchen, die Passwörter zu entschlüsseln.

15 Oft findet man in anderer Fachliteratur auch die Bezeichnung »klebriges Bit«.

Die beiden Bits werden über die Werte `u+s` (SUID) bzw. 4xxx in oktaler Schreibweise und `g+s` (SGID) bzw. 2xxx gesetzt.

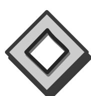
Listing 4.11 Setzen der Bits SUID und SGID

```
$ chmod u+s file
$ chmod 2555 file2
$ ls -l file file2
-rwSr--r--  1 steffen  autoren  0 Sep 15 19:42 file
-r-xr-sr-x  1 steffen  autoren  0 Sep 15 19:42 file2
```

4.4.3 Access Control Lists

Manchmal ist die Welt leider komplizierter, als man sie mit Unix-Rechten abbilden kann. Aus diesem Grund wurden für Linux und einige Dateisysteme, wie beispielsweise XFS oder ext3, die so genannten Access Control Lists, kurz ACLs implementiert. Zum aktuellen Zeitpunkt braucht man, um ACLs nutzen zu können, noch einen speziellen Kernelpatch, aber ab Version 2.6 des Linuxkernels sollen ACLs auch ohne Patch schon standardmäßig im Kernel integriert sein. Je nach Distribution wird man aber trotzdem noch den Kernel neu kompilieren müssen, wenn die Option nicht schon standardmäßig aktiviert wurde.

In diesem Kapitel wollen wir also nicht auf die Installation dieser Erweiterung eingehen, da diese sich mit jeder neuen Kernelversion ändert. Die ACLs wollen wir Ihnen aber trotzdem nicht vorenthalten, und werden sie hier kurz beschreiben.



Access Control Lists sind im Prinzip eine mächtige Erweiterung der Standardrechte. Stellen Sie sich vor, Sie haben eine Firma mit einer Abteilung Rechnungswesen. Diese Abteilung darf natürlich auf eine Datei bzw. eine Datenbank mit den aktuellen Rechnungen zugreifen. Nun ist aber ein Mitarbeiter in Ungnade gefallen und Sie möchten ihm das Recht auf diese eine Datei entziehen, allerdings soll er weiter auf alle anderen Daten der Gruppe Rechnungswesen zugreifen dürfen. Mit Unix-Rechten ist diese Situation, wenn überhaupt, nur sehr kompliziert lösbar, mit ACLs jedoch so einfach, wie mit `chmod` ein Recht zu setzen.

Bei ACLs werden die Rechte nicht mehr nur für den Eigentümer, die Gruppe und den Rest der Welt festgelegt – wie der Name schon sagt, kann mit einer Art Liste der Zugriff für jeden Nutzer und jede Gruppe separat gesteuert werden. Mit einem einfachen Aufruf von

Listing 4.12 ACL Administration mit setfacl

```
$ setfacl -m u:hannes:--- test.txt
$ setfacl -m g:autoren:rwx test.txt+
```

werden die Einträge der Datei *test.txt* für den Benutzer `hannes` und die Gruppe `autoren` modifiziert. Dem Benutzer (gekennzeichnet durch ein vorangestelltes u:) `hannes` wurden alle Rechte entzogen, da auf `---` gesetzt, und der Gruppe (g:) `autoren` alle Rechte gegeben.

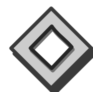
Möchte nun ein Benutzer auf eine Datei zugreifen, werden zuerst die Standardrechte aktiv. Ist er der Besitzer, läuft alles wie gehabt. Ansonsten werden die ACLs gefragt, und es gilt die Regel: die speziellste Regel greift. Ist also ein Eintrag für den Benutzer selbst vorhanden, zählt dieser, ansonsten der Eintrag der Gruppe, soweit vorhanden. Die Rechte aus der ACL können dabei aber nur soweit gehen, wie es die Standardgruppenrechte der Datei erlauben. Damit stehen also die Unix-Rechte über den ACLs, und alles hat seine Ordnung. Wenn allerdings kein spezieller Eintrag für den Benutzer oder seine Gruppe existiert, werden wie bisher die Vorgaben für den Rest der Welt bindend.

Eine ACL für eine bestimmte Datei oder ein bestimmtes Verzeichnis kann man sich übrigens mit `getfacl <Datei>` ähnlich wie bei `ls -l <Datei>` ansehen.

Listing 4.13 getfacl

```
$ getfacl file.txt
#file:file.txt
#owner:jploetner
#group:users
user::rw-
user:swendzel:rw-
group::r--
mask::rw-
other::---
```

Hier im Beispiel hat also der Benutzer `swendzel` noch ein explizit angegebenes Schreibrecht. Ansonsten sieht man die normalen Eigentümer- und Gruppen- sowie die sonstigen Rechte, sowie die durch die Gruppenrechte gegebene effektive Maske für die ACLs.



4.5 Das virtuelle Dateisystem

In den bisherigen Kapiteln des Buches wurde schon teilweise auf das virtuelle Dateisystem Bezug genommen, ohne jedoch genau zu erklären, was sich dahinter verbirgt. Andererseits wissen Sie aber schon eine ganze Menge, zum Beispiel, dass das Dateisystem nicht unbedingt mit der Festplatte gleichzusetzten ist, oder das Hardware über so genannte Gerätedateien repräsentiert wird. Keine Angst, wenn Ihnen das Ganze bisher mehr als spanisch vorkommt, denn erst jetzt werden auch die letzten Geheimnisse gelüftet.

Abstraktion

Das Dateisystem selbst ist in erster Linie völlig abstrakt und basiert auf dem altbekannten System von Datei und Verzeichnis. Der so genannte Root »/« ist die Wurzel des Dateisystems und damit das höchste Verzeichnis. In diesem Stammverzeichnis kann es wie in jedem Unterverzeichnis auch wieder Unterverzeichnisse, normale Dateien, Gerätedateien, FIFOs oder Verweise (Links) geben. Um leichter den Überblick zu behalten, gibt es aber eine mehr oder minder exakt definierte Ordnung[16], wo welche Dateien in einem Verzeichnis abzulegen sind. Im Folgenden möchten wir die wichtigsten Verzeichnisse und ihre Aufgaben einmal kurz vorstellen.

4.5.1 Die Verzeichnisstruktur

/boot

Das Verzeichnis */boot* beinhaltet alle Dateien, die mit beim Systemstart unmittelbar benötigt werden. Dort findet man einen oder durchaus auch einmal mehrere Kernel und andere Dateien wie beispielsweise die Konfigurationsdatei für den aktuellen Kernel. Das Verzeichnis ist normalerweise nur ein paar Megabyte groß, und wurde früher oft auf eine besondere Partition ausgelagert, die nah am Anfang der Festplatte war. Damals war es aus technischen Gründen noch notwendig, dass sich der Kernel innerhalb der ersten 1024 Zylinder[17] befindet, sonst konnte das System nicht gebootet werden.

daheim bei Linux'ens

Im */home*-Verzeichnis besitzt jeder Benutzer eines Unix-Systems sein eigenes Verzeichnis, auf das nur er Schreibrechte hat. Dort ist der Platz für alle seine persöhnlichen Dateien, Schriftstücke und Bilder. Im Normalfall sollte sich ein User auch nur für dieses Verzeichnis zu interessieren haben – der Rest der Festplatte geht ihn einfach nichts an. Die Chance, dass ein normaler Benutzer dort etwas kaputt macht, ist einfach zu groß. Dem-

16 Diese Ordnung ist allerdings von Unix-Derivat zu Unix-Derivat verschieden, aber hat man einmal das Prinzip verstanden, so findet man sich auch auf fremden Unix-Systemen, die nichts mit Linux zu tun haben, schnell zurecht.

17 Eine Maßeinheit, die den geometrischen Aufbau einer Festplatte beschreibt

zufolge sind deren Rechte auf andere Verzeichnisse sehr eingeschränkt, und sie dürfen oft, wenn überhaupt, nur lesen. Es reicht ja auch aus, Programme auszuführen. Veränderungen an diesen, wie das Einspielen neuer Versionen, ist nur dem Systemadmistrator erlaubt.

root's home

In */root* hat der Systemadministrator sein Heimatverzeichnis. Allerdings sollte mit dem Rootaccount nicht produktiv gearbeitet werden – es kann einfach zu viel schiefgehen. Von daher wird man in diesem Verzeichnis im günstigsten Fall maximal ein vergessenes und schon leicht angestaubtes Backup von vor zwei Wochen finden.

Wichtig ist vielleicht noch anzumerken, dass unter Linux' eigenen Dateisystemen standardmäßig 5% des Plattenplatzes[18] für den Superuser reserviert bleibt. Wenn Sie also als Benutzer arbeiten und die Meldung bekommen, dass die Platte voll sei und Sie nichts mehr machen können, können Sie als `root` dieses Problem immer noch beheben.

/var

Im Verzeichnis */var* finden sich Dateien, welche sich unter ständiger Veränderung befinden. Dazu gehören die Mailpostfächer in */var/mail*, die Logdateien der Dämonprozesse und des Systems in */var/log*, die Laufzeitdaten in */var/run* und Ähnliches.

temporäre Dateien

Temporäre Dateien werden im Verzeichnis */tmp* abgelegt. Dabei ist */tmp*, wie in diesem Kapitel bereits erwähnt, das einzige Verzeichnis neben */home*, wo auch normale Benutzer Schreibrechte haben – allerdings mit der Einschränkung des Sticky-Bits. So können alle Programme hier temporäre Dateien anlegen, ohne dass vertrauliche Daten preisgegeben werden. Das */tmp*-Verzechnis ist oft auch gar nicht physikalisch auf der Festplatte vorhanden, sondern einfach nur ein Bereich im RAM, der mal eben als RAMdisk gemountet wurde. So ist der Zugriff erstens extrem schnell und zweitens werden alle Daten beim nächsten Neustart automatisch verschwunden sein.

/usr

Ein weiteres wichtiges Verzeichnis mit diversen Subverzeichnissen ist */usr*. Hier finden sich im Gegensatz zum */var*-Verzeichnis ausschließlich statische Daten, wie wichtige Programme in */usr/bin*, Includedateien für die C(++)-Programmierung in */usr/include*, die Superuserbinarys in */usr/sbin/*, und die Dateien des X-Window-Systems in */usr/X11R6/* und Ähnliches. Dass sich unter */usr* nur statische Dateien befinden, hat den Vorteil, dass eine eventuell vorhandene eigene Partition für das Verzeichnis auch readonly gemountet werden kann, so dass selbst ein Hacker mit Rootrechten keine Programme zu seinen Gunsten verändern kann.

18 Dieser Wert ist veränderbar. Mehr dazu in Kapitel 12.

4.5.2 Dateinamen

Bei der Betitelung von Dateien sollten einige Regeln beachtet werden. Generell sollte von der Verwendung der Sonderzeichen abgesehen werden. Dies kann die Arbeit mit dem System und mit Shellskripten unglaublich vereinfachen. Es ist darüber hinaus zu beachten, dass Linux einen Unterschied zwischen der Groß- und Kleinschreibung der Dateinamen macht.[19]

Versteckte Dateien

Dateinamen, welche mit einem Punkt beginnen, gelten als versteckt und werden bei einem normalen `ls`-Aufruf ohne entsprechende Parameter nicht angezeigt, wie beispielsweise die *~/.cshrc* oder die *.login*. Mit dem `-a` Parameter des Kommandos werden diese Dateien jedoch ausgegeben. Versteckte Dateien sind also nicht versteckt, damit man sie nicht findet – dazu gibt es weiß Gott andere Möglichkeiten. Sie sind versteckt, damit sie während der täglichen Arbeit nicht stören und einen Überblick über die wirklich wichtigen Dateien verhindern. Aus diesem Grund sind meist nur benutzerspezifische Konfigurationsdateien im */home*-Verzeichnis des entsprechenden Users versteckt.

Listing 4.14 Anzeige der schüchternen Dateinamen

```
$ ls -a
.Xauthority   .bash_history  .bashrc  .less  .lessrc
...
```

4.5.3 Dateitypen

Wie Sie vielleicht schon geahnt haben, ist unter Linux – wie auch unter anderen Unix-Systemen – wirklich (fast) alles in Dateien realisiert. Sogar die Verzeichnisse sind auf Dateisystemebene eigentlich nur eine besondere Art von Dateien – ein Grund mehr, sich mal mit den unterschiedlichen Dateitypen näher auseinander zu setzen.

Reguläre Dateien

Hierbei handelt es sich um alle *normalen* Dateien. Dazu zählen Textdateien (z.B. Konfigurationsdateien wie */etc/hosts*), Binärdateien (JPEG-Bilder, Wavedateien, MP3-Dateien) sowie ausführbare Dateien wie Shellskripte oder im Binärformat (a.out oder ELF) vorliegende Programme wie */bin/ls*.

19 Es sei denn, Sie mounten ein Windows-Dateisystem ...

Verzeichnisse

Wie versprochen, sprechen wir bei Dateitypen natürlich auch die Verzeichnisse an. Sie existieren im hierachischen Aufbau des VFS und können jeweils wieder Dateien und Unterverzeichnisse beinhalten. Eigentlich speichern sie aber nicht die Dateien bzw. Verzeichnisse, sondern lediglich Verweise auf die an anderer Stelle gespeicherten Daten.

Das System sieht Verzeichnisse als Blockeinheiten, welche aus verschiedenen Größen[20] bestehen. Ein Verzeichnis beinhaltet die Nummern der Inode-Einträge aller Dateien, welche aus Sichtweise des Anwenders in diesem untergebracht sind. Soll eine Datei also von einem Verzeichnis in ein anderes verschoben werden, so muss einfach nur dieser Eintrag geändert werden. Das macht das Verschieben übrigens schneller als das Kopieren von Dateien, bei dem die Datei wirklich als Ganzes kopiert wird.

Sehen wir uns einmal die Ausgabe eines Verzeichnisses mit dem `ls`-Kommando an. Über den Parameter `-d` kann die Ausgabe des Verzeichnisses selbst, anstelle des Inhalts, erzielt werden.

Listing 4.15 nicht-rekursive Verzeichnisanzeige

```
$ ls -ld /root
drwx------ 5 root wheel 512 Sep 14 23:56 /root/
```

Gerätedateien

Gerätedateien haben Sie in diesem Kapitel bereits kennengelernt. Diese besonderen Dateien ermöglichen, wie Sie bereits wissen, den Userspace-Zugriff auf Kernelspace-Treiber. Man unterscheidet dabei zwischen Block- und Character-Geräten. Der Unterschied zwischen beiden Typen ist, dass bei den Character-Geräten nur jeweils ein Zeichen (ein character), bei den Block-Geräten dagegen ein ganzer Datenblock übertragen wird. Block-Geräte, wie beispielsweise Festplatten, müssen somit nicht für jedes Zeichen extra angesprochen werden, sondern können ganze Datenblöcke auf einen Schlag und damit gepuffert übertragen. Character–Devices, wie eine serielle Schnittstelle, sind hingegen ungepuffert und übertragen alle Daten sofort.

Major und Minor

Jedem Gerät ist eine so genannte Major- und Minor-Number zugeordnet. Die Major-Number ist dem Treiber zugeordnet, die Minor-Number selbst

20 Generell gilt: Je mehr Inodenummern, desto größer muss der Block sein, um diese zu speichern. Typische Blockeinheiten haben eine Größe von 2 KByte oder 4 KByte.

gibt die Gerätenummer für den Treiber an. So werden also die abstrakten und eigentlich willkürlichen Gerätenamen einem entprechenden Treiber zugeordnet.

Ein Aufruf des `ls`-Kommandos zeigt beim Gerätedateien-Listing entweder `b` für block device oder `c` für character device an.

Listing 4.16 Die Primärfestplatte

```
$ ls -l /dev/hd*
brwx-rw--- 1 root disk 3, 0 Sep 14 23:56 /dev/hda
brwx-rw--- 1 root disk 3, 1 Sep 14 23:56 /dev/hda1
brwx-rw--- 1 root disk 3, 2 Sep 14 23:56 /dev/hda2
...
```

devfs

Im neuen `devfs` bzw. `sysfs` dagegen, das ab Kernel 2.6 standardmäßig aktiv ist, wird einem Device nicht mehr über Major- und Minor-Nummern sein entsprechender Treiber zugeordnet, sondern über den Namensraum. Das `devfs` ist daher wie auch das `procfs` ein Pseudodateisystem, das Zugriffe auf seine Elemente (Dateien und Verzeichnisse) direkt im Kernel behandelt und das für den Benutzer sichtbare Dateisysteminterface nur für die einfache Handhabung bereitstellt.

Damit bietet das `devfs` unter anderem folgende Vorteile:

- **Schnelligkeit**

  Beim Zugriff auf einzelne Devices entfällt der Zugriff auf die Platte, um die Datei und damit die Major- bzw. Minor-Nummer zu lesen. Stattdessen wird über Dateiname und Verzeichnis der angesprochenen Gerätedatei der entsprechende Treiber geladen. Welcher Treiber das im Einzelnen ist, weiß der Kernel.

- **Platz**

  Das `devfs` ist virtuell, benötigt also nur etwas RAM für den Code und steht in keinem Verhältnis zu den Platzansprüchen des alten Systems. Auf Desktop- oder Server-Rechnern ist das zwar weniger relevant, wohl aber auf *embedded devices*.

- **Übersicht**

  Das */dev*-Verzeichnis ist jetzt auch übersichtlicher, da nicht über 1000 Dateien mit möglichen Geräten vorgehalten werden müssen. Die Treiber registrieren stattdessen während ihrer Initialisierung nur alle Gerätedateien, für die auch Hardware vorhanden ist.

Sockets

Sockets sind abstrakt und beschreiben Verbindungen zwischen zwei Endpunkten. Die einzelnen enthaltenen Informationen sind je nach Typ des Sockets (z.B ein Streamsocket für TCP, ein Datagrammsocket für UDP oder so genannter Unix–Domainsocket) verschieden.

Sockets werden beim `ls -l`-Kommando mit einem `s` versehen. Dabei handelt es sich jedoch immer um einen bestimmten Sockettyp, nämlich um einen Unix-Domainsocket. Andere Sockets, wie TCP-Sockets, die wir im Netzwerkkapitel näher behandeln, sind nur rein logische Repräsentationen von Verbindungsendpunkten und liegen damit nicht im Dateisystem.

Pipes und named-Pipes (FIFOs)

Pipes und named-Pipes (so genannte FIFOs) dienen zur Kommunikation zwischen Prozessen.[21] Wir werden uns in Kapitel 8 näher mit ihnen beschäftigen.

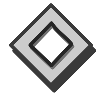
FIFOs werden beim Dateilisting via `ls`-Kommando mit einem `p` versehen.

Listing 4.17 Erstellung und Darstellung einer FIFO

```
$ mkfifo myfifo
$ ls -l myfifo
prw-r--r-- 1 swendzel users 0 Sep 15 18:04 myfifo
```

Links

Es gibt zwei Sorten von Links: symbolische und harte. Symbolische Links (oft auch als *Softlinks* oder *Symlinks* bezeichnet) sind die eigentliche Form eines Links. Bei ihnen handelt es sich um eine spezielle Datei, die als Inhalt schlicht den Dateinamen enthält, auf den gezeigt wird.



Erstellt man einen Link mit dem Namen *link* im Verzeichnis */usr* auf eine Datei (beispielsweise die Datei */home/swendzel*), so verweist *link* auf */home/swendzel*. Wird also *link* eingegeben, wird in Wirklichkeit die Datei */home/swendzel* angesprochen, und alle Änderungen an einer der beiden Dateien gelten automatisch auch für die andere.

symbolische Links

Links werden mit dem Kommando `ln` erstellt. Durch den Parameter `-s` erstellt man einen symbolischen Link.

21 Dafür wird die Bezeichnung Interprocess Communication, IPC, verwendet.

Im folgenden Beispiel wird über das `touch`-Kommando die Datei `myfile` erstellt und anschließend ein symbolischer Link `link` auf diese Datei erstellt.

Listing 4.18 Ein symbolischer Link

```
$ touch myfile
$ ln -s myfile link
$ ls -l link
lrwxr-xr-x 1 swendzel users 6 Sep 15 18:23 link ->
myfile
```

Hardlinks Ein Hardlink ist jedoch verschieden zum symbolischen Link. Er verweist nicht auf eine andere Datei, sondern stellt nur eine weitere Referenz für die eigentliche Datei dar. Das bedeutet, dass das Ziel des Links sich technisch gesehen in zwei Verzeichnissen gleichzeitig befindet. Außerdem hat die Datei einen erhöhten Linkcounter. Den braucht man, wenn man die Datei aus einem Verzeichnis löschen will. Die Datei muss ja auf der Festplatte bleiben – schließlich ist sie auch noch in einem anderen Verzeichnis.

Die zweite Spalte bei einem `ls -l`-Aufruf gibt immer die Anzahl der vorhandenen Links und damit den Linkcounter an.

Listing 4.19 Hardlinks

```
$ ln myfile hardlink
-rw-r--r-- 2 swendzel users 0 Sep 15 18:23 hardlink
-rw-r--r-- 2 swendzel users 0 Sep 15 18:23 myfile
```

4.5.4 Einhängen von Dateisystemen

Sie haben das Buch noch nicht in die Ecke geschmissen? Das freut uns! Kommen wir nun zu einem sehr wichtigen Thema, dem Einhängen von Dateisystemen. In der Fachsprache benutzt man hierfür das Wort »mount«, was eigentlich nur die englische Übersetzung des Wortes ist. Sprechen Sie jedoch vom »Mounten« einer CD, so weiss jeder Linux-Administrator genau, was Sie von ihm wollen.

Wie Sie bereits wissen, beinhaltet das virtuelle Dateisystem (VFS) Verzeichnisse. Diese können auch als Mountpoint (also Einhängepunkt) für andere Dateisysteme dienen. Oft wird beispielsweise das CD-Laufwerk in das Verzeichnis »/cdrom« gemounted.

mount

Um ein Dateisystem einzuhängen, wird also das Kommando `mount` benutzt. Dabei wird das Dateisystem mit dem Parameter `-t`, das zu mountende Gerät und der Mountpoint angegeben. Das Gerät kann sowohl ein CD-ROM Laufwerk als auch eine Festplattenpartition, eine Netzwerkresource (Network Filesystem) oder Ähnliches sein.

Listing 4.20 Beispiel eines mountings

```
# mount -t ext2 /dev/hdb1 /public
```

Hier wurde die erste Partition der zweiten Festplatte[22], auf der sich ein `ext2`-Dateisystem befindet, in das Verzeichnis */public* gemountet.

Rufen Sie `mount` ohne Parameter auf, um alle momentan eingehängten Dateisysteme zu sehen:

Listing 4.21 Was haben wir denn Feines eingehängt?

```
# mount
/dev/hda5 on / type ext2 (rw)
/dev/hda1 on /dos type vfat (rw)
none on /dev/pts type devpts (rw, gid=5, mode=620)
none on /proc type proc (rw)
```

Mit dem Kommando `umount` wird ein Dateisystem wieder ausgehängt. Einsteigern bereitet dieses Kommando jedoch oft Kopfschmerzen, da sich so manches Dateisystem nicht wieder ganz einfach unmounten lässt. Dies liegt oft daran, dass ein Prozess sich noch in diesem Dateisystem beschäftigt – beispielsweise befindet man sich gerade selbst im Mountpoint.

Listing 4.22 Unmounten einer Partition

```
# umount /public
```

eject

Ein weiteres, wichtiges Kommando ist `eject`. Mit diesem werden Laufwerke (z.B. DVD-Laufwerke) geöffnet, um das Medium herauszunehmen. Doch Achtung: CD-Laufwerke lassen sich nicht öffnen, solange sie gemountet sind. Alle Dateisysteme müssen umounted werden, bevor diese physikalisch entfernt werden dürfen, ansonsten kann Datenverlust auftreten!

22 Genauer gesagt: der Primary Slave des IDE Hostadapters

Listing 4.23 Immer raus damit!

```
# eject /dev/cdrom
```

df und du

Die beiden Befehle df und du geben Ihnen Informationen über den Speicherverbrauch einzelner Dateien, Verzeichnisse und ganzer Dateisysteme.

Mittels du erfährt man die Größe einer Datei oder eines Verzeichnisses inklusive aller Subverzeichnisse und Subdateien. Ohne Parameterangabe wird die Größe in Blockeinheiten ausgegeben. Mit dem -h-Parameter ist es jedoch möglich, sich eine automatisch gewählte (passende) Ausgabeeinheit anzeigen zu lassen.[23]

Listing 4.24 Das du-Kommando

```
$ du dir
83076   dir
$ du -h dir
41M     dir
```

Eine Übersicht über die Belegung der Dateisysteme gibt das Kommando df. Auch hierbei werden ohne entsprechende Angabe des -h-Parameters die Blockeinheiten als Belegungseinheit[24] gewählt.

Listing 4.25

```
$ df -h
Filesystem     Size Used Avail Use% Mounted on
/dev/hda5      1.6G 1.2G 441M  73%   /
/dev/hda1      2.0G 789M 1.2G  38%   /dos
```

Die Datei /etc/fstab

Die Datei */etc/fstab* enthält Informationen zum Mounten einzelner Dateisysteme. Sie legt den Mountpoint, das entsprechende Dateisystem und einige Mountoptionen fest.

Listing 4.26

```
# cat /etc/fstab | head -2
/dev/hda1    /     ext2   defaults   1   1
/dev/hda2    swap  swap   defaults   0   0
```

23 Das -h steht für *human readable*.
24 Es existieren noch weitere Parameter wie -k für eine Kilobyte-Angabe.

Der Aufbau dieser Datei ist tabellarisch gehalten. Jeder Datensatz wird in einer eigenen Zeile platziert, jedes Attribut wird mittels Leerzeichen vom nächsten getrennt.

Aufbau

Die erste Spalte legt das Blockgerät (also die Gerätedatei des Speichermediums) fest, welches gemounted werden soll. An dieser Stelle können auch Netzwerkdateisysteme in der Form `Host:Verzeichnis` angegeben werden.

In Spalte zwei ist der Mountpoint anzugeben. Handelt es sich bei einem Datensatz jedoch um das Swap-Dateisystem, so ist hier kein Mountpoint, sondern *swap* anzugeben.

Das dritte Feld legt das Dateisystem fest. Auf einer CD-ROM befindet sich nämlich ein ganz anderes Dateisystem, als auf einer Windows-Partition oder einer Linux-Partition. Generell können hier folgende Dateisysteme angegeben werden:

- 'minix'

  Das Minix-Dateisystem. Hierbei handelt es sich um ein bereits in die Jahre gekommendes Dateisystem mit sehr starken Beschränkungen u.a. für die Länge der Dateinamen.

- 'ext'

  Der Vorläufer des für Linux hauseigenen Dateisystems ext2.

- 'ext2'

  Dieses Dateisystem erlaubt recht lange Dateinamen und benutzt Inodes zur Verwaltung der Dateien.

- 'ext3'

  Die aktuelle Journaling-Version des ext2-Dateisystems. Diese extended-Dateisysteme sind speziell für Linux entwickelt worden, und somit natürlich in erster Linie zu empfehlen. Sie sind abwärtskompatibel, man kann also eine ext3-Partition mit einem ext2-Treiber mounten und alles läuft glatt. Außerdem entfällt bei ext3 ein langes Überprüfen der Partition, wenn beispielsweise durch einen Stromausfall das Dateisystem nicht ordentlich unmountet werden konnte. Das passiert normalerweise beim Shutdown des Systems automatisch.

- 'xfs'

  SGIs XFS. Dieses schon alte Dateisystem benötigt einen Kernelpatch, bietet sich jedoch besonders für die Verwaltung sehr großer Datenmengen an und unterstützt Access Control Lists und wie ext3 auch Journaling.

- 'reiserfs'

  Das ReiserFS ist ein relativ neues und sehr weit verbreitetes journaling Dateisystem, welches binäre Bäume als Grundlage seiner Datenverwaltung benutzt. Als das ext3-System noch nicht fertig war, wurde ReiserFS auf Grund seiner Journaling-Fähigkeiten ext2 oft vorgezogen.

- 'swap'

  Das swap-Dateisystem – ein Pseudodateisystem – wird zur Auslagerung momentan nicht benötigter Hauptspeicherdaten benutzt.

- 'msdos'/'vfat'

  Microsofts FAT16/32-Dateisysteme. Sollten Sie eine ältere Windows- oder DOS-Partition benutzen, so kann diese auch von Linux aus genutzt werden.

- 'ntfs'

  Microsofts NTFS wird ebenfalls unterstützt.

- 'iso9660'

  Dieses Dateisystem wird auf CD-ROMs verwendet.

- 'nfs'

  Das Netzwerkdateisystem NFS[25] wird für die Speicherung von Dateien auf Fileservern genutzt. Ein so von einem anderen Rechner gemountetes Dateisystem ist für den Benutzer bis auf Performanceaspekte identisch mit lokalen Verzeichnissen.

- 'procfs'

  Das Prozessdateisystem. Es enthält unter anderem Informationen über die aktuellen Prozesse des Rechners sowie andere Einstellungen und Laufzeitdaten des Kernels. Dieses Dateisystem ist ein echtes Pseudodateisystem, da Sie die Dateien und Verzeichnisse zwar sehen, aber alles wird während Ihres Zugriffs zur Laufzeit für Sie erstellt, es wird also keinerlei Platz auf der Festplatte benötigt.

Die vierte Spalte wird zur Festlegung einiger Optionen benutzt. Mehrere Optionen werden dann durch ein Komma getrennt. Es gibt folgende Optionen:

- `auto / noauto`

  Mit diesen Optionen wird festgelegt, ob ein Dateisystem automatisch beim Booten gemounted werden soll. Wenn man ein Dateisystem nicht beim Booten mountet, reicht später ein einfaches `mount` mit

25 network filesystem

dem Mountpoint oder dem Device als Parameter, um das Dateisystem einzubinden.

- `user=steffen,gid=1000`

  Mit einem solchen Parameter können die Zugriffsrechte für den Zugriff auf ein Dateisystem gesetzt werden.

- `ro / rw`

  Mit diesen Optionen kann festgelegt werden, ob ein Dateisystem nur lesbar (ro, read-only) oder mit Lese- und Schreibzugriff (rw, read & write) gemountet wird.

- `suid / nosuid`

  Über die `suid`-Option können Sie festlegen, ob Dateien mit SUID- bzw. SGID-Berechtigungen ausgeführt werden dürfen.

- `sync / async`

  Soll ein asynchroner oder synchroner I/O-Zugriff auf das Medium erfolgen?

- `atime / noatime`

  Regelt, ob die Zugriffszeiten auf Dateien (nicht) angepasst werden sollen.

- `dev / nodev`

  Erlaubt (keine) Nutzung von Character- und Block-Geräten von diesem Medium, sprich das Dateisystem, auf welchem sich das Verzeichnis */dev* befindet, sollte diese Option sinnvollerweise gesetzt haben.

- `exec / noexec`

  Diese Option erlaubt bzw. verhindert die Ausführung von Binärdateien.

- `user / nouser`

  Mit der `nouser`-Option hat nur `root` die Berechtigung dieses Medium zu mounten. Ist die `user`-Option gesetzt, ist dies dementsprechend also erlaubt.

- `default`

  Default-Option

  Diese Option setzt `rw`, `suid`, `dev`, `exec`, `auto`, `nouser` und `async`.

Es existieren noch einige weitere, teilweise dateisystemspezifische Optionen, welche an dieser Stelle nicht erläutert werden sollen. Falls Sie sich für diese Optionen interessieren, so hilft Ihnen die mount–Manpage[26] weiter.

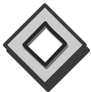
26 Wir werden uns im späteren Verlauf des Buches genauer mit den Manpages beschäftigen.

Spalte Nummer fünf beinhaltet entweder eine `1` oder eine `0`. Ist eine `1` gesetzt, so wird das Dateisystem für die Backup-Erstellung mittels des `dump`-Kommandos markiert. Da dieses Kommando aber kaum noch genutzt wird, brauchen sie sich über diesen Wert keine Gedanken zu machen. Wenn sie es genau nehmen, sollten allerdings alle Wechselmedien mit einer `0` gekennzeichnet werden. Schließlich würde man ja – wenn überhaupt – nur die lokalen Platten, aber keine zufällig eingelegten CD-ROMs sichern wollen.

Die letzte Spalte (eine `2`, `1` oder eine `0`) ist für die Setzung des fsck-Flags vorgesehen. Ist es mit einer Zahl größer Null gesetzt, überprüft fsck beim Booten nach einem fehlerhaften `unmount` [27] das Dateisystem auf Fehler. Die Zahlen selbst geben dabei die Reihenfolge beim Überprüfen an. Man sollte daher die Rootpartition (/) mit einer `1` und alle anderen Platten und Partitionen mit einer `2` versehen. Dort ist die Reihenfolge schließlich egal.

Mounten einer Windows-Partition

Sehr viele Anwender bevorzugen es, Linux als Parallel-Installation zu einer bereits vorhandenen Windows-Installation zu benutzen. Anschließend soll die Windows-Partition unter Linux gemountet werden.



Dies geschieht, wie bereits bekannt sein sollte, mit dem Kommando `mount`. Mit dem `-t`-Parameter ist es dabei wie gesagt möglich, das Dateisystem anzugeben. Für den Fall, dass */dev/hdb1* die FAT32-Partition ist und im Mountpoint */mnt/dos* gemountet werden soll, würde folgender Aufruf die FAT-Partition mounten:

Listing 4.27 Mounten einer DOS-Partition

```
# mount -t vfat /dev/hdb1 /mnt/dos
```

Da Sie aber nun bereits in der Lage sind, die */etc/fstab*-Datei zu nutzen, könnte natürlich auch einfach folgender Eintrag erzeugt werden und mit einfachem `mount /mnt/dos` die Partition gemountet werden.

Listing 4.28 /etc/fstab und FAT

```
/dev/hdb1     /mnt/dos  vfat    defaults   0   0
```

27 Beispielsweise beim Absturz des Rechners

8 Die Shell

»Wer den ganzen Tag mit anderen zusammen in Gruppen verbringt, in seinen Reden keine vernünftigen Themen berührt und es liebt, seinen Geisteswitz auf Kleinigkeiten zu verschwenden, der wird schwerlich Großes leisten.«
Konfuzius

Herzlichen Glückwunsch! Sie sind nun am wohl wichtigsten Kapitel des Buches angelangt. Die Shell ist das A-und-O unter Linux und wird Ihnen wohl auch in den nächsten Jahren der Linux-Nutzung Gesellschaft leisten.

In diesem Kapitel stecken größtmögliche Bemühungen, um auch möglichst viele Feinheiten der Shell-Grundlagen zu erklären. Trotz des Umfanges dieses Kapitels beschränkten wir uns jedoch oftmals auf das Wichtigste.

8.1 Grundlegendes

Wir sind nun an dem Punkt angelangt, an dem wir eine große Menge des Stoffes einspeisen, welchen wir teilweise vorweggenommen haben. Dies und der Punkt, dass die Shell das wohl wichtigste Arbeitsmittel eines Unix-Anwenders ist (stellen Sie sich einmal MS-DOS ohne Batch-Dateien und Eingabeaufforderung vor), spiegeln die zentrale Rolle dieses Kapitels wieder.

Zunächst werden wir uns mit den Grundlagen der Shell, einigen wichtigen und populären Shell-Programmen und Shell-Variablen beschäftigen. Anschließend widmen wir uns der Shell-Skriptprogrammierung, erstellen Beispielskripts, welche Ihnen zugleich den Linux-Alltag erleichtern werden und befassen uns mit den zwei beliebten Tools `awk` und `sed`.

8.1.1 Was ist eine Shell?

Eine Shell ist quasi das Programm, in welches Sie Ihre Befehle und Kommandos zum Aufruf von Programmen eingeben. Möchten Sie beispielsweise das Programm X starten, so erteilen Sie der Shell den Befehl dazu. Die Shell interpretiert das Kommando (daher werden Shells auch oft als Kommandointerpreter bezeichnet) und leitet es bei Bedarf an den Kernel weiter (z.B., um ein Programm auszuführen).

Login Shell

Wie bereits in vorherigen Kapiteln erläutert, wird nach dem erfolgreichen

Login die Login-Shell eines Benutzers gestartet. Von dieser Shell aus werden alle Programme der Arbeitssession gestartet, aber auch Sub-Shells als Kind-Prozesse erzeugt.

Beim Verlassen der Login-Shell logt man sich aus dem System aus. Zur Weiterarbeit ist eine erneute Anmeldung über das `login`-Programm notwendig.

8.1.2 Welche Shells gibt es?

sh

Ohne an dieser Stelle zu weit ins Detail zu gehen, sollte erst einmal ein Wort zu den populärsten Shells fallen. Lassen Sie uns mit der Bourne-Shell beginnen. Diese wurde Ende der 70er geschrieben. Aufgrund zu geringer Fähigkeiten dieser Shell wurde später die C-Shell entwickelt. Hierbei wurde die Syntax an die Programmiersprache C angelehnt und die Arbeit mit der Shell etwas komfortabler gestaltet.

csh

ksh

Später wurde die Korn-Shell entwickelt. Diese baut auf den Funktionalitäten sowie der Syntax der Bourne-Shell auf. Hinzu kommt, dass die Features der C-Shell übernommen wurden und nun eine recht beliebte, wenn zunächst auch kostenpflichtige[1] Shell zur Verfügung stand.

bash

tcsh

Später wurden die freie »Bourne-Again-Shell« (`bash`) sowie die TC-Shell (`tcsh`) entwickelt, welche von vielen heutigen Unix-Nutzen verwendet werden. Die `tcsh` baut, wie der Name schon sagt, auf der `csh`, die `bash` wiederum auf der `sh` und `ksh` auf.

zsh

Später wurden noch weitere Shells wie die Scheme-Shell (`scsh`) entwickelt, welche auf der funktionalen Sprache »Scheme« basiert (und sich über ihre Syntax steuern lässt) und daher besonders für Entwickler dieses LISP-Dialektes interessant ist. Eine äußerst beliebte Shell mit Bourne-Shell Syntax ist die Z-Shell (`zsh`). Diese Shell ist modular aufgebaut und verfügt über einige Raffinessen (siehe AUGTTZSH).

In diesem Buch werden wir uns mit der Bourne-Shell sowie der `bash` befassen, da diese verbreiteter als die C-Shell Reihe sind und die meisten Shell-Skripte (unter anderem auch die Runlevelskripte für den Startvorgang der Linux-Distributionen), deren Syntax und Features verwenden.

1 Auf heutigen Linux-Systemen ist nicht die original kostenpflichtige Korn-Shell, sondern die freie Version `pdksh` (public domain korn shell) installiert.

8.1.3 Die Shell als Programm

Zunächst ist die Shell ein Programm wie jedes andere. Über den Aufruf des Programmes wird es also schlicht und einfach gestartet. Normalerweise übernimmt das Programm `getty` diesen Startvorgang. Jedoch kommt es sehr oft vor, dass ein Benutzer eine Shell in der Shell starten möchte. Beim Verlassen einer auf diese Weise gestarteten Shell befindet man sich anschließend wieder in der vorherigen Shell. Beeendet man jedoch wie in Kapitel 5 beschrieben die *login-Shell*, loggt man sich aus dem System aus.

Listing 8.1 Shell-Start

```
user$ ksh     // Start der Korn-Shell
$ csh         // Start der C-Shell
% exit        // zurück zur Korn-Shell
$ exit        // zurück zur Login-Shell
user$ exit    // beenden der Shell-Session
logout

myhost login:  // getty wartet auf erneute Anmeldung
```

8.1.4 Die Login-Shell wechseln

Mit dem Programm `chsh` ist es möglich, die Login-Shell eines Benutzers und dessen Identitätsangaben, etwa den Vor- und Zunamen, zu verändern.

Listing 8.2 chsh

```
# chsh <Benutzername>
// Editor wird gestartet, Änderungen müssen
// eingetragen werden
...
```

Doch bedenken Sie beim Ändern der Login-Shell Folgendes: Es werden nur jene Shells akzeptiert, die auch im System registriert sind. Die Registrierung erfolgt wiederum in der Datei */etc/shells*. Um beispielsweise die in */usr/local/bin* abgelegte `zsh` in dieser Datei hinzuzufügen, genügt es, eine entsprechende Zeile anzuhängen:

Listing 8.3 /etc/shells

```
/bin/sh
/bin/csh
/bin/ksh
```

```
/bin/bash
/bin/tcsh
/usr/local/bin/zsh   # zsh wird nun auch aktzeptiert
/usr/local/bin/scsh  # Scheme-Shell
```

8.1.5 Der Prompt

Eine Shell verfügt über einen Standardprompt und einige Nebenprompts. Doch was ist eigentlich solch ein Prompt? Im Prinzip die Aufforderung zur Eingabe eines Kommandos. Je nach Shell – und besonders nach der persönlichen Konfiguration – sehen diese Prompts anders aus. Bisher benutzten wir im Buch meist den Prompt `user$`, jedoch wäre auch durchaus eine andere Kombination denkbar.

Die `bash` bietet an dieser Stelle eine Menge Möglichkeiten zur Gestaltung des Prompts. Nach dem Start der `bash` sieht man in der Regel Folgendes: `bash-2.05b#`.

Der Prompt wird über eine Variable[2] mit dem Namen »PS1« gestaltet. Setzt man diese Variable auf einen Wert »X«, so ändert sich der Prompt in »X«. Im Normalfall exportiert man diese Variable, jedoch reicht uns an dieser Stelle zunächst einmal das ganz normale, nicht dauerhafte Verändern des Promptes aus.

Im folgenden Beispiel geben wir über das `echo`-Programm den Inhalt der PS1-Variable aus und setzen diesen anschließend neu. Das Ergebnis zeigt, dass der neue Prompt übernommen wurde. Mit dem `unset`-Kommando kann der Prompt gelöscht werden. Am Ende wird wieder unser Standardprompt gesetzt.

Listing 8.4 Setzen des Prompts

```
user$ echo $PS1
user$
user$ PS1="prompt > "
prompt > PS1="Steffen% "
Steffen% unset PS1
PS1="user$ "
user$ ls
```

2 Eine Variable speichert einen variablen Wert. In diesem Fall das Aussehen Ihres Prompts. Sie trägt den Namen PS1 und ist über diesen ansprechbar. Doch mehr dazu später ...

...

Dieser Prompt bietet uns jedoch noch keine netten Features wie die Anzeige des Arbeitsverzeichnisses. Für Aufgaben dieser Natur werden so genannte Escape-Sequenzen in den Prompt eingebettet.

Sequenz	Wirkung
\a	Ausgabe eines Tones im PC-Speaker
\d	Zeigt das Datum an
\e	Escape-Zeichen
\h	Der Hostname (z.B. 'rechner')
\H	FQDN–Hostname (z.B. 'rechner.netzwerk.edu')
\j	Anzahl der Hintergrundprozesse
\l	Name des Terminals
\n	Neue Zeile (Prompts können tatsächlich über mehrere Zeilen verteilt werden).
\r	Carriage-Return
\s	Der Name der Shell
\t	Zeit im 24-Stunden-Format
\T	Zeit im 12-Stunden-Format
\@	Zeit im AM/PM-Format
\u	Der Name des Benutzers
\v	Die Version der `bash`
\V	Wie \v, jedoch mit Patch-Level
\w	Gibt das Arbeitsverzeichnis an
\W	Gibt nur das aktuelle Arbeitsverzeichnis ohne höhere Ebenen der Verzeichnishierachie an.
\#	Anzahl der bereits aufgerufenen Kommandos wärend der Shell-Session des Terminals.
\$	Ist man als normaler Benutzer eingeloggt, erscheint ein Dollar-Zeichen, *root* bekommt eine Raute (#) zu sehen.
\\	Ein Backslash

Tabelle 8.1 Escape-Sequenzen

es gibt noch mehr ...

Es existieren noch weitere Escape-Sequenzen, beispielsweise zur Festsetzung der farblichen Hervorhebung. Diese werden im Rahmen dieses Buches jedoch nicht behandelt, und funktionieren nicht auf allen Terminals.

Einige Distributionen und eine große Anzahl der Benutzer verwenden die »Benutzer@Host Verzeichnis$«-Variante, welche ein an dieser Stelle sehr gut passendes Beispiel zur Nutzung der Escape-Sequenzen darstellt.

Listing 8.5 Setzung des `bash`-Prompts mit Escape-Sequenzen

```
user$ PS1="\u@\h \w\$"
swendzel@xyai /usr$ ls
...
```

8.1.6 shellintern vs. Programm

In der Shell werden, wie Sie bereits wissen, Programme – wie beispielsweise ein Programm zum Versenden von E-Mails – gestartet. Hierbei gibt es zwei unterschiedliche Gruppen von Programmen. Die eine Gruppe besteht aus den tatsächlichen, auf der Festplatte abgelegten Programmen. Die andere nennt sich *shellintern*, und wurde sozusagen in die Shell »reinprogrammiert«.

type

Nun können wir Ihnen leider nicht die allgemeine Frage danach beantworten, welche Kommandos intern bzw. extern sind. Dies liegt ganz einfach daran, dass jede Shell verschiedene Kommandos implementiert hat. Die `bash` verfügt zum Beispiel über ein internes `kill`-Kommando, die Bourne-Shell nicht. Möchten Sie Einzelheiten erfahren, nutzen Sie einfach mal das Programm `type`:

Listing 8.6 Builtin oder Programm?

```
$ type kill
kill is a shell builtin
$ type ls
ls is /bin/ls
```

which

Im Gegensatz zu `type`, gibt Ihnen `which` den vollständigen Pfad für ein bestimmtes *Programm* an. Da `kill` auch als Programm auf der Festplatte existiert, wird also dessen Pfad ausgegeben:

Listing 8.7 Pfadausgabe

```
$ which kill
/bin/kill
```

```
$ which ls
/bin/ls
```

Wenn Sie in der `bash` also einfach nur `kill` eingeben, wird die shellinterne Variante benutzt. Bei `/bin/kill` wird hingegen das Programm aus dem entsprechenden Verzeichnis genommen. Für Sie ergeben sich aber außer einer leicht erweiterten Funktionalität und einem latenten Geschwindigkeitsvorteil bei der Bash-Version keine Unterschiede.

8.1.7 Kommandos aneinander reihen

Verschiedene Kommandos in der Shell können aneinander gereiht werden. Der Grund für eine Aneinanderreihung ist die Platzersparnis auf dem Bildschirm und natürlich auch die zeitweise Abarbeitung der Kommandos.

Stellen Sie sich einmal Folgendes vor:

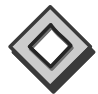
Zehn Kommandos sollen nacheinander ausgeführt werden, wobei jedes einzelne dieser Kommandos voraussichtlich einige Minuten Ausführungszeit beanspruchen wird. Nun könnten Sie die ganze Zeit vor dem Rechner sitzen bleiben und die Kommandos einzeln eingeben. Doch wäre es nicht besser, wenn Sie sich nicht darum kümmern müssten und die Zeit zum Kaffee kochen nutzen könnten?

Der Trennungsoperator

Trennungsoperator

Szenario Nummer eins: Die Kommandos sollen der Reihe nach ablaufen, egal was mit den voherigen Kommandos geschieht. Für diesen Fall verwendet man den Trennungsoperator – das Semikolon (;).

Das folgende Beispiel soll die Wirkung und Anwendung des Trennungsoperators simulieren. Dabei wird unter anderem ein Verzeichnis ausgegeben, welches nicht existiert:



Listing 8.8 Trennungsoperator – ein Beispiel

```
user$ ls VerZeiChniS; uname; find / -name Datei
/bin/ls: VerZeiChniS: No such file or directory
Linux
/usr/local/share/Datei
/home/user/Datei
```

Wie Sie sehen, werden unabhängig vom `ls`-Kommando alle anderen Kommandos ausgeführt.

Weiter nur bei Erfolg

&& Szenario Nummer zwei: Die Kommandos sollen der Reihe nach, unter der Bedingung des Erfolges des vorherigen, ausgeführt werden. Für den Fall, dass ein Programm nicht korrekt abläuft, wird das nächste Programm also nicht gestartet und die Abarbeitung gestoppt. Um dies zu realisieren, wird anstelle des Trennungsoperators ein doppeltes kaufmänisches und-Zeichen (&) geschrieben.

Listing 8.9 Erfolgsbedingte Kommandoreihe

```
user$ ls VerZeiChniS && uname &&  find / -name Datei
/bin/ls: VerZeiChniS: No such file or directory
```

Es stellt sich nun die Frage, woher die Shell weiß, ob ein Kommando korrekt ausgeführt wurde. Die Antwort ist recht simpel: Programme liefern einen so genannten Rückgabewert an die Shell zurück. In der Programmiersprache C würde dies beispielsweise durch ein `return 0;` in der main-Funktion[3] erreicht werden.[4]

Weiter nur bei Fehlschlag

|| Natürlich gibt es auch zum `&&` noch eine Alternative, die genau das Gegenteil bewirkt: Der hintere Befehl wird nur ausgeführt, falls der erste fehlschlägt:

Listing 8.10 Abfangen des Misserfolgs

```
user$ ls VerZeiChnis || echo "HILFE!"
/bin/ls: VerZeiChniS: No such file or directory
HILFE!
```

8.1.8 Multi-Line-Kommandos

Oft erstrecken sich Kommandoeingaben über mehrere Zeilen. Ist dies der Fall, kann der Backslash-Operator (\) verwendet werden, um die aktuelle Eingabezeile in der nächsten fortzusetzen:

Listing 8.11 Kommandos über mehrere Zeilen

```
user$ find /usr/local/share/WindowMaker/ \
```

3 Einige Entwickler halten es seltsamer Weise nicht für nötig, einen Rückgabewert für die main-Funktion zu implementieren. Dies zeugt von einem schlechten und sehr unprofessionellen Programmierstil.

4 Auch Funktionen innerhalb eines Shell-Skripts können solch eine Funktionalität bieten.

```
  -name Image.jpg
/usr/local/share/WindowMaker/Backgrounds/Image.jpg
```

In diesem Buch haben wir diesen Operator auch schon mehrmals benutzt, falls die Zeilen im Listing sonst zu lang geworden wären.

8.2 Arbeiten mit Verzeichnissen

Im Folgenden werden wir auf das Arbeiten mit Verzeichnissen eingehen. Erinnern Sie sich also noch einmal an die Konzepte des VFS und was Sie in diesem Zusammenhang vielleicht beachten müssen.

8.2.1 Pfade

Zuerst müssen wir natürlich diesen Begriff klären: Ein Pfad gibt einen Weg durch den hierarchischen Verzeichnisbaum hin zu einem bestimmten Ziel an.

Ein vollständiges Verzeichnis wie */home/jploetner* beschreibt also auch einen Pfad, nämlich, wie man zu eben diesem Verzeichnis gelangt.

Pfadnamen

Unter Unix kann man Pfade dabei auf zwei unterschiedliche Arten angeben: Es gibt die so genannten *relativen* und die *absoluten* Pfade. Normalerweise gibt man einen Pfad vom Wurzelverzeichnis ausgehend an, so wie etwa diesen hier: */usr/local/bin*. Solch einen kompletten, also vom Wurzelverzeichnis ausgehenden Pfadnamen bezeichnet man als *absolut*.

Arbeitsverzeichnis

Allerdings wissen wir auch, dass jeder Prozess sein *Arbeitsverzeichnis*[5] kennt. Ausgehend von diesem Arbeitsverzeichnis kann eine relative Pfadangabe erfolgen. Befindet man sich beispielsweise im Verzeichnis */usr/local*, könnte man über die Angabe des Verzeichnisses *bin* das Verzeichnis */usr/local/bin* ansprechen.

> Alle Pfadangaben, die mit einem Slash (/) anfangen, werden als absolute, von der Wurzel des Dateisystems ausgehende Pfade behandelt. Alle anderen Pfade werden als relativ, also vom aktuellen Arbeitsverzeichnis ausgehend, behandelt.

5 Oft wird auch der englische Begriff *working directory* verwendet.

8.2.2 Das aktuelle Verzeichnis

Bevor wir wild in den Verzeichnissen hin und her wechseln, wollen wir erst einmal unseren aktuellen Standort erfahren. Dazu nutzen wir am besten den `pwd`-Befehl:

Listing 8.12 pwd – print working directory

```
$ pwd
/home/jploetner
```

8.2.3 Verzeichniswechsel

Zum Verzeichniswechsel wird das Kommando `cd`[6] verwendet. Auch DOS-Benutzern sollte die grundlegende Funktionalität[7] dieses Befehls bekannt sein. Man gibt nämlich das Zielverzeichnis als einziges Argument über einen absoluten oder relativen Pfadnamen an `cd` weiter.

Unter Unix gibt es neben den bereits erwähnten absoluten und relativen Pfaden noch spezielle Bezeichner für Verzeichnisse. Diese sind natürlich jeweils wieder im absoluten bzw. relativen Kontext zu sehen:

- **.**

  Der Punkt (.) bezeichnet das aktuelle Arbeitsverzeichnis.

- **..**

  Zwei Punkte geben das nächst höhere Verzeichnis an. Würden Sie sich also im Verzeichnis */usr/local/bin* befinden, würde ein `cd ..` den Wechsel in */usr/local* zur Folge haben.

- **~-**

  Dies zeigt auf das Verzeichnis, in dem Sie sich vor dem letzten `cd`-Aufruf befanden.

- **~**

  Der Tilde (~)-Operator bezeichnet das Heimatverzeichnis des Benutzers. Ein Spezialfall ist »~Name«, wobei *Name* der Account eines lokalen Benutzers sein muss.

6 `cd` steht für change-directory.

7 Was hier allerdings nicht funktioniert ist `cd..`. Der Grund dafür ist einfach: Linux verlangt vom Benutzer eine saubere Eingabe der Kommandos. Der Kommandoname sollte also nicht mit dem ersten Parameter kombiniert werden, da sich sonst ein neuer (natürlich nicht auffindbarer) Kommandoname *cd..* ergibt.

- **$HOME**

  Die globale Variable $HOME wird beim Login eines Benutzers gesetzt und zeigt auf dessen Heimatverzeichnis.

- **»«**

  Ein parameterloser `cd`-Aufruf wechselt in das Heimatverzeichnis des Benutzers.

Eine Kombination des Pfadnamens mit Hilfe dieser Abkürzungen ist natürlich auch problemlos möglich:

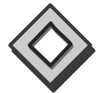
Listing 8.13 Beispiel für einen Verzeichniswechsel

```
$ pwd
/usr/local
$ cd ~/Verzeichnis/../buch/
$ pwd
/home/swendzel/buch
```

8.2.4 Und das Ganze mit Pfaden ...

Natürlich kann man diese ganzen »speziellen« Angaben für Verzeichnisse jeweils mit weiteren, komplettierenden Pfaden kombinieren. So wirkt ein »..« natürlich immer relativ, ein *$HOME* hingegegen natürlich absolut.

Schauen wir uns zur besseren Verständlichkeit ein kleines Beispiel an:



Listing 8.14 Verzeichniswechsel auf Unix-Art

```
// Wir befinden uns in...
$ pwd
/usr/local

// Ins Arbeitsverzeichnis wechseln (kein Effekt):
$ cd . ; pwd
/usr/local

// nächst höheres Verzeichnis:
$ cd .. ; pwd
/usr

// absolute Pfadangabe:
$ cd /usr/local/bin; pwd
/usr/local/bin
```

```
// ...und eine simple relative Pfandangabe
$ pwd
/usr/local
$ cd share/slrn ; pwd
/usr/local/share/slrn

//...komplizierter:
$ pwd
/usr/local/bin
$ cd ../../X11R6/lib/modules; pwd
/usr/X11R6/lib/modules

//...und völlig sinnlos:
$ cd /usr/X11R6/../local/bin; pwd
/usr/local/bin
```

Am besten suchen Sie sich an dieser Stelle noch einmal die Beschreibung der Verzeichnishierarchie aus Kapitel 4 heraus und erkunden Ihr Dateisystem!

8.3 Die elementaren Programme

An dieser Stelle sollen sowohl die grundlegenden internen Shell-Kommandos als auch die wichtigen Programme zur täglichen Arbeit mit der Shell besprochen werden. Fast alle dieser Kommandos »spielen« mit stinknormalem Text herum. Denken Sie jetzt jedoch bitte nicht, dass diese Kommandos nicht zeitgemäß wären: Linux ohne Shell(-Programme) wäre wie Windows ohne grafische Oberfläche, MacOS ohne Maus oder Solaris mit NT-Administratoren.

8.3.1 echo und Kommandosubstitution

Beginnen wir mit dem `echo`-Kommando. Sinn und Zweck von `echo` ist es, Text auf dem Bildschirm auszugeben. Der auszugebene Text wird dabei einfach als Parameter angegeben.

Listing 8.15 Das `echo`-Kommando

```
user$ echo "Das echo Kommando ist nicht immer \
Shellintern."
Das echo Kommando ist nicht immer Shellintern.
```

In Skripten wird echo oft zum Ausgeben der Werte von Variablen benutzt oder um die Ausgaben eines Programmes in Text einzubetten. Da wir Variablen erst im späteren Verlauf dieses Kapitels behandeln, sie an dieser Stelle jedoch kurz gebrauchen werden, sei Folgendes gesagt: Eine Variable speichert einen Wert.

Doch nun zurück zur Ausgabe von Variablen. Es gibt drei verschiedene Möglichkeiten zur Ausgabe von Text mittels echo.

Kommando-substitution

Die erste benutzt ganz normale Anführungszeichen. Bei dieser Variante kann der Wert einer Variablen ausgegeben werden. Die zweite Variante benutzt Backshifts und bewirkt die Ausführung eines Befehles und damit die Integration der Ausgabe dieses Befehles in den eigentlichen Text. Die Ausführung eines Befehls auf diese Weise wird als *Kommandosubstitution* bezeichnet. Variante Numero drei wird in Hochkommas gepackt und erlaubt keine Wertausgaben oder Kommandosubstitutionen. Variablen-Aufrufe und Kommandos werden also direkt ausgegeben. Das folgende Beispiel soll diese Schreibweisen zum besseren Verständnis demonstrieren, später werden wir lernen, dass Variablen über die Syntax »$VARIABLEN_NAME« angesprochen werden.

Listing 8.16 Entwirrung der Schreibweisen

```
// normale Anführungszeichen geben neben dem eigentl.
// Text auch den Wert von Variablen preis:
user$ echo "Der Wert von NUMMER ist $NUMMER"
Der Wert von NUMMER ist 13

// Backticks nutzt man zur Kommandosubstitution:
user$ echo "Heute ist ‘date‘ !"
Heute ist Sat Oct 18 17:41:09 CEST 2003 !

// Mit Hochkommas wird Ihnen gar nichts gegönnt
user$ echo ’Heute ist date und der Wert von X ist $X’
Heute ist date und der Wert von X ist $X
```

8.3.2 sleep

Das sleep-Kommando wartet einen gewissen Zeitraum, bevor es sich beendet. Dies macht hin und wieder Sinn in der Shell-Programmierung oder in Startskripten. Der Aufruf sleep 10 »schläft« für zehn Sekunden.

Listing 8.17 Anwendungsbeispiel für `sleep`

```
user$ tail -30 /home/$USER/.profile
# Beispiel für den Start der grafischen Oberfläche
# Wenn der Benutzer innerhalb von zehn Sekunden
# Strg+C drückt, wird nicht gestartet.

...
echo "Grundinitialisierung fertig"
echo
echo "=============================="
echo "Starte X11 in zehn Sekunden..."
echo "=============================="
sleep 10
startx
```

8.3.3 type – intern oder extern?

Wie bereits oben erwähnt, verfügen verschiedene Shells über differenzierte interne Kommandos, so hat die `bash` beispielsweise ihr eigenes `kill`-Kommando.

type Ein Aufruf von `type` verschafft Klarheit darüber, ob es sich bei einem Programm tatsächlich um eine Datei oder ein internes Kommando handelt.

Listing 8.18 Intern oder extern? Das ist hier die Frage!

```
user$ type type
type is a shell builtin
user$ type uname
uname is /bin/uname
user$ type kill
kill is a shell builtin
user$ type /bin/kill
/bin/kill is /bin/kill

// Starten wir mal eine andere Shell
user$ ksh
$ type type
type is an exported alias for 'whence -v'
$ whence -v uptime
uptime is a tracked alias for /usr/bin/uptime
```

8.3.4 Erstellen eines alias

Unter Unix ist es möglich, einen so genannten *alias* zu erstellen. Ein solcher *alias* wird verwendet, um eine Kurzform für ein, in der Regel etwas längeres Kommando, zu schaffen. Beispielsweise könnte ein *alias* namens »ll« erstellt werden, welcher stellvertretend für »ls -laF« steht.

Um diese Funktionalität der Linux-Shells zu verwenden, wird auf das Kommando `alias` zurückgegriffen. Im Normalfall listet es nur die aktuell eingerichteten Kommando-Aliasse auf, doch es kann auch zur Erstellung eines neuen benutzt werden.

Listing 8.19 Das `alias`-Kommando

```
user$ alias
alias ll='ls -laF'
alias ls='/bin/ls -aF'
user$ ls /projects/netlib
./       eigrp.h   imap.h          pop3.h    test*
../      hello.h   ip4.h           rip.h     test.c
arp.h    http.h    ip6.h           signal.h  testcode/
bgp.h    icmp.h    net_error.h     smtp.h    udpd*
dhcp.h   icmp6.h   net_wrapper.h   snmp.h    udpd.c
dns.h    icmprd.h  netlib.h        tcpscn.h  udpscn.h
egp.h    igrp.h    ospf.h          telnet.h  x11.h
```

Ein neuer *alias* wird via »`alias AName="Kommando"`« erstellt, wobei das Kommando nur in Anführungszeichen geschrieben werden muss, wenn es nicht druckbare Zeichen enthält oder Escape-Sequenzen angewandt werden müssen. Ein alias wird via `unalias` entfernt.

Listing 8.20 Einen eigenen Alias einrichten und löschen

```
$ alias p=pwd
$ p
/home/swendzel/projects/netlib
$ alias backup="tar -czf backup_`date +%d.%m.%y`.tgz \
/home/swendzel/buch
$ backup
tar: Removing leading `/' from absolute path names
  in the archive
$ unalias backup
$ backup
bash: backup: command not found
```

8.3.5 cat

`cat` ist eines der wichtigsten Programme jedes Linux-Rechners und gibt die ihm angegebenen Dateien auf dem Monitor aus. Die Primärverwendung von `cat` ist die Umlenkung der Ausgabe der Datei in eine Datei oder die Weitergabe des Dateiinhalts an ein anderes Programm. Wir werden darauf in naher Zukunft noch genau zu sprechen kommen.

Listing 8.21 Das cat-Programm

```
user$ cat /etc/passwd
root:x:0:0::/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
...
...
nobody:x:99:99:nobody:/:
swendzel:x:1000:100:Steffen W.,,,:/home/swendzel:
/bin/bash

// Es ist möglich, mehrere Dateien ausgeben zu lassen
user$ echo "Inhalt von DateiA"> DateiA
user$ echo "Inhalt von DateiB"> DateiB
user$ cat DateiA DateiB
Inhalt von DateiA
Inhalt von DateiB
```

8.4 Programme für das Dateisystem

Wir erwähnten in vorherigen Kapiteln die Kommandos `ls` (zum Auflisten von Dateien), `ln` (zum Erstellen von Links), `chmod` und `chgrp` (für das Setzen von Zugriffsrechten) sowie Kommandos zur Administration von Access-Controll-Lists. Doch wie erstellt man eigentlich ein Verzeichnis, löscht dieses wieder, kopiert oder verschiebt Dateien?

8.4.1 mkdir – Erstellen eines Verzeichnisses

Ein Verzeichnis wird ganz einfach mit Hilfe des Programmes `mkdir` erstellt. Der Verzeichnisname wird dabei einfach als Parameter übergeben, doch achten Sie bitte auf die in Kapitel 4 angesprochenen Schreibweisen für Dateinamen – erinnern Sie sich daran, das Dateinamen beispielsweise Case-Sensitive sind, »Verzeichnis« also nicht das gleiche wie »VERZEICHNIS« ist?

Listing 8.22 mkdir

```
user$ ls -aF
./    DateiA  backup_kap6.tex  kap08.tex
../   DateiB  cmds_to_add      kap6.tex
user$ mkdir Verzeichnis
./    DateiA  Verzeichnis/     cmds_to_add  kap6.tex
../   DateiB  backup_kap6.tex  kap08.tex
```

8.4.2 rmdir – Löschen von Verzeichnissen

Der Löschvorgang eines Verzeichnisses ist genauso simpel wie dessen Erstellung. Nur, dass hierfür nicht das Kommando `mkdir` (= make directory), sondern `rmdir` (= remove directory) verwendet wird. Die Syntax entspricht der von `mkdir`. Der Nachteil dieses Programmes ist jedoch der, dass keine rekursive Löschung eines Verzeichnisses möglich ist, das heißt, nur Verzeichnisse, welchen keine Dateien zugeordnet[8] sind, können gelöscht werden.

Listing 8.23 rmdir in Aktion

```
user$ ls -laF VerzeichnisA VerzeichnisB
VerzeichnisA:
total 8
drwxr-xr-x  2 swendzel users  4096 Oct 18 18:11 ./
drwxr-xr-x  5 swendzel users  4096 Oct 18 18:11 ../

VerzeichnisB:
total 12
drwxr-xr-x  2 swendzel users  4096 Oct 18 18:11 ./
drwxr-xr-x  5 swendzel users  4096 Oct 18 18:11 ../
-rw-r--r--  1 swendzel users     5 Oct 18 18:11 file
user$ rmdir VerzeichnisA
user$ rmdir VerzeichnisB
rmdir: 'VerzeichnisB': Directory not empty
```

rmdir -p

Beherbergt ein Verzeichnis jedoch nur eine einzige Hierachie von Subverzeichnissen ohne Subdateien, so kann mit dem `-p`-Parameter ein Löschvorgang des Verzeichnisses, inklusive der Subverzeichnisse, bewirkt wer-

8 Das Wort »zugeordnet« wurde bewusst gewählt, um nicht zu vergessen, dass Verzeichnisse die Dateien nicht wirklich enthalten, sondern nur Verweise auf deren I-Node Einträge beherbergen. Die Daten der Dateien selbst können sich am physikalischen Punkt X des Speichermediums befinden. Weiteres hierzu enthält Kapitel 4.

den. Würde also das Verzeichnis B ein Subverzeichnis SA und dieses wiederum ein Subverzeichnis SB »beinhalten«, so wäre das Ergebnis der folgenden beiden Aufrufe äquivalent:

Listing 8.24 Verzeichnis und Subverzeichnis löschen

```
user$ rmdir -p VerzeichnisB/SA/SB
user$ rmdir VerzeichnisB/SA/SB; \
rmdir VerzeichnisB/SA; rmdir VerzeichnisB
```

8.4.3 cp – Kopieren von Dateien

cp -r

Das Programm `cp` (copy) legt eine Kopie von einer, bereits im Dateisystem vorhandenen Datei an. Dabei kann es sich natürlich auch um ein Verzeichnis handeln. Die Syntax ist einfach und in der Form »`cp [Option] DateiA [DateiB] Ziel`« gehalten, wobei mehrere Dateien in das Ziel kopiert werden können. Ein rekursives Kopieren von Verzeichnissen ist über den `-r`-Parameter möglich.

Listing 8.25 Dateien und Verzeichnisse kopieren

```
user$ cp prog kopie_prog
user$ mkdir Verzeichnis
user$ cp prog kopie_prog Verzeichnis/
user$ cp -r Verzeichnis kopie_Verzeichnis
```

8.4.4 mv – Verschieben einer Datei

Mittels des `mv`–Kommandos werden Dateien »verschoben«. Nun ... nicht ganz. Eigentlich wird die Datei nur einer anderen Verzeichnisdatei zugeordnet, der physikalische Dateninhalt der Datei bleibt im Normalfall dort, wo er ist [9] – einzige Ausnahme: der Verschiebungsvorgang über verschiedene Dateisysteme.

`mv` funktioniert rekursiv, d. h. Verzeichnisse samt deren Inhalt können ohne weitere Parameter komplett verschoben werden. Parameter Nummer eins gibt die Quelle, Nummer zwei das Ziel an.

Aber es gibt noch eine Nutzungsmöglichkeit für `mv`: das Vergeben eines neuen Namens für eine bestehende Datei.

9 Dies ist nicht nur der Unterschied zwischen `cp` und `mv`, sondern auch der Grund dafür, dass ein `mv`-Aufruf viel schneller erledigt ist als ein Kopiervorgang.

Listing 8.26 Das mv-Kommando

```
user$ mv kopie_Verzeichnis /home/user/neu
user$ mv Datei Neuer_Name_der_Datei
```

8.4.5 rm – Löschen von Dateien

Das den Windows-Benutzern als `del` (delete) bekannte Kommando zum Löschen von Dateien, heißt unter Linux `rm` (remove) und kann eine ganze Menge toller Sachen. Grundlegend wird `rm` mit dem zu löschenden Dateinamen aufgerufen: `rm [Optionen] Dateiname`.

`rm` kann zunächst einmal, neben dem Löschen regulärer Dateien, auch das Löschen von Verzeichnissen bewerkstelligen. Hierzu wird der Parameter `-d` übergeben.

rm -r

Des Weiteren besteht die Möglichkeit zur Abfrage jedes Löschvorgangs über den `-i`-Parameter. Ein rekursives Löschen ist via des bereits recht bekannten `-r`-Parameters möglich.

Listing 8.27 Löschen mit Nachfrage

```
user$ rm -ri Verzeichnis
rm: descend into directory 'Verzeichnis'? y
rm: remove 'Verzeichnis/filea'? y
rm: remove 'Verzeichnis/fileb'? y
rm: remove 'Verzeichnis/filec'? y
rm: remove 'Verzeichnis'? y
```

Einige Befehle, wie auch `rm`, bieten einen Parameter zur Terminierung der Optionsliste: `-`. Dadurch können Sie Dateien mit einem Namen wie »*-k*« löschen: `rm - -k`.

8.4.6 touch – Setzen der Zugriffszeiten von Dateien

Mit dem `touch`-Kommando können die Zugriffszeiten, so genannte »Timestamps«, von Dateien angepasst werden. Genauer gesagt wird der letzte Zugriff auf die Datei sowie die letzte Änderung am Dateiinhalt auf einen neuen Zeitpunkt gesetzt.

Der Zeitpunkt des Zugriffs wird dabei in der Form »MMDDhhmm« übergeben. Der letzte Zugriff wird über den Parameter `-a` (last access), die Modifizierung via `-m` (last modification) gesetzt.

Listing 8.28 Setzen des »letzten Zugriffs« auf eine Datei

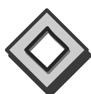
```
user$ ls -l file
-rw-r--r-- 1 swendzel users 10 Oct 21 21:42 file
user$ touch 10102246 file
user$ ls -l file
-rw-r--r-- 1 swendzel users 10 Oct 21 21:46 file
```

Ein nettes Feature von `touch` ist, dass ein noch nicht existierender neuer Dateiname mit leerem Inhalt erzeugt werden kann, indem man »`touch filename`« aufruft. Dies ist in der Systemadministration ein nettes Mittel, um »mal eben« eine Logdatei zu erstellen.

8.4.7 cut – Abschneiden von Dateiinhalten

Dateien (und auch die Standardeingabe) können über das Programm `cut` auf die eigenen Bedürfnisse zusammengeschnitten werden. Besonders in der Erstellung von Shell-Skripten spielen solche Funktionalitäten eine wichtige Rolle, da oftmals Datenströme angepasst werden müssen, um bestimmte Ausgaben zu erreichen. Wir werden später das Programm `awk` kennen lernen, mit welchem wir speziell solche Umformungen mit einer Syntax gestalten können. Sie ist der Programmiersprache C recht ähnlich.

Doch nun zurück zu `cut`. `cut` kann die Eingabedaten auf zwei Weisen »abschneiden«: mit Hilfe von Spalten (`-c`) und Feldern (`-f`). Dabei werden die Nummern der jeweiligen Einheit über Komma getrennt bzw. mit einem »–« verbunden.



Nun mag dies etwas verwirrend erscheinen, doch wozu gibt es denn Beispiele? Im Folgenden soll die Datei */etc/hosts*, welche aus drei Spalten besteht, die jeweils durch ein Leerzeichen voneinander getrennt sind[10], an unsere Bedürftnisse angepasst werden. Die erste Spalte gibt die IP-Adresse eines Computers, die zweite dessen vollen Domain-Namen und die dritte dessen bloßen Hostname an. Wir interessieren uns nun ausschließlich für die IP und den Hostname.

Da die einzelnen Felder durch ein Leerzeichen getrennt sind, geben wir dies via `-d`-Parameter als »Trennungszeichen« für die Spalten an. Da es sich beim Leerzeichen um ein nicht druckbares Zeichen handelt, »escapen« wir es, indem wir »\ « schreiben.

10 Dies muss nicht zwangsläufig so sein.

Listing 8.29 Beispielanwendung für `cut`

```
user$ cut -d\  -f 1,3 /etc/hosts
127.0.0.1 localhost
192.168.0.1 merkur
192.168.0.2 venus
192.168.0.3 erde
192.168.0.4 mars
192.168.0.5 jupiter
...
user$
```

8.4.8 paste – Zusammenfügen von Dateien

Nein, `paste` ist nicht – wie einige Leute glauben – das Gegenstück zum `cut`-Programm. `cut` schneidet die Teile, welche Sie benötigen, aus einem Text heraus. `paste` fügt jedoch keine Teile ein, sondern fügt ganze Dateien zusammen.

Die Zusammenfügung erfolgt zeilenweise über ein angebbares Trennungszeichen. Schauen wir uns einmal die Ausgabe des obigen `cut`-Beispiels an. Dort geben wir die IP–Adressen und die Hostnamen der Rechner im Netzwerk aus. In der folgenden fiktiven Situation gehen wir davon aus, dass die IP-Adressen in der Datei *IPAdressen* und die Hostnamen in der Datei *Hostnames* untergebracht sind. Wir möchten nun eine zeilenweise Zuordnung erstellen, wobei die einzelnen Spalten durch einen Doppelpunkt voneinander getrennt sein sollen.

Listing 8.30 Beispiel für das Zusammenfügen zweier Dateien

```
user$ paste -d : IPAdressen Hostnames
127.0.0.1 localhost
192.168.0.1 merkur
192.168.0.2 venus
192.168.0.3 erde
192.168.0.4 mars
192.168.0.5 jupiter
```

8.4.9 tac – den Dateiinhalt umdrehen

Es könnte vorkommen, dass eine Datei in einer Form vorliegt, welche umgedreht werden muss. Beispielsweise eine Tabelle, welche die Benut-

zerdaten von User-ID 1000 bis 10000 enthält, jedoch mit 10000 statt von 1000 beginnt. In diesem Fall hilft `tac` sehr einfach weiter.

Listing 8.31 `tac` dreht unsere hosts-Datei um

```
user$ tac /etc/hosts
192.168.0.5 jupiter.sun jupiter
192.168.0.4 mars.sun mars
192.168.0.3 erde.sun erde
192.168.0.2 venus.sun venus
192.168.0.1 merkur.sun merkur
127.0.0.1 localhost.sun localhost
```

8.4.10 nl – Zeilennummern für Dateien

Oft kommt es vor, dass der Quellcode eines Programmes – oder auch eines Shell-Skripts – im Usenet gepostet oder erklärt werden soll. An dieser Stelle (aber auch bei jeglicher Form von Tabelle und Plaintext-Datenbank) sind Zeilennummerierungen ein sehr hilfreiches Mittel, um dem Empfänger oder dem verarbeitenden Programm die Arbeit mit der Datei zu erleichtern.

An genau dieser Stelle setzt `nl` an und fügt der angegebenen Datei die Zeilennummern hinzu. Die Datei selbst wird dabei jedoch nicht manipuliert: Die Ausgabe erfolgt auf der Standardausgabe.

8.4.11 wc – Zählen von Zeichen, Zeilen und Wörtern

Mittels dieses Programmes können Sie ganz einfach die Wörter (-w) eines Textes (sofern dieser im ASCII-Format vorliegt), die Zeilen (-l) des neuesten Quellcodes oder dessen Zeichen (-c) zählen.

Listing 8.32 Zeilen der Buchdateien zählen

```
user$ wc -l kap??.tex
     677 kap01.tex
     597 kap02.tex
       7 kap03.tex
    1116 kap04.tex
     645 kap05.tex
    1044 kap06.tex
    1818 kap07.tex
     691 kap08.tex
      12 kap09.tex
```

```
  19 kap10.tex
 746 kap11.tex
  10 kap12.tex
  15 kap13.tex
  16 kap14.tex
  16 kap15.tex
7429 total
```

8.4.12 od – Dateien zur Zahlenbasis x ausgeben

Möchten Sie einmal eine Binärdatei verstehen? Nun, dazu genügt manchmal schon ein einfacher Hex-Editor oder das Dump-Komando `od`. Mittels dieses netten Programmes können Dateien in ASCII, dezimaler, oktaler und hexadezimaler Darstellungsweise ausgegeben werden.

Die oktale Schreibweise wird über den Parameter `-b`, die ASCII-Ausgabe via `-c` erzielt. Dabei wird jeweils ein Byte pro Spalte dargestellt.

Bei der Ausgabe in Hex-Form (`-x`) und dezimaler Form (`-d`) werden jeweils zwei Byte der Datei ausgegeben.

Listing 8.33 Hex-Dump des Kernels mit `od`

```
user$ od -x /vmlinuz
0000000 c0b8 8e07 b8d8 9000 c08e 00b9 2901 29f6
0000020 fcff a5f3 19ea 0000 bf90 3ff4 d88e d08e
0000040 fc89 e18e 78bb 1e00 c564 b137 5706 a5f3
0000060 1f5f 45c6 2404 8964 643f 478c 3002 30e4
0000100 cdd2 3113 b1d2 bb02 0200 02b4 f1a0 cd01
0000120 7313 500c 53e8 8901 e8e5 0158 eb58 bede
0000140 01d6 98ac d4a3 8101 dafe 7301 9114 d231
0000160 db30 3e8a 01f1 c7fe e7d0 01b8 cd02 7213
0000200 b8e1 9000 c08e 03b4 ff30 10cd 09b9 bb00
0000220 0007 dabd b801 1301 10cd 00b8 8e10 e8c0
...
```

8.4.13 Mehr oder weniger, das ist hier die Frage!

more und less

Nun möchten wir Ihnen zwei ganz besonders wichtige Programme in der Unix-Welt vorstellen: `more` und `less`. Beide Kommandos können Dateien auf dem Bildschirm seitenweise ausgeben. Dabei können einzelne Seiten mit den »Bild-Auf«- und »Bild-Ab«-Tasten gescrollt werden. Die Leertaste bewirkt das Weiterblättern einer Seite, die Cursor-Tasten werden

zum Scrollen von einzelnen Zeilen bzw. von Textblöcken (nach links bzw. rechts) verwendet.

Im Gegensatz zu `more` kann `less`, auch wenn der Text aus einer so genannten Pipe[11] stammt, den Buffer zurückscrollen. Die Handhabung der beiden Kommandos ist in der grundlegenden Benutzung identisch.

Listing 8.34 Funktionsweise von `more` und `less`

```
user$ more $BUCH/kap08.tex
\gpKapitel{Die Shell}

\begin{gpAnleser}
Herzlichen Glückwunsch. Sie sind nun am wohl
wichtigsten Kapitel des Buches
angelangt. Die Shell ist das A--und--O unter Linux
und wird Ihnen wohl auch
...
user$ cat $BUCH/kap08.tex | less
\gpKapitel{Die Shell}

\begin{gpAnleser}
Herzlichen Glückwunsch! Sie sind nun am wohl
wichtigsten Kapitel des Buches angelangt. Die
Shell ist das A--und--O unter Linux und wird
...
```

Es gibt Betriebssysteme, bei denen keine Unterscheidung zwischen diesen beiden Kommandos mehr getätigt wird, und die nur einen Verweis auf ein und denselben Inode-Eintrag kennen. Ein Beispiel dafür ist OpenBSD:

Listing 8.35 Ohne Worte

```
user$ ls -i /usr/bin/less
309244
user$ find /usr/bin -inum 309244 -print
/usr/bin/less
/usr/bin/more
/usr/bin/page
```

11 Pipes werden weiter unten behandelt.

8.4.14 head und tail

Zwei wichtige Programme sind `head` und `tail`. Ersteres zeigt den Kopf einer Datei, besser gesagt, die ersten Zeilen, Letzteres das Ende einer Datei auf dem Bildschirm an.

Die Anzahl der auszugebenen Zeilen wird via `-n` angegeben, wobei `n` kein Parametertyp selbst, sondern die Anzahl ist.

Listing 8.36 Die letzten und ersten Logeinträge

```
# Die letzten fünf Einträge der messages-Datei
# liefern uns aktuelle Meldungen:
user$ tail -5 /var/log/messages
Oct 25 16:28:50 laptop kernel: device lo left
promiscuous mode
Oct 25 16:29:25 laptop kernel: device lo entered
promiscuous mode
Oct 25 16:50:16 laptop -- MARK --
Oct 25 17:10:18 laptop -- MARK --
Oct 25 17:25:32 laptop kernel: device lo left
promiscuous mode

# Die ersten zwei Einträge liefern uns alte Daten
# vom September:
user$ head -2 /var/log/messages
Sep 19 15:41:31 laptop syslogd 1.4.1: restart.
Sep 19 15:41:32 laptop kernel: klogd 1.4.1, log
source = /proc/kmsg started.
```

`tail` kann der `-f`-Parameter übergeben werden. Dieser listet zunächst die letzten Zeilen der angegebenen Datei auf, wartet aber auf neue. Würde also `tail -f /var/log/messages` aufgerufen und nach fünf Minuten eine neue Logmeldung eingetragen werden, so würde diese automatisch ausgegeben werden.[12]

8.4.15 sort und uniq

Als Shell-Anwender kommt man recht oft – auch wenn man es aus der Windows-Welt kommend nicht so ganz glauben mag – in die Situation,

12 Das `tail` Programm wartet jeweils eine Sekunde, bis die nächste Prüfung auf neue Zeilen in der Zieldatei gestartet wird. Für schnelle Aktualisierungen der Ausgabe ist es also weniger geeignet.

bestimmte Zeilen von Dateien zu sortieren und redundante Datensätze zu entfernen.

Gegeben sei folgende Beispieldatei, welche zwei Spalten beinhaltet. Die erste gibt eine Nummer an, die dem Protokoll (Spalte 2) zugeordnet ist. Einige Dateneinträge sind redundant. Im Folgenden wollen wir diese Datensätze ordnen lassen.

Listing 8.37 Die Beispieldatei

```
001 ICMP
002 IGMP
089 OSPF
003 GGP
006 TCP
022 IDP
000 IP
012 PUP
017 UDP
022 IDP
255 RAW
```

`sort` hilft uns nun, diese Daten nach einer numerischen Reihenfolge korrekt zu sortieren (ohne anführende Nullen gibt es allerdings Probleme!). Was nun noch fehlt, ist, dass die redundanten Datensätze entfernt werden, besser gesagt, dass jeder von diesen Datensätzen nur einmal vorkommt. Dies wird mittels `uniq` bewerkstelligt.

Sofern man nicht mit Pipes arbeitet, werden die Aufrufe folgendermaßen abgewickelt: »`sort/uniq Dateiname`«

Listing 8.38 Die intelligente Lösung

```
user$ sort Beispieldatei | uniq
000 IP
001 ICMP
002 IGMP
003 GGP
006 TCP
012 PUP
017 UDP
022 IDP
089 OSPF
255 RAW
```

8.4.16 Dateien aufspalten

Ein nettes Tool, um große Dateien in kleinere Hälften aufzuteilen, ist `split`. Die Aufteilung erfolgt entweder durch Zeilen (-l) oder aber durch Bytes (-b).

Gehen wir einmal davon aus, dass eine Backup-Datei auf eine Diskette kopiert werden soll. Eine Diskette bietet 1440 kb Speicherplatz, wir benötigen von der Backup-Datei also 1440 kb große Teile, um eine effiziente Speichernutzung auf den »Backup-Medien« zu erzielen.

Die Datei selbst hat eine Größe *x*. `split` erstellt nun solange 1440kb große Dateien, bis die komplette Backup-Datei aufgeteilt ist.

Wenn die Datei nicht die Größe eines Vielfachen von 1440 kb an Größe hat, wird die letzte Datei natürlich nur die verbleibenden Restdaten, also das Ende der Backupdatei enthalten, und damit nicht den kompletten Speicherplatz belegen.

Bei der Aufteilung in Byte können folgende Suffixe verwendet werden: *b* für Blockeinheiten zu je 512 Byte, *k* für Kilobyte sowie *m* für Megabyte.

Listing 8.39 Aufteilen der Backup-Datei in 1440 k große Teile

```
user$ split -b 1440k backup.tgz
user$ ls xa?
xaa xab xac xad xae xaf xag
```

Die Dateien `xaa`, `xab`, `xac` ... sind die neu erstellten Teildateien. Doch wie fügt man sie »nu' wieder 'zam«? Im Folgenden ist eine sehr, sehr vereinfachte Variante gezeigt, bei der wir über `cat` eine Datei via Ausgabeumlenkung immer an das Ende der Backup-Datei anhängen.

Listing 8.40 Zusammenfügen der Dateien

```
user$ cat xaa >  backup.tgz
user$ cat xab >> backup.tgz
user$ cat xac >> backup.tgz
...
user$ cat xag >> backup.tgz
```

Listing 8.41 Die bessere Lösung

```
# Später in diesem Kapitel werden wir folgende
# Z-Shell Verbesserung dieser Lösung verstehen
```

```
rm backup.tgz
foreach file ( xa? )
   cat $file >>backup.tgz
end

# bzw.
foreach file ( xa[a-g] )
...

# oder...
foreach file ( xa* )
...
```

8.4.17 Zeichenvertauschung

Wir haben immer ein Dilemma mit heruntergeladenen, zu kategorisierenden Dateien. Die eigentlichen Dateien des Types befinden sich in Kleinbuchstaben auf der Platte, die heruntergeladenen haben jedoch oftmals Großbuchstaben im Namen.

Nun, dies ist eines von den Szenarien, an welchen `tr` Abhilfe schaffen kann. `tr` konvertiert ein Zeichen *x* in *y*, z.B. einen Großbuchstaben in einen kleinen Buchstaben, ein Leerzeichen in einen Unterstrich oder auch eine runde Klammerung in eine eckige.

Dabei werden die zu konvertierenden Zeichen in der Form »[alt] [neu]« übergeben, zu löschende Zeichen werden durch den Parameter `-d` bzw. `-delete`, zu komplementierende über `-c` bzw. `-complement` gekennzeichnet.

Listing 8.42 Das Kommando `tr`

```
user$ cat Datei
Da-tei-in-halt
user$ cat Datei | tr -d \-
Dateiinhalt
user$ cat Datei | tr a \?
D?-tei-in-h?lt
```

8.5 Linux und DOS

In diesem Abschnitt wollen wir uns mit zwei Thematiken beschäftigen. Zuerst mit den so genannten *mtools*, mit denen es möglich ist, eine nicht gemountete DOS-Diskette zu benutzen. Anschließend sehen wir uns zwei

Programme zur Zeichenkonvertierung vom DOS- zum Unix-Format und umgekehrt an.

Es ist möglich, dass diese Programme nicht auf Ihrem Rechner installiert sind. In diesem Fall müssen manuell Packages der jeweiligen Distribution nachinstalliert werden, um die Programme zu benutzen, oder eine Version aus dem Internet bezogen[13] werden.

8.5.1 Die mtools

Die *mtools* stellen eine Sammlung von Programmen zur Nutzung von DOS-Speichermedien (also z.B. eine Diskette, auf welcher sich ein FAT-Dateisystem befindet) dar. Entwickelt wurden diese von Emmet P. Gray. Das Besondere dabei ist, dass die Medien nicht ins Dateisystem eingehängt werden müssen.

Das *mtools*-Package enthält 13 Programme, welche jeweils einen Ersatz zum entsprechenden MS-DOS Programm darstellen.

Tool	Zweck	unter DOS
`mattrib`	setzt Dateiattribute	`attrib.exe`
`mcd`	wechselt in ein anderes Verzeichnis	`cd`
`mcopy`	Kopieren von Dateien	`copy`
`mdel`	Löschen von Dateien	`del`
`mdir`	gibt den Inhalt eines Verzeichnisses aus	`dir`
`mformat`	erstellt ein MS-DOS Dateisystem	`format.com`
`mlabel`	Speichermedium mit einem Namen (Label) versehen	`label.exe`
`mmd`	Erstellen eines Verzeichnisses	`md` und `mkdir`
`mrd`	Löschen eines Verzeichnisses	`rd` und `rmdir`
`mread`	Datei 1:1 kopieren (ohne Ersetzung von DOS- bzw. Linux-Zeichen wie bei `mcopy` von DOS zu Linux	
`mwrite`	Datei 1:1 kopieren (ohne Ersetzung von DOS- bzw. Linux-Zeichen wie bei `mcopy` von Linux zu DOS	
`mren`	Umbenennen bzw. Verschieben einer Datei	`rename`
`mtype`	Dateiinhalt ausgeben	`type`

Tabelle 8.2 Die *mtools*

13 Die *mtools* sind unter http://mtools.linux.lu/ zu finden.

Im Anhang ist auf Seite 489 eine Auflistung von DOS- und Linux-Befehlen zu finden. Dort sind die internen und externen DOS-Kommandos ihren äquivalenten Linux-Varianten zugeordnet.

8.5.2 dos2unix und unix2dos

Nein, es nicht alles damit gelöst, dass Sie eine FAT32-Partition unter Linux mounten können. Haben Sie schon mal eine Textdatei unter Linux erstellt und anschließend unter Windows in den Editor geladen? Sehen wir uns einmal an, was passiert . . .

Wir erstellen unter Linux eine Textdatei mit folgendem Inhalt:

Listing 8.43 Diese Datei muss als Testobjekt herhalten

```
user$ cat < EOF > file.txt
linux is like a wigwam:
no windows, no gates
and always an apache inside.
EOF
```

Laden wir diese Datei nun unter Windows in den Editor, sieht dies folgendermaßen aus:

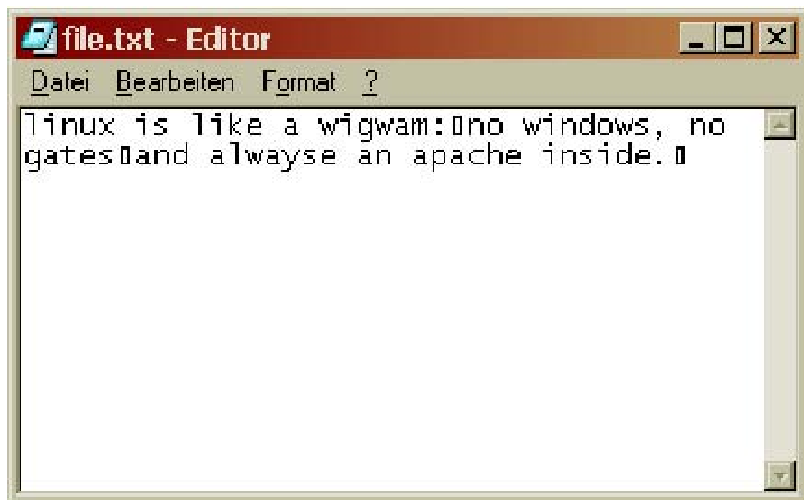


Abbildung 8.1 Unix-Textdateien im Notepad unter Windows

der Zeile ein Ende

Die eckigen Kästchen stellen das Zeilenende unter Linux dar. Windows

jedoch benutzt eine andere Kennzeichnung des Zeilenendes, nämlich »\r\n«, Linux hingegen verzichtet auf das »\r«. Daher sind ASCII-Dateien, welche zwischen den beiden Systemen ausgetauscht werden, nicht unbedingt ansehnlich.

Die beiden Kommandos `unix2dos` und `dos2unix` helfen bei der Konvertierung vom einem zum anderen System. Kopieren wir unsere *file.txt* doch einmal auf die Windows-Partition mit Hilfe dieser Werkzeuge. . .

Listing 8.44 So geht's richtig!

```
user$ unix2dos file.txt /win/file.txt
```

Wenn Sie sich die Datei nun im Notepad anschauen möchten, steht diesem Vorhaben nichts mehr im Wege.

8.6 Startskripte

Da wir uns in diesem Buch primär ans Beispiel der `bash` halten möchten, wird an dieser Stelle auch das Startskriptsystem dieser Shell beschrieben.

In den Startskripten werden globale und benutzerspezifische Initialisierungen vorgenommen. So werden Funktionen definiert, der ein oder andere Alias eingerichtet oder Variablen wie $PATH, der Programmsuchpfad, gesetzt.

Ist die `bash` als Login-Shell eingerichtet, so werden zunächst die Dateien */etc/profile* und – sofern vorhanden – die *.bash_profile* im Heimatverzeichnis des Benutzers ausgeführt. Anschließend werden die ebenfalls im Heimatverzeichnis liegenden Dateien *.bash_login* und *.profile* ausgeführt.

Die Datei */etc/profile* enthält globale Einstellungen. Dies ist praktisch, da bei jedem Login eines Benutzers eine vom Administrator vorgegebene Einstellung übernommen werden kann. So könnte zum Beispiel ein bestimmtes Kommando oder ein für alle Benutzer verwendbarer Alias eingerichtet werden.

selbst ist der User

Die anderen Dateien können vom Benutzer selbst eingerichtet und dazu verwendet werden, persönliche Einstellungen vorzunehmen.

Nachdem man sich aus der `bash` ausloggt hat, wird die Datei *.bash_logout* ausgeführt. In diese Datei kann man nützliche Funktionen, etwa zum Löschen temporärer Daten, einbauen.

.bashrc

Wird eine interaktive, nicht-Login-Shell gestartet liest die `bash` die Datei `.bashrc` im Heimatverzeichnis des Benutzers ein und führt sie aus.

Hier ist eine minimale *.bashrc*:

Listing 8.45 Beispiel für eine .bashrc Datei

```
# Setzung von einigen Variablen
export NULL=/dev/null
export MAINLOG=/var/log/messages
export TERM=xterm-color
export LC_ALL=de_DE

export NNTPSERVER='news.btx.dtag.de'
export EDITOR="vi"

export IRCNICK="cdp_xe"
...

# Alias-Definitionen
alias ls="/bin/ls -aF"
alias ll="ls -alhoF"
alias cl="cd ..;ls"
alias cll="cd ..;ll"
...

# Willkommens-Text fuer jede neue Shell
echo
echo "Welcome on `hostname`, $USER!"
echo
printf "%79s\n" "`uname -a`"
printf "%79s\n" "`uptime`"
...

# Der Willkommens-Spruch
/usr/games/fortune
```

8.7 Ein- und Ausgabeumlenkung

Ein sehr bedeutendes und simpel gehaltenes Feature der Unix-Shells ist die Ein- und Ausgabeumlenkung. Doch was hat es damit eigentlich auf sich?

Unter Linux verwenden Konsolenprogramme die Ausgabe von Text auf dem Terminal, um einem Anwender Informationen zu übermitteln. Die-

se Form der Ausgabe wird über die oben bereits angesprochene Standardausgabe (STDIN) geschickt. Man kann diese Standardausgabe jedoch auch umleiten, beispielsweise in ein anderes Programm (dies geht mit Hilfe von Pipes, die weiter unten besprochen werden) oder in eine Datei. Diese Umleitung der Ausgabe wird, wie Sie wohl bereits erahnen, als »Ausgabeumlenkung« bezeichnet und mit dem größer-als-Operator (>) realisiert.

Der Operator wird hinter das auszuführende Programm geschrieben und anschließend der Dateiname angegeben, in den die Ausgabe umgelenkt werden soll.

Listing 8.46 Ausgabeumlenkung in eine Datei

```
user$ ls -l
total 3800
-rw-r--r-- 1 swendzel wheel  379 Nov 1 1:34 Makefile
-rw-r--r-- 1 swendzel wheel 1051 Nov 2 0:56 anhang.aux
-rw-r--r-- 1 swendzel wheel 1979 Nov 1 0:53 anhang.tex
-rwx------ 1 swendzel wheel  283 Nov 1 1:13 backup
...
user$ ls > output
user$ head -4 output
total 3808
drwxr-xr-x  3 swendzel wheel 1536 Nov 30 7:48 ./
drwxr-xr-x 21 swendzel wheel  512 Nov 22 3:00 ../
-rw-r--r--  1 swendzel wheel  379 Nov 15 1:34 Makefile
```

Eingabeumlenkung

Das gleiche Prinzip verfolgt auch die Eingabeumlenkung mit dem Unterschied, dass hier natürlich das gegenteilige Verfahren genutzt wird: Der Inhalt einer Datei wird als Eingabe für ein Programm verwendet. D. h., die Eingabe wird nicht mehr manuell getätigt, kann in einer Datei dauerhaft gespeichert und immer wieder als Steuerung für ein Programm verwendet werden.

Ein gutes Beispiel für solch eine Anwendung ist das `mail`-Programm. Der Inhalt einer Datei kann so durch Eingabeumlenkung ganz schnell und einfach an den Mann gebracht werden.

Listing 8.47 Eingabeumlenkung

```
user$ mail -s Testmail swendzel@eygo.sun < output
```

8.7.1 Fehlerausgabe und Verknüpfung von Ausgaben

Nummerierung

Neben der Schreibweise »>Ausgabe« ist auch die Schreibweise »Nummer>Ausgabe« möglich, wobei Nummer die Nummer des Ausgabekanals angibt. Der Eingabekanal hat die Nummer 0, die Standardausgabe die 1 und die Standard-Fehlerausgabe die 2. Nur durch Angabe der Nummer kann demzufolge auch die Standard-Fehlerausgabe umgelenkt werden, da die Nummer der Shell mitteilt, was genau umgeleitet werden soll.

Listing 8.48 Fehler- und Standardausgabe umlenken

```
user$ ls /root 1>/dev/null 2> Fehler
user$ cat Fehler
ls: root: Permission denied
```

Verknüpfung der Kanäle

Die Umlenkung der Ausgabe kann auch verknüpft werden. Dies wird realisiert, indem man einem anderen Kanal das Ziel eines voher umgeleiteten Kanals zuweist. Die zugehörige Schreibweise zeigt das folgende Listing:

Listing 8.49 Sowohl Standard- als auch Fehlerausgabe ins Nirvana schicken

```
user$ ls /* > /dev/null 2>&1
user$
```

Die Ausgabeumlenkung kann übrigens zur gleichen Zeit wie die Eingabeumlenkung realisiert werden: `programm < Eingabe > Ausgabe`

8.7.2 Anhängen von Ausgaben

Zu den obigen Möglichkeiten kommt noch hinzu, dass eine Ausgabe an eine bereits vorhandene Datei oder eine Eingabe an bereits vorhandene Eingaben, angehängt werden kann. Dazu verwendet man den Umleitungsoperator einfach doppelt.

Listing 8.50 Umleitung hinzufügen

```
user$ echo "Das ist Zeile 1" > Output
user$ echo "Das ist noch eine Zeile" >> Output
user$ cat Output
Das ist Zeile 1
Das ist noch eine Zeile
```

8.7.3 Gruppierung der Umlenkung

Es ist möglich, eine Umlenkung mehrerer Programme zu gruppieren, das heißt, alle in dieser Gruppe enthaltenen Kommandos sind von der Umlenkung der Gruppe betroffen. Eine Gruppierung wird auf zwei Arten vorgenommen: mit normalen und mit geschweiften Klammern. Verwendet man normale Klammern, so wird für die Befehlsgruppe zusätzlich eine Sub-Shell (siehe Glossar) gestartet, die enthaltenen Anweisungen nehmen also weniger Einfluss auf die aktuelle Shell.

Listing 8.51 Gruppierung

```
user$ { ls -l; uptime } > Output
user$ tail -2 Output
zzz
 6:46PM up 3:46, 4 users,load averages:0.24,0.19,0.18
user$ ( rm ‘find / -name ’*.core’‘ ) 2> /dev/null &
```

8.8 Pipes

Pipes (|) sind mehr oder weniger mit dem Feature der Ausgabeumlenkung verwandt. Der Unterschied besteht darin, dass Pipes die Ausgabe eines Kommandos nicht in eine Datei, sondern in ein weiteres Kommando leiten. Dieses zweite (und gegebenenfalls auch dritte, vierte ...) Kommando wird die Ausgabe des ersten Programmes daher als Eingabe sehen. Setzen wir das oben bereits aufgeführte Mail-Beispiel doch einmal via Pipe um:

Listing 8.52 Mailen via Pipe

```
# Der alte Aufruf:
user$ mail -s Testmail swendzel@eygo.sun < output

# Hier mit Pipes:
user$ cat output | mail -s Testmail swendzel@eygo.sun

# Pipes kann man eigentlich für fast alles
# verwenden...
user$ wc -l kap??.tex | sort | \
awk '{
    print "Datei " $2 " hat momentan " $1 " Zeilen";
     }'
...
```

8.8.1 Tee kochen mit tee

Doch was ist, wenn man die Ausgabe einer Pipe nicht nur weiterleiten, sondern gleichzeitig umlenken möchte? Dafür gibt es das Programm `tee`. Man übergibt `tee` den Dateinamen, in den die Ausgabe (nur `stdin`, nicht `stderr` werden unterstützt) umgeleitet werden soll. Zugleich wird die Ausgabe jedoch auf dem Bildschirm (also `stdout`) ausgegeben und kann entweder in eine weitere Datei oder eine Pipe geleitet werden.

Listing 8.53 Anwendung von tee

```
user$ wc -l kap??.tex | sort | tee output.tex | \
./make_status
```

8.8.2 Named Pipes (FIFOs)

Named Pipes (so genannte FIFOs) erweitern die Fähigkeiten einer Pipe. Eine FIFO kann als Datei auf dem Dateisystem erzeugt werden (was mit dem Befehl `mkfifo` bewerkstelligt wird) und Daten eines Prozesses einlesen. Das Tolle daran ist, dass mehrere Prozesse diese FIFO verwenden können.

FIFOs arbeiten nach dem First-In-First-Out-Prinzip (daher der Name). Das bedeutet: Die Daten, die zuerst in einer FIFO abgelegt werden, werden auch zuerst wieder vom lesenden Prozess gelesen.

In der Regel benötigt man dieses Feature zwar nicht, aber wenn beispielsweise ein IRC-Client auf Shell-Ebene entwickelt wird, der Telnet als Sender und Empfänger der Daten verwenden soll, wäre es eventuell eine prima Lösung.

Listing 8.54 Erstellung und Verwendung einer FIFO

```
user$ mkfifo fifo

# Unter der Annahme, ein Daemonprozess zur
# Verarbeitung der FIFO--Einschriebe existiert und
# läuft, schreiben wir nun einen String in diese:
user$ echo Gleich sind Shellskripte an der Reihe.\
> fifo

# Sinnloser Weise könnte der Empfängerprozess den
# Inhalt der FIFO einfach nur ausgeben. Für einen
# Beispiel-Empfänger ist dieses Beispiel jedoch
# optimal...
```

```
user$ cat fifo
Gleich sind Shellskripte an der Reihe.
```

8.9 Grundlagen der Shell-Skript-Programmierung

Sofern Sie nach dem obigen Stoff eine kleine Lernpause benötigen, gönnen Sie sich diese. Das nun folgende Thema ist etwas schwieriger und sollte daher mit voller Aufmerksamkeit studiert werden. Die Shellskript-Programmierung ist in der UNIX-Welt ein definitiv wichtiges Mittel zur Administration von Systemen.

Zunächst werden wir uns mit den grundlegenden Dingen der Shellskripte beschäftigen. Anschließend werden Schleifen, Arrays und bedingte Anweisungen erläutert. Am Ende des Kapitels sollten Sie ohne größere Probleme ein Backup-Skript für Ihren Server erstellen können.[14]

8.9.1 Doch was ist ein Shellskript denn genau?

Ein Shellskript ist zunächst einmal eine Datei. In dieser Datei ist mindestens ein Shell-Kommando enthalten. Die in der Datei enthaltenen Kommandos werden der Reihe nach von der Shell ausgeführt.

Damit solch ein Shellskript überhaupt ausführbar wird, sollte man es mit den entsprechenden Rechten versehen. Als Programmierer benötigt man Lese-, Schreib- und Ausführ-Zugriff auf ein Skript. Die anderen Personen des Systems sollten nicht unbedingt Zugriff auf solche Skripte haben.

Listing 8.55 Setzung der Shellskript-Permissions

```
user$ chown user:users Skript.sh
user$ chmod 0700 Skript.sh
```

Ein Shellskript wird ausgeführt, indem man es wie ein normales Programm aufruft. Dabei ist zu beachten, dass das Skript in einem *bin*-Verzeichnis, wie etwa */usr/local/bin*, abzulegen ist und dieses Verzeichnis in der PATH-Variable enthalten ist. Natürlich kann ein Skript auch im Arbeitsverzeichnis ausgeführt werden. Dazu muss dieses allerdings explizit angegeben werden:

14 Ferner sollte nach einiger Zeit des hierauf aufbauenden Selbststudiums durchaus die Überwachung von Netzwerkrechnern oder die Auswertung des SMTP-Traffics eines Mailservers über ein Shellskript – eventuell auch mit einem CGI-Webinterface – möglich sein.

Listing 8.56 Ausführen eines Skriptes

```
user$ ./Skript.sh
--- Ausgaben des Skripts ---
```

8.9.2 Wie legt man los?

Als erstes benötigt man einen Editor[15] (siehe Kapitel 9). In diesen werden die Anweisungen des Skripts eingegeben und abgespeichert. Anschließend setzt man noch die obigen Zugriffsrechte und fertig ist das erste Shellskript.

Ein Shellskript beginnt ordnungsgemäß mit der folgenden Zeile:

Listing 8.57 Interpreterangabe

```
#!<Pfad zum Interpreter>

// Also zum Beispiel:
#!/bin/sh
```

Interpreter

Dies hat den einfachen Grund, dass der Kernel sich die ersten zwei Zeichen einer auszuführenden Datei anschaut. Handelt es sich dabei um die Zeichen »#!«, so bedeutet dies, dass für die Ausführung der Datei ein so genannter Interpreter benötigt wird. Ein Interpreter ist ein Programm, welches in der Lage ist, diese Anweisungen zu verstehen und zu verarbeiten. Im Falle der Shellskript-Programmierung mit der Bourne-Shell, wäre das beispielsweise die Bourne-Shell selbst.[16] Der Pfad dieser Shell ist */bin/sh*.

8.9.3 Das erste Shellskript

Es ist nun an der Zeit, die Theorie in die Paxis zu portieren. Unser erstes Shellskript wird nur eine einfache Meldung, nämlich »Hello World!«, auf dem Bildschirm ausgeben.

Listing 8.58 hello.sh

```
user$ cat hello.sh
#!/bin/sh

echo "Hello World!"
```

15 Zum Beispiel der vi-Editor, joe oder Emacs

16 Ein weiteres Beispiel wäre ein Perlskript. Der entsprechende Interpreter wäre in diesem Fall /usr/bin/perl.

```
user$ chmod +x hello.sh # Ausführ-Recht
user$ ./hello.sh
Hello World!
```

8.9.4 Kommentare

Wie in jeder guten Programmiersprache bietet auch die Shell eine Möglichkeit, Kommentare im Quelltext zu platzieren. Besonders in großen Skripten ist es manchmal sehr praktisch, einige Hinweise an sich selbst zu richten, falls man später wieder einen Durchblick in den Code bekommen möchte.

In der Shellskript-Programmierung werden Kommentare mit einer Raute (#) eingeleitet. Alles, was hinter solch einer Raute steht, wird bis zum Ende der Zeile als Kommentar gewertet, daher kann auch direkt hinter eine Anweisung ein Kommentar eingefügt werden:

Listing 8.59 hello2.sh

```
#!/bin/sh

# Dieses Skript gibt die Meldung "Hello World!" aus
echo "Hello World!" # Meldung ausgeben
echo                # Eine freie Zeile und Ende
```

8.9.5 Variablen

Variablen sind elementare Bauteile der Programmierung. In einer Variablen – besser gesagt, deren Speicherplatz – wird ein Wert (etwa ein String oder das Geburtsdatum der Katze) abgelegt. Indem man nun auf diese Variable zugreift, kann der Wert abgefragt oder verändert werden. Der Zugriff und die Identifikation einer Variablen wird über den ihr vergebenen Namen erreicht.

Eine Variable wird in den meisten Fällen als eine Art Platzhalter für dynamische Kommandoaufrufe verwendet. Dabei wird einem Kommando jeweils ein Wert x, der von einer beliebigen Situation abhängig gemacht wurde, übergeben. Ein Beispiel dafür wäre die Übergabe von Parametern an das `echo`-Kommando oder die Abfrage der Konfigurationsdaten einer Schnittstelle X mit `ifconfig`.

Über den Zuweisungsoperator (=) wird ein Wert X der Variablen `var` zugewiesen. Der spätere Zugriff auf eine Variable erfolgt mit dem Dollarzeichen ($).

Wird ein Variablenname vergeben, spricht man von der *Deklarierung* einer Variablen. In der Regel wird einer Variablen jedoch zusätzlich bei der Deklaration ein Wert zugewiesen. Diese Zuweisung eines Wertes bezeichnet man als *Initialisierung*.

Im folgenden Listing wird die Variable VARA mit dem Wert »Katze« initialisiert und ihr Wert mit dem `echo`-Kommando ausgegeben.

Listing 8.60 Verwendung einer Variablen

```
VARA="Katze" # Wert zuweisen

echo VARA     # Falsch! Gibt nicht den Wert von
              # VARA, sondern den String "VARA" aus

echo $VARA  # Wert korrekt ausgegeben
```

Variablen bekommen in der Shell-Programmierung normalerweise Namen, die nur aus Großbuchstaben bestehen. Absolut notwendig ist dies jedoch nicht.

Gültigkeit einer Variable

Eine Variable ist zunächst nur in ihrer Shell (bzw. im Shellskript) gültig. Allerdings ist es möglich, Variablen an Kind-Prozesse der Shell zu vererben. Dies ist wichtig, damit Programme über Variablen konfiguriert werden können.

Der IRC-Client `BitchX` etwa, wird auf diese Weise konfiguriert. Seine Variablen werden beim Login oder dem Shellstart initialisiert und dann *exportiert*, damit seitens des Clients ein Zugriff auf sie möglich wird.

Um eine Variable den Kindprozessen nun zugänglich zu machen, muss diese mit dem `export`-Befehl *exportiert* werden: `export IRCHOST=»eygo.sun«`

8.9.6 Rechnen mit Variablen

Nein, auf trockene Mathematik wird hier verzichtet. Es geht an dieser Stelle nur um die grundlegenden Rechenoperationen in der Shell, anhand von Variablen. In Sprachen wie awk, Perl oder C können Variablen einfach in der Form `var = 4858 + 4854828 * PI` oder `var += 484` verrechnet werden und mit `var++/-` in- beziehungsweise dekrementiert werden.

Leider bietet die Shell diese Möglichkeit nicht auf einem so direkten Wege. Die Shell sieht die Variableninhalte als Strings, daher benötigt man externe Programme, um Variablen zu verrechnen.

Wir verwenden an dieser Stelle das `expr`-Programm. Dieses kann diverse Ausdrücke verrechnen und lässt sich simpel bedienen.

Addition und Subtraktion

Will man lediglich addieren (oder eine Subtraktion durchführen), geht dies folgendermaßen: Die Rechnung wird als Parameter dem `expr`-Kommando übergeben und dieses gibt das Resultat aus:

Listing 8.61 Einfach expr

```
user$ expr 384 + 484 + 12
880
user$ expr 1000 - 500 + 499
999
```

Weitere Möglichkeiten

`expr` bietet noch einige weitere Möglichkeiten zur Berechnung von Werten. An dieser Stelle wollen wir uns jedoch im einfachsten Bereich bewegen. Neben der Multiplikation und Division steht auch noch die Modulo-Operation zur Verfügung, die den Rest einer Division ausgibt.

Listing 8.62 Einfach expr

```
user$ expr 17 \* 1948 \/ 14
2365
user$ expr 10 \% 7
3
user$ expr 10 \* \( 17 \* 1948 \/ 14 \)
23650
```

Einige Rechenoperatoren sollten mit einer Escapesequenz (etwa *) versehen werden, damit `expr` sie nicht missinterpretiert.

Wertzuweisung an Variablen

Die Werte, die vom `expr`-Kommando ausgegeben werden, können via Kommandosubstitution an die Variable zugewiesen werden:

Listing 8.63 Kommandosubstitution mit Variablen

```
export VALUE=‘expr 10 \* 10‘
```

8.9.7 Benutzereingaben für Variablen

Mit dem Kommando `read` ist es möglich, die Eingaben eines Benutzers in einer Variablen zu speichern. Dies kann äußert hilfreich sein, wenn es beispielsweise darum geht, ein Installationsskript zu schreiben, welches die Einstellungen des Benutzers abfragt (etwa das Zielverzeichnis zur Installation von PHP-Dateien).

Listing 8.64 Das read-Kommando

```
#!/bin/bash

echo "Wohin installieren?"
read ZIEL
echo "Installiere in $ZIEL..."
cp -r * $ZIEL
```

8.9.8 Arrays

Arrays können Sie als eine Ansammlung von mehreren Werten in einer Variablen, die über einen Index ansprechbar sind, verstehen. Die einzelnen Werte sind in den *Elementen* eines Arrays gespeichert. Arrays werden folgendermaßen initialisiert:

```
Name=(Element1 Element2 ... ElementN)
```

Der Zugriff erfolgt mit der Syntax `$Name[Element]`:

Listing 8.65 Arrays initialisieren

```
user$ array=(Katze Hund Maus)
user$ echo ${array[2]}
```

> Die Elemente beginnen mit der Zahl 0, d. h., das 0-te Element ist das erste zugewiesene.

Neue Elemente hinzufügen

Neue Array-Elemente werden einfach mit dem Syntax `array[Element]=Wert` zugewiesen.

Listing 8.66

```
user$ array[3]=Polarfuchs
user$ echo ${array[*]}
Katze Hund Maus Polarfuchs
```

Array-Länge

Die Länge eines Arrays bekommt man mit `echo ${#Name[*]` heraus, wobei der Stern-Operator für alle Elemente steht. Möchte man also alle Elemente des Arrays ausgeben, so kann man dies mit `echo ${array[*]}` bewerkstelligen.

8.9.9 Kommandosubstitution und Schreibweisen

Variablen können interessanterweise in verschiedene Schreibweisen eingebettet werden. Betrachten wir einmal die Verwendung einer Variablen im `echo`-Kommando. Zunächst geben wir eine Variable, eingebettet in Text aus, wobei wir Anführungszeichen (double Quotes) verwenden:

Listing 8.67 Anführungszeichen

```
user$ CMD="uptime"
user$ echo "Mein Lieblingskommando ist $CMD."
Mein Lieblingskommando ist uptime.
```

Hochkomma

Bei der Verwendung von Anführungszeichen wird der Wert der Variable also ausgegeben. Doch wie geht man vor, wenn man den Variablennamen selbst ausgeben möchte? Ganz einfach: Man verwendet Hochkommas (Quotes). Werte, welche in Hochkommas stehen, werden von der Shell nicht weiter betrachtet.

Listing 8.68 Hochkommas

```
user$ echo 'Mein Lieblingskommando ist $CMD.'
Mein Lieblingskommando ist $CMD.
```

Backquotes

Eine weitere, besonders wichtige, Schreibweise ist die bereits erwähnte Kommandosubstitution. Dabei werden Backquotes verwendet und das in ihnen eingeschlossene Kommando ausgeführt. Die vom Kommando produzierte Ausgabe wird anstelle des Wertes bzw. des Variablennamens ausgegeben:

Listing 8.69 Kommandosubstitution

```
user$ echo "Mein Lieblingskommando ist `$CMD`"
```

```
Mein Lieblingskommando ist  6:44PM up 3:20, 4 users,
load averages: 0.26, 0.14, 0.11
user$ echo ''pwd''
'pwd'
user$ echo 'pwd'
/home/swendzel/projekte/LINUX_BUCH
```

8.9.10 Argumentübergabe

Konsolenapplikationen werden oftmals mit bestimmten Werten – besser gesagt, Argumenten, etwa einem Dateinamen – beim Aufruf versorgt, um Ihnen mitzuteilen, was zu tun ist bzw. womit das, was zu tun ist, getan werden soll. Dem Programm `cat` wird zum Beispiel der Dateiname der Datei übergeben, von der es den Inhalt auslesen und auf der Standardausgabe ausgeben soll.

Shellskripte bieten die gleiche Möglichkeit zur Verarbeitung von Argumenten. Wird ein Shellskript gestartet, so kann man auf die einzelnen, übergebenen Argumente mit Hilfe der dafür vorgesehenen Variablen zugreifen. Die Variablen sind durchnummeriert, $0 gibt das erste Argument, also den Namen des Skripts, $1 das erste, dem Skript übergebene Argument, $2 das zweite Argument an.

Leider ist dies etwas verwirrend. Um Ihnen das Verständnis für diese wichtige Funktionalität besser zu vermitteln, haben wir ein kleines Shellskript geschrieben, das seine eigenen Argumente ausgibt.

Listing 8.70 Argumentübergabe

```
user$ cat argvscript.sh
#!/bin/sh

echo $0 $1 $2 $3
```

Doch blicken wir, bevor wir dieses Skript verbessern, etwas tiefer in diese Thematik. Es gibt nämlich noch einige weitere Informationen, die Ihnen bei der Argumentübergabe zur Verfügung stehen.

Das obige Skript gibt nur die ersten drei, dem Skript übergebenen Argumente aus. Das Problem dabei ist, dass man eventuell gar keins oder zwei oder vielleicht auch vier übergibt und das Skript nicht dynamisch darauf reagieren kann. Es gibt eine Variable mit der Bezeichnung $#. $# beinhaltet die Anzahl der übergebenen Parameter. Des Weiteren steht $* bzw.

Argument Anzahl

$@ zur Verfügung. Beide Variablen geben alle übergebenen Argumente aus.

Generell kann über die Schreibweise $N nur auf die Parameter 0–9 zugegriffen werden. Die `bash`-Shell, welche wir in diesem Buch von nun an verwenden werden, bietet die Möglichkeit, mittels der Schreibweise ${N} auf das N-te übergebene Argument zuzugreifen.

Eine neue Version des obigen Skripts gibt uns mittels einer Schleife[17] alle übergebenen Argumente dynamisch aus.

Listing 8.71 argvscript.sh etwas dynamischer

```
user$ cat argvscript2.sh
#!/bin/bash

for ARGUMENTNAME in $@
do
     echo $ARGUMENTNAME
done
user$ ./argvscript2.sh 3949 Merkur Venus Erde Mars
3949
Merkur
Venus
Erde
Mars
```

8.9.11 Funktionen

In der Programmierung dienen Funktionen primär dem Zweck, mehrere Anweisungen in einen Block zu gliedern. Ein Programm, welches beispielsweise zwei Berechnungen, eine Summierung und eine Division durchführt, könnte eine Funktion, in der der Code zur Summierung und eine, in der der Code zur Division untergebracht ist, beinhalten.

Die meisten Programmiersprachen verfügen über eine Funktion zur Ausgabe von Text auf dem Bildschirm, die mit `print`, `puts`, `printf` oder ähnlichen Namen betitelt wurden. Diese Funktionen beinhalten jeweils den entsprechenden Code, stellen im Programmquelltext jedoch nur eine einzelne Zeile pro Aufruf dar. Und damit sind wir bei einem weiteren

17 Schleifen werden erst später in diesem Buch behandelt. Generell dienen Schleifen zur mehrmaligen Abarbeitung von Befehlen. In diesem Beispiel bedeutet dies, dass für jedes Argument in der Variable $@ einmal die Anweisungen der for-Schleife durchlaufen werden.

Vorteil der Funktionen: Sie können mehrmals aufgerufen werden, ohne den Funktionscode erneut zu implementieren.

Eine Funktion wird mit dem Schlüsselwort `function` eingeleitet und hat folgende Syntax:

Listing 8.72 Funktionssyntax

```
function Funktionsname
{
    # Codebereich
}

auch folgende Syntax ist möglich:

Funktionsname()
{
    # Codebereich
}
```

Eine Funktion wäre damit definiert und implementiert. Doch der Code würde, wenn er so in einer Datei abgelegt werden würde, nicht ausgeführt. Dazu bedarf es des Aufrufs einer solchen Funktion. Betrachten wir einmal folgendes Skript:

Listing 8.73 Func.sh

```
#!/bin/bash

function PrintUsers
{
   who | awk '{ print $1 }' | uniq
}

echo "Momentan am System angemeldet:"
PrintUsers

sleep 180
echo "Und 3 Min. später:"
PrintUsers
```

Das Skript gibt die Benutzer des Systems aus, die momentan eingeloggt sind. Der Funktionscode wird zu dem Zeitpunkt ausgeführt, an dem `PrintUsers` aufgerufen wird.

Funktionen können in ihrem Codebereich auch andere Funktionen aufrufen. Man spricht in diesem Fall von Funktionsschachtelung.

Parameterübergabe und Funktionsvariablen

Wie einem Shellskript selbst, so können intern auch den Funktionen ein oder mehrere Parameter übergeben werden, wobei die Abfrage dieser genauso wie im Hauptbereich des Skripts erfolgt (siehe Argumentübergabe).

Was bei Programmiersprachen wie C selbstverständlich ist, nämlich das in einem Funktionsbereich (besser gesagt, in einem Gültigkeitsbereich) deklarierte Variablen nur *lokal* gültig sind, muss der Shell erst gesagt werden. Eine *lokale* Variable bedeutet, dass die Variable nicht für den Rest des Shellskripts zugänglich ist, sondern nur innerhalb der Funktion gültig ist. Nachdem die Funktion beendet wird, wird auch der Speicher der Variablen wieder gelöscht.

Um dies zu erreichen wird das Kommando `local` verwendet: `local var1 var2 varN`.

globale Variablen

Möchten Sie allerdings die Variable auch an Subshells vererben, die Variable also zu einer *globalen* machen, kann dies mit dem `export`-Kommando erfolgen. Die Syntax entspricht der von `local`:

Listing 8.74 Lokale und globale Variablen

```
user$ cat func2.sh
#!/bin/bash

function PrintSysInfo
{
      export OS=‘uname‘
      local CPU=‘uname -p‘
      HOST=‘uname -n‘

      (echo $OS)  # Geht, da global.
      echo $CPU
}

PrintSysInfo

echo $HOST
echo $CPU    # Falsch, da local!
```

```
user$ ./func2.sh
Linux
AMD Duron(tm) ("AuthenticAMD 686-class)
eygo.sun
```

8.9.12 Bedingungen

Bedingte Anweisungen stellen einen elementaren Grundstein der Programmierung dar. Mittels dieser Bedingungen können Werte abgefragt und dementsprechend darauf regiert werden.

Ein einfaches Beispiel dafür wäre Folgendes: Ein Benutzer soll in einem Programm angeben, ob er einen Ausdruck seines Dokuments haben möchte. Das Programm muss nun die Eingaben des Benutzers prüfen und diese, zum Beispiel in einer Variablen, speichern. Enthält die Variable den Wert »Ja«, erfolgt der Ausdruck, andernfalls wird ganz einfach davon abgesehen.

if

Zur Formulierung von bedingten Anweisungen verwendet man in der Regel die `if`-Anweisung (es gibt weitere Möglichkeiten, die wir weiter unten behandeln möchten). Sie hat folgende Syntax:

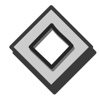
Listing 8.75 Die if-Anweisung

```
if [ BedingungA ] && [ BedingungB ]
then
       Anweisungen
elif [ BedingungA ]
then
       Anweisungen
else
       Anweisungen
fi
```

Die `if`-Anweisung in Zeile 1 legt die Grundbedingung fest. Ist diese erfüllt, werden die Anweisungen, die hinter dem `then`-Schlüsselwort stehen, ausgeführt. Ist die Bedingung jedoch nicht erfüllt, wird gepfüft, ob die – sofern vorhanden – nächste `elif`-Bedingung erfüllt ist und deren Anweisungen ausgeführt. Ist auch diese nicht erfüllt, wird zur nächsten `elif`-Anweisung gesprungen, bis es keine mehr gibt. Existiert noch eine `else`-Anweisung wird diese nur ausgeführt, falls alle anderen Bedingungen nicht zutrafen.

Bedingungen können auch verknüpft werden, ein && beispielsweise bedeutet, dass sowohl Bedingung 1 als auch Bedingung 2 erfüllt sein müssen, damit die Anweisungen ausgeführt werden. Des Weiteren gibt es noch die Negierung (!) – die Bedingung ist also erfüllt, wenn ihr Inhalt nicht erfüllt wurde. Außerdem verfügbar: Das »oder« (||), bei dem nur eine der verknüpften Bedingungen erfüllt sein muss.

Es gibt eine ganze Menge an Möglichkeiten, Bedingungen zu formulieren, die um einiges über String-Vergleiche hinausgehen. So kann man prüfen, ob eine Datei existiert oder eine bestimmte Datei ein Verzeichnis ist. Da dieses Sub-Kapitel lediglich zur Einführung in die Shellskript-Programmierung dient, werden wir Sie nicht auch noch damit quälen.



Das folgende Skript prüft, ob ein und, wenn ja, welcher Wert als Erster Parameter übergeben wurde und führt eine bedingte Anweisung aus. Der Parameter `-z` in einer Bedingung prüft, ob das angegebene Element leer ist. Beachten Sie bitte, dass Variablen, deren Inhalt in Form eines Strings verglichen werden soll, in Anführungszeichen geschrieben werden müssen.

Listing 8.76 Bedingte Anweisungen

```
#!/bin/sh

if [ -z $1 ]
then
      echo "Erforderlicher Parameter fehlt!"
elif [ "$1" = "backup" ]
then
      echo "Erstelle Backup."
      ...
elif [ "$1" = "restore" ]
then
      echo "Spiele Sicherungsdaten wieder ein."
      ...
else
      echo "Unbekannter Parameter."
fi
```

Vergleichen von Zahlen

Oftmals hat man es mit Zahlen zu tun. Diese werden allerdings auf spezielle Arten verglichen. Die folgende Tabelle listet die Vergleichsmöglichkeiten auf.

Vergleich	Beschreibung
$a -eq $b	Die verglichenen Werte sind gleich (equal).
$a -ne $b	Die Werte sind ungleich (not equal).
$a -lt $b	$a ist kleiner als $b.
$a -le $b	$a ist kleiner-gleich $b.
$a -gt $b	$a ist größer als $b.
$a -ge $b	$a ist größer-gleich als $b.

Tabelle 8.3 Bedingungen für Zahlen

Würden wir das obige Beispiel anhand von Zahlenvergleichen realisieren, wäre zum Beispiel folgende Bedingung denkbar:

Listing 8.77

```
elif [ $1 -eq 1 ]
then
      echo "Erstelle Backup."
      ...
...
```

Returncodes

Via Returncodes ist es möglich, den Erfolg eines ausgeführten Programmes zu überprüfen. Führt ein Shellskript beispielsweise ein Backup vom Verzeichnis */export/home/nobody* durch und das Verzeichnis existiert nicht, sollte das Programm einen entsprechenden Fehlercode zurückgeben. In der Regel steht eine »0« für eine erfolgreiche Programmausführung, eine »1« für eine fehlerhafte.

Fehlercodes werden immer in der Shell-Variable $? hinterlegt.

Listing 8.78 Prüfen des Rückgabewertes

```
user$ ls Datei
Datei
user$ echo $?
```

```
0
user$ ls DateiABCD
ls: DateiABCD: No such file or directory
user$ echo $?
1
user$ ls /root 2>/dev/null 1>output
user$ if [ $? -eq 1 ]; then
then> echo "Programmausführung fehlerhaft, breche
dquote> Skript ab."
then> fi
Programmausführung fehlerhaft, breche Skript ab.
```

8.9.13 Bedingte Anweisungen – Teil 2

Beschäftigen wir uns noch mit einer weiteren Möglichkeit, bedingte Anweisungen zu programmieren, nämlich der `case`-Anweisung. Der Nachteil dieser Anweisung ist, dass sie nur eine Variable verarbeitet. Darin liegt aber auch wiederum ihr Vorteil gegenüber der Anweisung `if`. Möchten Sie nämlich dort eine Variable auf N Werte überprüfen, so müssen jeweils `elsif`-Bedingungen formuliert werden. `case` bietet dafür eine kürzere, bessere Schreibweise.

Die Syntax von `case` ist folgendermaßen aufgebaut:

Listing 8.79 case-Syntax

```
case "$VARIABLE" in
  WertA)
      Anweisungen für A
      ;;
  WertB)
      Anweisungen für B
      ;;
  *)
      Anweisungen
      ;;
esac
```

Die Werte »WertA« und »WertB« werden für die Variable $VARIABLE überprüft. Beinhaltet $VARIABLE einen dieser Werte, werden die entsprechenden Anweisungen ausgeführt. Die beiden Semikolons beenden den Anweisungsbereich.

Der letzte Werte-Test (*) ist sozusagen das `else` der `if`-Anweisung, der immer dann greift, wenn obige Werte nicht mit dem Variablenwert übereinstimmten.

Die `case`-Anweisung wird in der Regel für die Erstellung von Runnlevel-Skripten (besonders unter Solaris) verwendet. Man übergibt dem Skript dabei ein Argument $1, welches Werte wie »start« oder »stop« enthält, um einen Dienst im Runlevel N (oder manuell) zu starten bzw. zu beenden.

Skripte, die Kommandos im Hintergrund ausführen, können Sie auf diese Art und Weise wundervoll stoppen und starten. Das folgende Skript wendet dieses Verfahren an, um den `syslog`-Dämon zu starten bzw. anzuhalten.

Listing 8.80 Start-/Stop-Skript

```
#!/bin/bash

case "$1" in
   start)
        echo "starte syslogd"
        /usr/sbin/syslogd
        ;;
   hup)
        echo "rekonfiguriere syslogd"
        kill -HUP `ps auxw | grep syslogd | \
                   awk '{print $2}'`
        ;;
   stop)
        echo "stoppe syslogd"
        kill `ps auxw | grep syslogd | \
                   awk '{print $2}'`
        ;;
   *)
        echo "Kommando unbekannt!"
        ;;
```

8.9.14 Die while-Schleife

Sie verfügen nun über das, zum Verständnis von Schleifen notwendige Grundwissen. Einer Schleife wird eine Bedingung übergeben. Ist diese

erfüllt, werden ihre Anweisungen solange ausgeführt, bis diese Bedingung nicht mehr erfüllt ist.

Eine Bedingung ist sowohl in einer Schleife als auch in einer normalen `case`-Verzweigung oder `if`-Anweisung immer wahr, wenn sie den Wert »1« ergibt. Probieren Sie einmal folgende Anweisung aus:

Listing 8.81

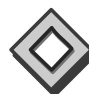
```
user$ if [ 1 ]
then
     echo true
fi
```

Die Syntax dieser Schleife ist im folgenden Listing abgebildet. Die Anweisungen werden mit dem `do`-Schlüsselwort eingeleitet und mit `done` beendet.

Listing 8.82

```
while [ Bedingung ]
do
    Anweisung A
    Anweisung B
    ...
done
```

Doch in der Praxis lernt man bekanntlich am besten. Im Folgenden wird eine *Endlosschleife* erstellt, die alle 30 Minuten die verfügbare Kapazität der Datenträger überprüft. Sinkt die verbliebene freie Kapazität auf unter 5 % (das `df`-Kommando gibt in diesem Fall einen Wert von über 95 % für die genutzte Kapazität an), wird eine Meldung ausgegeben. Das Prozentzeichen wird mit dem `sed`-Programm herausgefiltet. `awk` und `sed` lernen Sie zu einem etwas späterem Zeitpunkt in diesem Kapitel kennen.



Listing 8.83 Endlosschleife

```
#!/bin/bash

while [ 1 ]
do
     df -h | grep -v '[Uu]se' | sed s/\%// | \
     awk '{
```

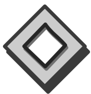
```
            if($5>95)
                 print $1 " is nearly full. ("$5"%)"
        }'
        sleep 1800
done
```

8.9.15 Die for-Schleife

Die `for`-Schleife bietet gegenüber der `while`-Schleife einen Vorteil: Sie kann eine Liste von Dateien durcharbeiten. D. h., die Schleife wird für jede angegebene Datei einmal durchlaufen. Dabei wird einer Variablen für jeden Schleifendurchlauf der Wert eines Dateinamens zugewiesen, mit dem Sie innerhalb des Anweisungsblocks arbeiten können.

Listing 8.84 Syntax der for-Schleife

```
for VAR in Dateien
do
        AnweisungA
        AnweisungB
        ...
done
```



Stellen Sie sich einmal vor, dass alle Benutzerverzeichnisse archiviert werden sollen. Die Benutzerverzeichnisse der Benutzer, die mit dem Buchstaben »A« beginnen, sind im Verzeichnis */home/a/*, die mit »B« beginnen, in */home/b/* untergebracht usw. Als Archivmedium stehen ZIP-Laufwerke mit einer Kapazität von 250 MB zur Verfügung und die Benutzer-Quotas beschränken sich pro Buchstabenverzeichnis auf genau diese Kapazität.

Mit der `for`-Schleife kann nun ein simples Skript entwickelt werden, welches jeweils ein Verzeichnis auf ein ZIP-Medium sichert und den Admin danach auffordert, die nächste ZIP-Disk einzulegen.:

Listing 8.85 ZIP-Backup

```
#!/bin/bash

ZIPDEVICE=/dev/sd0d
TARGET=/mnt/zip

for DIR in /home/*
do
```

```
    if [ -d $DIR ] # Ist es ein Verzeichnis?
    then
        mount -t vfat $ZIPDEVICE $TARGET
        if [ $? -eq 0 ]
        then
            cp -r $DIR $TARGET
            umount $TARGET
            echo "Bitte nächstes Medium einlegen."
        else
            echo "Mountvorgang schlug fehl!"
            exit
        fi
    fi
done
```

Im Optimalfall sollte noch die verfügbare Kapazität auf dem Medium überprüft und mit dem `tar`-Kommando eine Komprimierung erzielt werden. Doch hätten wir dies gezeigt, wäre wohl das eigentliche Ziel, die Demonstration der `for`-Schleife untergegangen.

8.9.16 Menüs bilden mit select

In der Regel wird eine Benutzereingabe mit mehreren Möglichkeiten folgendermaßen getätigt: Auf dem Bildschirm werden die Möglichkeiten ausgegeben, der Benutzer gibt die gewünschte Auswahl ein. Dabei gibt er beispielsweise einen String »backup« ein, um das Dateisystem zu archivieren. Doch was passiert, wenn er »Backup«, »Bäckup« oder »Bkup« eingibt? Das Skript wird eine Fehlermeldung liefern und sich beenden bzw. eine unvorhergesehene Aktion durchführen und eventuell Schaden am System anrichten.

Mit Hilfe der `select`-Anweisung können Menüs besser aufgebaut werden. Den Auswahlmöglichkeiten sind dabei Zahlen zugeordnet, die der Benutzer auswählen kann. Dabei kann seitens des Anwenders wesentlich weniger schief laufen. Zudem wird bei unbekannter Eingabe einfach eine erneute Eingabeaufforderung ausgegeben, so dass kein zufälliger oder falscher Code ausgeführt wird.

Syntax bereits bekannt

Die Syntax ist der `for`-Schleife ähnlich. In der Eingabevariablen wird der Wert der gewählten Auswahl hinterlegt.

Listing 8.86 select-Syntax

```
select AUSWAHL in MöglichkeitA MöglichkeitB ...
```

```
do
     Anweisung(en)
done
```

Zum allgemeinen Verständnis gibt es natürlich wie immer ein Beispiel, welches Ihnen die eigentliche Funktionsweise etwas offener darlegen sollte, als es die bloße Syntax könnte. Das folgende Skript bietet die Auswahl einer Logdatei im Verzeichnis */var/log* an. Die ausgewählte Datei wird mit dem `tail`-Programm überwacht. Das Skript wird durch die Tastenkombination `STRG+C` bzw. via `exit`-Funktion beendet.

Listing 8.87 Logwatcher mit select und tail

```
#!/bin/bash

cd /var/log

select AUSWAHL in authlog daemon messages maillog \
                  ftpd named.log exit
do
    if [ "$AUSWAHL" == "exit" ]; then
        exit
    fi
    tail -f  $AUSWAHL
done
```

Bei der Eingabeaufforderung für die Menüauswahl wird der Prompt `PS3` angezeigt. Dieser ist, wie bereits weiter oben beschrieben, eine Variable. Wird diese mit einem anderen Wert versehen, können Sie Ihre eigene Aufforderungsmeldung definieren: `export PS3='input>'`.

8.9.17 Das Auge isst mit: der Schreibstil

Im Laufe der Zeit werden Ihre Shellskripte etwas größer und komplexer ausfallen. Aus viel Code schlau zu werden, ist nicht schwer, aus unübersichtlichem Code schlau zu werden, jedoch schon. Daher empfiehlt es sich, wie in jeder Programmiersprache, auch in den Shellskripten bestimmte Formen einzuhalten. Dies wird Ihnen selbst und natürlich all denen, an die Sie Ihren Code weitergeben möchten, das Lesen erleichtern.

Dazu gehört, dass nicht alle Kommandos aneinander gereit, sondern in Zeilen separiert werden. Doch was noch viel wichtiger ist: Die Anweisungsblöcke bedingter Anweisungen und Schleifen sollten jeweils übereinander stehen.

Betrachten Sie einmal folgendes Beispiel:

Listing 8.88 Unübersichtlicher Code

```
   for NAME in ~/Haustiere/*
do mail -s "$NAME" < text;
if [ "$NAME" == "Felix" ]
do
       echo "Felix gefunden."; done
       done
```

Aus Erfahrung ist bekannt, dass ein Großteil der Anwender tatsächlich solchen Skriptcode schreiben. Eine bessere Lösung wäre doch, die Kommandos zu separieren und die Schleifen hierachisch anzuordnen:

Listing 8.89 Übersichtlicher Code

```
for NAME in ~/Haustiere/*
do
      # Dieser Bereich ist der for-Schleife
      # untergeordnet.
      mail -s "$NAME" < text;
      if [ "$NAME" == "Felix" ]
      do
           # Der if Anweisung untergeordnet.
           echo "Felix gefunden."
      done
done
```

Sofern Sie nicht gerade ein Buch schreiben und die Textabschnitte die Erklärungen des Codes enthalten, sind Kommentare eine gern gesehene Erweiterung.

8.10 Reguläre Ausdrücke: awk und sed

Bisher wissen Sie schon eine ganze Menge. Kommen wir nun zum letzten großen Sub-Kapitel im Shell-Bereich: den regulären Ausdrücken (engl. regular expressions) und damit zu den Programmen `awk` und `sed`.

Wir werden vorwiegend grundlegende Formen der regulären Ausdrücke behandeln und weniger oft benutzte auslassen, um im Rahmen des Buches zu bleiben und Sie nicht unnötig zu »quälen«.

Zum Ende dieses Sub-Kapitels werden wir uns noch mit dem Tool `grep` beschäftigen. Doch nun zurück zur Einleitung.

Wozu das Ganze?

Reguläre Ausdrücke bieten die Möglichkeit, Kommandozeilenaufrufe und Programme in sehr kurzer Weise zu gestalten, wenn es um die Verarbeitung von Text-Streams geht. Dies wiederum geht leider auf Kosten der Lesbarkeit, doch lassen sie sich mit etwas Übung bis zu einem gewissen Grad problemlos entziffern.

Um sich ein etwas genaueres Bild dieser Thematik zu machen, sei hier ein Beispiel gegeben.

Wir befinden uns in einem Verzeichnis X, dessen Inhalt aus drei Dateien besteht: *Baum, baum* und *Haus*.

Da reguläre Ausdrücke auch bei Programmen wie `ls` angewandt werden können, werden wir sie nun mit Hilfe dieser obigen Dateien auflisten.

Zuerst sollen alle Dateien aufgelistet werden, die mit einem beliebigen Zeichen beginnen und auf »aum« enden. Hierzu verwenden wir den regulären Ausdruck »?«:

Listing 8.90 Der Zeichen-Operator

```
user$ ls ?aum
Baum baum
```

Als Nächstes sollen alle Dateien, welche mit einem kleinen oder großen B beginnen, aufgelistet werden, anschließend alle Dateien, die mit einem »m« enden:

Listing 8.91 Einer für alles

```
user$ ls [bB]aum
Baum baum
user$ ls *m
Baum baum
user$ ls *a*
Baum baum Haus
```

Wir werden diese Operatoren im Verlaufe des Kapitels noch näher kennen lernen, hier zunächst erst einmal (der Übersicht halber) eine Tabelle.

Zeichen	Beschreibung
.	Beliebiges Zeichen
*	Beliebige Anzahl von beliebigen Zeichen (d. h., auch kein Zeichen entspricht diesem Metazeichen)
+	Belibige Anzahl des Zeichens (mind. einmal)
?	Einzelnes oder kein Vorkommen eines Zeichens
[..]	Alle in der Klammerung eingeschlossenen Zeichen können an dieser Stelle vorkommen. Beispielsweise »[0-9]«.
[ˆ..]	Eine Negierung. Kommt ein Zeichen nicht vor, ist die Bedingung erfüllt.
ˆ	Zeilenanfang
$	Zeilenende
x	Das Zeichen kommt x mal vor.
x,	Das Zeichen kommt x oder mehrmals vor.
x,y	Das Zeichen kommt x bis y mal vor.

Tabelle 8.4 Reguläre Ausdrücke

8.10.1 awk – Basics und reguläre Ausdrücke

awk ist eine Skriptsprache zur Verarbeitung von ASCII-Text. Sie wurde nach ihren Entwicklern Aho, Kernighan und Weinberger benannt und im Laufe der Jahre zu einem populären Werkzeug der Anwender und Administratoren.

Der Grund dafür ist unter anderem der, dass *awk* sehr einfach zu erlernen ist, da es nur recht wenige Befehle gibt. Außerdem ist die Syntax an die Sprache C angelehnt und daher schon vielen Nutzern vertraut.

awk kann über Skripte oder direkt über die Kommandozeile benutzt werden, wobei jeweils das Programm `awk` bzw. `gawk` für diese Zwecke verwendet wird. `awk` ist das eigentliche, auf jedem Unix-System vorhandene Grundprogramm, besser gesagt, der Interpreter. `gawk` ist die GNU-Version und auf jedem Linux-System verfügbar. Es gibt außerdem noch die `nawk`-Variante, welche zum Beispiel auf OpenBSD-Systemen zum Einsatz kommt.

awk starten

`awk` wird wie auch die Implementierungen `nawk` und `gawk` ganz einfach über die Kommandozeile gestartet. Die Befehle zur Verarbeitung des Textes werden entweder in Hochkommas (Schift+Raute) oder in einer Datei abgelegt und als zweiter Parameter bzw. dritter Parameter bei Optionen übergeben. Danach folgt optional eine Datei, welche die zu verarbeitenden Daten enthält, welche mittels Pipe übergeben werden können.

Listing 8.92 So starten Sie awk

```
user$ awk '{print $1}' DateiA DateiB ... DateiN
user$ cat Datei | awk '{print $1}'
user$ awk -f skript.awk Datei
user$ cat Datei | awk -f skript.awk
```

8.10.2 Arbeitsweise von awk

Das Programm besteht aus drei dieser Teile: dem BEGIN-Teil, dem Hauptteil und dem END-Teil. Nach dem Aufruf wird zunächst einmal der eventuell vorhandene »BEGIN«-Teil des Programmcodes, der zur Initialisierung verwendet wird, abgearbeitet.

Anschließend wird jede einzelne Zeile des zu verarbeitenden Textes separat verarbeitet, was im Hauptteil geschieht. Der Hauptteil enthält also den Code für alle Anweisungen, welche mit den Zeilen durchgeführt werden sollen, und wird für jede Zeile komplett neu ausgeführt. Bei großen Dateien ist es daher recht sinnvoll, auf zu rechenaufwändige Anweisungen zu verzichten und einen effizienten Code zu schreiben.

Nachdem die Eingabedatei komplett verarbeitet wurde, wird (sofern implementiert) der Code im »END«-Teil des Skripts ausgeführt.

Hier ein Beispiel für den Aufbau eines awk-Skriptes:

Listing 8.93 Aufbau eines Skriptes mit awk

```
BEGIN {
    print "Monatsabrechnung"
}

{
    print $1 "+" $2 "=" $1+$2
}
```

```
END {
   print "Geschafft."
}
```

Trennung von Kommandos: In *awk* werden einzelne Kommandos durch den Trennungsoperator, ein Semmikolon (;) oder eine neue Zeile separiert: `print $1; variable=1`.

8.10.3 Reguläre Ausdrücke anwenden

Eine sehr nette Fähigkeit von *awk* liegt darin, als Filter für Muster zu dienen. Damit weist es ähnliche Funktionalitäten wir `grep` auf, welches wir später kennen lernen werden. Und was verwendet man dazu? Richtig! Reguläre Ausdrücke.

Listing 8.94 Aufruf von awk mit einem Muster

```
user$ cat file
Steffen, Friedrichshafen
Tobias, Ettenbeuren
Johannes, Karlsruhe
user$ awk '/u/' testfile
Tobias, Ettenbeuren
```

Einfache Strings

Einfache Strings können durch die Angabe des Strings selbst gefiltert werden. Diese werden in die Hochkommas und zwei Slashes geschrieben:

Listing 8.95 Filtern auf Zeichensalat

```
user$ awk '/on/' Zweigstellen
Bonn
London
user$ awk '/S/' Zweigstellen
Salzburg
Stockholm
```

Der Punkt-Operator

Der Punkt-Operator steht, wie Sie bereits wissen, für ein beliebiges Zeichen an einem einzigen Platz. Man kann ihn mitten in Strings einbauen, um zum Beispiel sowohl große als auch kleine Buchstaben zu erwischen. Eine ebenfalls praktische Anwendung wäre die Namensfindung – es könn-

te vorkommen, dass der Name einer Person entweder mit »c« oder »k« geschrieben wird.

Listing 8.96 Der Punkt-Operator

```
user$ awk '/K.ln/' Zweigstellen
Köln
```

Der Additions-Operator

Es ist keine Addition im eigentlichen Sinne; es ist nur der langweilige Operator, dessen Bedingung nur erfüllt ist, sofern mindestens ein Mal das vor ihm geschriebene Zeichen auftritt.

Listing 8.97 Mindestens ein Mal muss das n vorkommen

```
user$ awk '/n+/' Zweigstellen
Bonn
München
London
Bern
Köln
```

Die Zeichenvorgabe

Es ist möglich, eine bestimmte Zeichenvorgabe zu setzen. Eines dieser von Ihnen vorgegebenen Zeichen, muss dann an der entsprechenden Stelle im String vorkommen. Die ausgewählten Zeichen werden dabei in eckige Klammern eingebettet: »[abc]«. Außerdem können einige Abkürzungen wie »`a-z`« oder »`0-9`« verwendet werden. Einzelne Zeichengruppen werden mittels Komma getrennt.

Listing 8.98 Diese Zeichen dürfen alle vorkommen

```
user$ awk '/M?nch[a-z,0-9][nNzkvps]/' Zweigstellen
München
```

Negierte Zeichenvorgabe

Das Gegenteil zur obigen Zeichenvorgabe ist die negierte Zeichenvorgabe. Die dabei angegebene Menge von Zeichen darf an der entsprechenden Stelle nicht vorkommen, damit die Bedingung erfüllt ist.

Listing 8.99 Negierte Zeichenvorgabe

```
user$ awk '/M?nch[^e][^bn]/' Zweigstellen
```

```
user$ awk '/M?nch[^X][^bY]/' Zweigstellen
München
```

Diese Negierung kann nur auf Mengen und nicht direkt auf Einzelzeichen, d. h. ohne eckige Klammerung, angewandt werden. Später werden wir sehen, dass man auf diese Art und Weise den Anfang einer Zeile beschreibt.

Zeilenanfang und -ende

Oftmals sortiert man Datensätze nach dem Anfang bzw. dem Ende einer Zeile – ein gutes Beispiel hierfür wäre die Aussortierung der Logdatei-Einträge des Tages x. Die regulären Ausdrücke stellen uns hierfür zwei Zeichen zur Verfügung: ^ und $, wobei der Zeilenanfang durch Ersteres angegeben wird und als so genanntes »XOR« (exclusive or) bezeichnet wird, welches in der Digitaltechnik Verwendung findet. Das Dollarzeichen sollte jedem bekannt sein und wird in `awk` (ungeachtet der Bedeutung in regulären Ausdrücken) für die Kennzeichnung eines Variablenzugriffs verwendet[18].

Listing 8.100 Filtern nach Zeilenanfang und -ende

```
user$ awk '/^B/' Zweigstellen
Bonn
Bern
user$ awk '/n$/' Zweigstellen
Bonn
München
London
Bern
Köln
```

8.10.4 awk – etwas detailierter

Nein, wir möchten an dieser Stelle keine 200 Seiten *awk*-Theorie vermitteln, sondern uns auf wenige Seiten beschränken, um Ihnen in recht kurzer Form die Grundzüge dieser Sprache zu erklären, welche zum Teil bereits weiter oben vermittelt wurden.

awk bietet die Möglichkeit, auf einige wichtige Variablen zuzugreifen, die für die korrekte Ausführung des Programmcodes wichtig erscheinen. Sehen wir uns diese einmal an ...

18 Dies muss nicht zwangsläufig so sein, macht aber einen guten Programmierstil aus.

- »$1, $2, . . . $N«

  Diese Variablen geben die Werte der Spalten des Eingabestreams an. $1 ist dabei die erste Spalte, $2 die zweite usw.

- »$0«

  In dieser Variablen ist die komplette Zeile, welche sich in der aktuellen Verarbeitung befindet, abgelegt. Damit könnten Sie sich auch ein eigenes `cat`–Programm basteln: `print $0`

- »ARGC« und »ARGV«

  Wie bei jedem Betriebssystem, welches über ein CLI (Command Line Interface) verfügt, gibt es Programme (und Skripte), denen Parameter (etwa das zu erstellende Verzeichnis) übergeben werden. In »ARGV«, einem so genannten Array (wir werden uns noch genauer mit Arrays auseinandersetzen), werden diese Argumente gespeichert, »ARGC« gibt lediglich deren Anzahl an.

- »CONVFMT«

  Diese Variable gibt das Konvertierungsformat für Zahlen in Strings an und ist für uns in dieser Einführung nicht weiter von Bedeutung. Die Variable »OFMT« gibt übrigens das Ausgabeformat von Zahlen an.

- »ENVIRON«

  Dieser Array speichert die Umgebungsvariablen, in denen der Aufruf des *awk*-Codes erfolgt, samt ihren Werten.

- »ERRNO«

  Tritt ein Fehler auf, speichert diese Variable dessen Wert. Fragt man diesen Wert nun über eine entsprechende Funktion ab, so wird eine – zumindest für den Entwickler – verständliche Fehlermeldung ausgegeben.

- »FIELDWIDTHS«

  Normalerweise gibt man in *awk* das Zeichen, welches die einzelnen Spalten der zu verarbeitenden Daten trennt, direkt an. Möchte man lieber fixe Spaltengrößen verwenden, so kann man diese in »FIELDWIDTHS« angeben.

- »FILENAME«

  Normalerweise gibt man den Namen der Eingabedatei (also der Datei, welche die zu verarbeitenden Daten enthält) direkt an, oder `cat`tet diese über eine Pipe an *awk*. Tut man dies jedoch nicht, wird standardmäßig die Standardeingabe der Shell als Datenquelle verwendet.

Ist »FILENAME« gesetzt, wird jedoch der ihr zugewiesene Dateiname als Datenquelle verwendet.

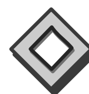
- »FNR«

  Die Zeilen der Eingabedatei sind durchnummeriert, und »FNR« enthält die aktuell verarbeitete Zeilennummer. Dies ist beispielsweise dann hilfreich, wenn es nötig ist, die ersten *N* Zeilen einer Quelldatei zu verarbeiten, oder wenn man eine Durchnummerierung seiner Datensätze erreichen möchte.

- »FS«

  »FS« steht für »field separator« und gibt das Zeichen an, welches zur Trennung der Spalten in der Quelldatei verwendet wird. »FS« wird direkt beim Aufruf von `awk` übergeben:

  In der Datei */etc/passwd* werden die Spalten mittels Doppelpunkten getrennt. Um diese Datei spaltenweise auszulesen, setzen wir den field separator auf das Doppelpunkt-Zeichen, was mit dem Parameter `-F` erledigt wird:



Listing 8.101 field separator

```
user$ awk -F: '{print "Benutzer/UID: "$1"/"$3}' \
/etc/passwd
Benutzer/UID: root/0
Benutzer/UID: bin/1
Benutzer/UID: daemon/2
Benutzer/UID: adm/3
Benutzer/UID: lp/4
Benutzer/UID: sync/5
...
```

  Die `print`-Funktion wird, wie zu sehen ist, für die Ausgabe von Text und Variablen-Werten verwendet, doch dazu später mehr.

- »NF«

  Diese Variable gibt die Anzahl der Felder in der Quelldatei an; »NF« steht für »number of fields«.

- »NR«

  Die Anzahl der Datensätze in der Quelldatei werden in »NR« abgelegt (Diese Ausdrucksweise ist eigentlich grundlegend falsch, da Variablen eigentlich keine Werte »enthalten«, sie stehen nur für einen Speicherbereich, in welchem diese abgelegt werden, bzw. verweisen (im Falle

eines Daten-Pointers) auf diesen Bereich.). >NR« steht für »number of records«[19]«.

- »OFS«

  Dies ist das Gegenstück zu »FS«, der Output-Separator, also das Trennungszeichen der ausgegebenen Daten.

- »ORS«

  Dies ist das Separierungszeichen für einzelne Datensätze bei der Ausgabe, »ORS« steht für »output record separator«.

- »RS«

  Natürlich gibt es auch zur obigen Output-Version ein Gegenstück: den Separator für Datensätze in der Eingabe.

- »RT«

  Der »record terminator« legt das Zeichen fest, welches das Ende der Datensätze angibt.

Zusätzliche Parameter beim Aufruf

Zu diesem Zeitpunkt kennen Sie nur die grundlegende Form eines *awk*-Aufrufs und den Parameter zur Trennung der einzelnen Spalten (field separator). Allerdings sind noch einige weitere wichtige Parameter vorhanden.

Mit `-f <Datei>` gibt man den Namen eines Skriptes an, welches den Programmcode enthält.

Vor dem Start können Sie übrigens auch Variablen erzeugen und diese mit neuen Werten beglücken, was mit »`awk -v Variable=Wert`« geschieht.

`-copyright` gibt die Kurzform der Lizenz, mit der *gawk* ausgeliefert wurde, aus.[20] Dreimal dürfen Sie raten, welche das ist. Der Kompatibilitätsmodus zum »Ur-*awk*« wird mit `-compat` und `-traditional` aktiviert und eine Anwendungshilfe mit `-usage` und `-help` ausgegeben.

8.10.5 awk und Variablen

Auch in *awk* gibt es Variablen. Und das Tolle daran: Deren Handhabung ist einfach. Werte werden direkt über den Zuweisungsoperator (also das Gleichheitszeichen) an die Variable gebracht. Inkrement- und Dekre-

---

19 In der Manpage ist die Rede von »records«, womit natürlich die Datensätze selbst gemeint sind.

20 Wer zusätzlich die verwendete *awk*-Version sehen möchte, sollte `-version` verwenden.

ment–Operatoren der Sprache »C« sind auch hier nutzbar, so kann der Wert via »variable++« um »1« erhöht werden und mit »variable-« um den selben Wert gesenkt werden.

Schreiben wir einmal ein kleines Testprogramm, welches die Zeilen der Eingabedatei, ähnlich wie das wc-Programm, zählt. Dazu nehmen wir eine Variable »Linecount«, welche die Zeilen zählt, setzen diese am Anfang (also im »BEGIN«-Bereich) auf den Wert »0« und zählen sie bei jedem Durchlauf des Hauptprogrammes eine Zeile weiter:

Listing 8.102 Der Zeilencounter in awk

```
BEGIN {
  # Im Initialisierungs-Teil weisen wir
  # der Variable "Linecount" den Wert
  # 0 zu, sie wird die Zeilen zählen.
     Linecount=0;
}
{ # Da die Hauptschleife bei jeder Zeile
  # durchlaufen wird, erhöhen wir ihren
  # Wert einfach jedesmal...
     Linecount++;
}
END {
  # ...und geben am Ende den Wert aus:
     print "Wert: " Linecount;
}
```

Ein Vergleich mit unserem Skript und dem wc-Kommando zeigt uns, dass alles funktioniert: Die Ergebnisse sind äquivalent.

Listing 8.103 Der Test

```
user$ awk -f script.awk file
Wert: 367
user$ wc -l file
    367 file
```

Kommentare werden in *awk* mit einer Raute (#) eingeleitet und gelten jeweils für die aktuelle Zeile ab dem Punkt, an dem sie gesetzt wurden.

Rechenoperationen

... gibt es natürlich auch in *awk*, und zwar bedeutend komfortabler, als in der bloßen Shellskript-Programmierung. Es gibt verschiedene Standardoperatoren, wie den Additions- (+) und Subtraktionsoperator (-), Multiplikation (*) und Division (/). Aber auch Kombinationen mit dem Zuweisungsoperator sind möglich. So kann aus »`Variable = Variable + 2;`« ein kurzes »`Variable += 2;`« oder aus »`Variable = Variable / 5;`« ein »`Variable /= 5;`« – wie in der Programmiersprache »C« – gemacht werden.

Listing 8.104 Rechenbeispiel für awk

```
BEGIN {
   Var = 1000;
   Var = 999; print Var;
   Var = Var * 2; print Var;
   Var += 10; print Var;
   Var *= 2; print Var;
}
```

Listing 8.105 Anwendung des Rechenbeispiels

```
user$ awk -f script.awk
999
1998
2008
4016
^D
```

Neben diesen Rechenoperationen können auch noch einige weitere, wie die Potenzierung, durchgeführt werden: »`var = 3 ^ 3`« weist »var« den Wert drei hoch drei, also 27 zu. Modulo-Operationen sind über den gleichnamigen Operator (%) ebenfalls möglich: »`var = 5 % 4`«.

Die Inkrementierung und Dekrementierung von Variablen kennen Sie bereits; was wir Ihnen jedoch noch verschwiegen haben, ist der Unterschied zwischen Pre- und Postinkrementierung beziehungsweise -dekrementierung.

Eine Pre-Verarbeitung hat die Syntax »`++/-variable`«, eine Post-Verarbeitung »`variable++/-`«. Der Unterschied dieser beiden Varianten ist, dass bei der Pre-Version eine Verarbeitung in einer Anweisung noch vor der eigentlichen Anweisung durchgeführt wird. Bei der Post-Variante ge-

schieht dies erst, nachdem solch eine Anweisung beendet wurde. Aber am besten lernt man ja bekanntlich an Beispielen.

Listing 8.106 Post- und Pre-In- und -Dekrementierung

```
user$ cat test.awk
BEGIN {
	test = 1;

	print test++;
	print test;
	print ++test;
}
user$ awk -f test.awk
1
2
3
```

8.10.6 Bedingte Anweisungen

Es stehen natürlich auch in *awk* einige Möglichkeiten zur Verfügung, bedingte Anweisungen mittels relationaler Ausdrücke zu formulieren. Im Rahmen dieser kleinen *awk*-Einführung sollen die `if`-Anweisung und die `for`-Schleife behandelt werden.

if

Mittels dieser Anweisungen werden die Werte von Variablen getestet. Dafür werden so genannte Bedingungen erstellt, die entweder erfüllt werden oder eben nicht. Wird eine Bedingung also (nicht) erfüllt, wird entweder die nachstehende Anweisung oder eine ganze Gruppe dieser ausgeführt, wobei in diesem Fall ein Bereich für die Anweisungen innerhalb von geschweiften Klammern geschaffen werden muss.

Bedingung	Beschreibung
Operator	Bedeutung
a < b	a muss kleiner als b sein.
a > b	a muss größer als b sein.
a <= b	a muss kleiner oder gleich b sein.
a => b	a muss größer oder gleich b sein.
a == b	a muss den gleichen Wert wie b haben.
a != b	a muss ungleich b sein.
a in b	b muss ein Array und a ein Element dieses Arrays sein.
a && b	sowohl a als auch b müssen erfüllt sein.
a & b	Diese Bedingung ist nur dann erfüllt (also >=1), wenn die binären Werte von a und b verknüpft mit einem logischen »und« mindestens 1 ergeben.
a || b	Ein logisches »oder«. Es ist zu beachten, dass es in *awk* kein einfaches oder-Zeichen (|) für diese Zwecke gibt, es würde als Pipe-Zeichen behandelt werden.
! a	Diese Bedingung ist erfüllt, wenn a nicht erfüllt ist.
x ? a : b	Wenn die Bedingung »x« erfüllt ist, wird a als Bedingungswert bezeichnet, andernfalls b.

Tabelle 8.5 Grundlegende und logische Bedingungen

Zum besseren Verständnis folgt nun ein Beispiel: Der Wert der Variablen »var« wird mit der `if`-Anweisung auf verschiedene Bedingungen geprüft. Ist die erste Bedingung nicht erfüllt, so wird mit `else if` eine weitere Verzweigung dieser Anweisung eröffnet. Durch das `else` wird also nur eine Prüfung vorgenommen, wenn die vorherige Bedingung nicht erfüllt ist.

Wenn auch die zweite Bedingung nicht erfüllt ist, »var« also den Wert »100« hat, tritt die letzte `else`-Anweisung in Kraft. Eine bloße `else`-Anweisung ohne zusätzliches `if` wird immer dann ausgeführt, wenn alle vorherigen Bedingungen unerfüllt sind.

Listing 8.107 So gehts

```
BEGIN {
      var=100;

      if(var > 100)
```

```
            print "var ist > 100"
    else if(var < 100)
            print "var ist < 100"
    else {
            print "var ist 100"
            print "Hier haben wir eine Gruppierung"
            print "von Anweisungen."
    }

    if(var <= 1000)
            print "var ist kleiner-gleich 1000"

    if(2 || 3){
            print "Bedingung erfüllt."
    }
}
```

for und while

Kommen wir nun zu zwei Schleifentypen, die uns in ähnlicher Form bereits aus der Shell-Programmierung bekannt sind und daher nicht nochmals in ihrer Funktionalität, dafür aber in ihrer Syntax erklärt werden.

In der `for`-Schleife gibt es in der Regel drei Parameter (eine Ausnahme stellt die `in`-Bedingung für Arrays dar). Das erste Argument legt den Wert einer Variablen fest, wird also zur Initialisierung verwendet. Der mittlere Teil enthält die Bedingung in der gleichen Form wie die `if`-Anweisung. Der letzte Parameter gibt die Anweisung an, welche bei jedem Schleifendurchlauf ausgeführt werden soll. Die einzelnen »Teile« werden dabei durch ein Semikolon voneinander getrennt.

Listing 8.108 Die for-Schleife

```
Syntax:      for( Init; Bedingung; Anweisung ) {
                     Anweisung1;
                     Anweisung2;
                     ...;
                     AnweisungN;
             }

Beispiel:    for(var=1; var<=5; var++)
                     print var;
```

Die `while`-Schleife läuft wie in der Shell so lange durch, bis die Bedingung nicht mehr erfüllt wird. Diese Bedingung ist das Einzige, was im Schleifenkopf angegeben werden muss. Die Anweisungen werden in *awk*, jedoch nicht in `do` und `done`, sondern wie wir es hier gewohnt sind, in geschweifte Klammern eingepackt.

Listing 8.109 Die while-Schleife

```
Syntax:      while( Bedingung ) {
                   Anweisung1;
                   Anweisung2;
                   ...
                   AnweisungN;
             }

Beispiel:    var=1;
             while(var<=5)
                   print var++;
```

8.10.7 Funktionen in awk

Kommen wir nun zum Thema Funktionen. Einerseits sollen Sie an dieser Stelle jene Funktionen kennen lernen, welche im Standard-awk bereits vorhanden sind, so genannte Builtin-Funktionen, andererseits lernen Sie Ihre eigenen zu erstellen.

Wie auch bei der Shell-Programmierung können Funktionen in *awk* Parameter übergeben werden, welche bei der Deklaration der Funktion – ähnlich wie in C – definiert werden. Zur Funktionsbestimmung wird das Schlüsselwort »`function`« verwendet. Anschließend folgt der Funktionsname und die in Klammern eingeschlossenen Parameter.

Listing 8.110 Funktionen in awk

```
Syntax:      function Name( Parameterliste ) {
               Anweisung1;
               Anweisung2;
               ...
               [return <Wert>] # Optional
             }

Beispiel:    function Sum(suma, sumb) {
               sum = suma + sumb;
               return sum;
```

```
}

BEGIN { print "Berechne Werte..." }

{   Summe+=Sum($1, $2); }

END { print "Endwert: " Summe; }
```

Anwendung:
```
user$ awk -f testb.awk sum
Berechne Werte...
Endwert: 6798
```

awk unterstützt die so genannten Rückgabewerte von Funktionen. Dabei werden die zurückgegebenen Funktionswerte in der Funktion selbst berechnet und an eine Variable oder an eine andere Funktion weitergegeben. Für diese Zwecke wird das Schlüsselwort »`return`« verwendet.

Die obige Funktion könnte mit ihrem Rückgabewert beispielsweise folgende Verwendung finden: `print Sum(4, 7);`. Dabei würde die Builtin-Funktion `print` den Rückgabewert als eigenen Parameter ansehen und ausgeben.

awk gibt übrigens auch einen Rückgabewert (wie es sich für gute Programme gehört) an die Shell zurück. Bei einem Rückgabewert von »0« war die Ausführung erfolgreich, andernfalls ist der Wert größer 0.

8.10.8 Builtin-Funktionen

Builtin-Funktionen stehen nicht nur in der Shell, sondern auch in *awk* zur Verfügung und stellen eine der wichtigsten Komponenten dieser Sprache dar. Zum Umfang gehören sowohl mathematische als auch Funktionen zur String-Verarbeitung. Im Rahmen dieser Einführung werden wir uns auf die wichtigsten Funktionen konzentrieren.

getline(...)

Die `getline`-Funktion liest eine Eingabe des Benutzers ein und speichert diese optional in einer gewünschten Variablen, andernfalls in $0.

print, sprintf und printf

Die Funktion `print` lernten Sie bereits kennen. Die *awk*-Version funktioniert im Prinzip so, wie die der Shell-Version. `printf` ist ebenfalls mit der Shell-Variante relativ gleichzusetzen.

`sprintf(f, e)` macht aus dem regulären Ausdruck »e« einen String mittels der Formatangabe »f« und gibt diesen zurück.

system(Kommando)

Diese Funktion ist in einer großen Anzahl von Programmiersprachen vorhanden und bietet die Möglichkeit, Shell-Befehle direkt vom Programm aus auszuführen, was aber nicht zum guten Programmierstil gehört. Das Kommando wird hierbei als String übergeben, sollte also in Anführungszeichen gesetzt werden.

fflush(Datei)

Diese Funktion leert die Puffer aller, im Skript verfügbaren bzw. angegebenen Deskriptoren.

cos(x), sin(x)

Diese beiden Funktionen geben den Kosinus bzw. Sinus von x zurück.

exp(x)

`exp()` ist die Exponentialfunktion von *awk*.

int(String)

Diese Funktion arbeitet ähnlich wie `atoi()` in C und gibt die Zahl aus dem String »String« zurück:

```
print int("70b");
```

log(x)

Hierbei handelt es sich um die Logarithmus-Funktion von *awk*.

rand() und srand(x)

Ja, *awk* ist fast so wie C, man kann zwar nicht so viel machen, aber es ist ein Grund mehr, diese Sprache aufgrund der Syntax zu lieben. Auch die Funktionen zur Generierung von Zufallszahlen haben die gleichen Namen. `srand` setzt dabei den Initialisierungswert für `rand()` mit x – wird kein Argument übergeben, wird die aktuelle Uhrzeit dafür verwendet. `rand()` selbst gibt einen Wert im Bereich von 0 < x < 1 zurück.

sqrt(x)

`sqrt()` gibt die Wurzel aus x zurück.

gsub(r, s [, t])

Der String »t« wird nach dem regulären Ausdruck, »r« durchsucht. Jeder gefundene Ausdruck wird mit »s« überschrieben: `gsub( »Saturn«, »Merkur«, $1); print $1);`

index(s, t)
`index()` gibt die Position des Strings »t« im String »s« zurück.

lenght([s])
Diese Funktion gibt die Länge von »s«, einem String (in Zeichen) zurück. Für den Fall, dass kein Argument gesetzt wurde, wird die Länge von $0 zurück gegeben.

match(s, r)
Gibt die Position des regulären Ausdrucks »r« im String »s« zurück.

split(s, a [, r])
Mit Hilfe des regulären Ausdrucks »r« wird der String »s« auf den Array »a« aufgeteilt. Jeweils ein passender Ausdruck wird als Element im Array abgelegt. Die Anzahl der Elemente wird zurückgegeben.

sub(r, s [, t])
Der String »t« wird nach dem regulären Ausdruck, »r« durchsucht. Der erste gefundene Ausdruck wird mit »s« überschrieben (also fast wie `gs-ub()`.

substr(s, i [, n])
Diese Funktion gibt einen Substring von »s« zurück. Dieser String fängt ab dem i-ten Zeichen an und endet entweder am String-Ende oder – sofern n gegeben ist – am n-ten Zeichen.

tolower(s) und toupper(s)
`tolower()` gibt den String »s« in einer Form zurück, bei dem alle Großbuchstaben des Parameters durch die entsprechenden kleinen (aus »S« wird »s« usw.) ersetzt werden. `toupper` ist für die gegenteilige Operation zuständig.

Zeitfunktionen

`awk` findet besonders im Bereich Logdatei-Parsing Verwendung. Dort ist es oftmals nötig, bestimmte Uhrzeiten und Daten herauszufiltern bzw. zu setzen. `gawk` bietet hierfür einige Erweiterungen an.

`systime()` gibt die Anzahl der seit dem 1. Januar 1970 verstrichenen Sekunden zurück. Diesen Rückgabewert bezeichnet man als »Timestamp«.

Die Funktion `strftime([Format [, Timestamp]])` wandelt diese Zahl in ein, von Menschen leicht interpretierbares Format um. Das Format kann dabei an die Bedürftnisse des Entwicklers angepasst werden und setzt sich aus verschiedenen Angaben der Form %X zusammen.

Format	Bewirkt
%y	Das Jahr im zweistelligen Format (03)
%Y	Das Jahr im vierstelligen Format (2003)
%m	Der Monat (01–12)
%h	Der Monat in dreistelliger Schreibweise (Jan, ... Nov, Dez)
%d	Der Tag des Monats (01–31)
%H	Die Stunde im 24-Stunden-Format (00–24)
%I	Die Stunde im 12-Stunden-Format (00–12)
%p	Die Minute (00–59)
%D	Das komplette Datum im Format »Monat/Tag/Jahr«

Tabelle 8.6 Möglichkeiten der Zeitformate

Da auch die `strftime`-Funktion einen Wert zurückgibt, kann dieser direkt mit `print` ausgegeben werden. Als Timestamp-Parameter verwenden wir wiederum den Rückgabewert der `systime`-Funktion:

Listing 8.111 Beispielanwendung für Zeitformatierung

```
user$ awk 'BEGIN {
        print "Heute ist der " strftime("%d.%m.%Y",
                                        systime());
}'
Heute ist der 19.11.2003
```

8.10.9 Arrays und String-Operationen

Auch Arrays finden in *awk* ein Zuhause. Dabei wird, wie bei Variablen, ein beliebiger Name vergeben, der das Array bezeichnet. Die Elemente werden in eckigen Klammern eingebettet: `array[7] = "Linux";`.

Assoziativität

`awk` verfügt über so genannte assoziative-Arrays, das bedeutet: Die Elemente können über String-Werte angesprochen werden, wie es auch in Perl möglich ist.

Des Weiteren kann der weiter oben angesprochene `in`-Operator dazu verwendet werden, die Arrays zu durchsuchen:

Listing 8.112 Arrays in awk

```
user$ awk 'BEGIN {
        array[0] = "InhaltA";
        array[1] = "InhaltB";
```

```
        print "-----"
        for(val in array)
                print "Element "val ": "array[val];
} END {
        array["Hund"] = "Wuff";
        array["Katze"] = "Miau";
        array["Schaf"] = "Maeh";

        print "-----"
        for(val in array)
                print val ": "array[val];

}'
-----
Element 0: InhaltA
Element 1: InhaltB
-----
Element Schaf: Maeh
Element Hund: Wuff
Element Katze: Miau
```

Elemente löschen

Einzelne Array-Elemente können über das Schlüsselwort »`delete`« gelöscht werden: »`delete array[element];`«. Probieren Sie den obigen Code doch einfach mal aus und fügen Sie zur Übung ein `delete array["Katze"];` vor der `for`-Schleife ein.

8.10.10 Was noch fehlt

Dies war eine kurze Einführung in *awk*. Wir haben Ihnen einige, durchaus grundlegende Eigenschaften dieser Sprache vorenthalten. Nun, andere Leute schreiben über diese Sprache ganze Bücher – doch genau dies möchten wir an dieser Stelle nicht. Dafür legen wir Ihnen [ASPT] als weiterführende Lektüre sowie [SAKG] als Nachschlagewerk ans Herz. Außerdem soll an dieser Stelle ein Verweis auf die exzellente Manpage (awk(1)) nicht fehlen.

8.10.11 sed

`sed` (*stream editor*) ist ein sehr mächtiges und auf allen UNIX-Systemen populäres Tool. Man verwendet *sed* ähnlich wie *awk*. Allerdings ist *sed* weniger eine Programmiersprache – wobei es allerdings auch möglich

ist, Skripte zu erstellen –, sondern viel mehr ein Programm zur Prüfung von Text-Streams auf reguläre Ausdrücke. Die spezielle Stärke liegt dabei darin, diese gefilterten Ausdrücke zu bearbeiten. So können Textbereiche beispielsweise ersetzt oder auch gelöscht werden.

Die Texte, welche *sed* verarbeiten soll, kommen *default* von `stdin`, können aber auch über eine extra angegebene Datei kommen, oder über eine Pipe bzw. Eingabeumlenkung übergeben werden.

Listing 8.113 Nutzung von sed

```
sed [Optionen] 'Adresse' [Dateiname]

user$ echo "Dies ist kein Satz." > austausch.txt
user$ sed s/kein/ein/ austausch.txt
Dies ist ein Satz.
```

Adressen

In der obigen Aufruf-Syntax findet sich der Parameter »Adresse«. Dieser dient zur Beschreibung des zu verarbeitenden Bereichs. Entweder wird nach einem String bzw. einem regulären Ausdruck gesucht oder aber die Zeilenadresse übergeben. Diese Adressierung wird in Hochkommas eingebettet und ist mittels Leerzeichen von der »Aktion« getrennt. Bei einer »Aktion« handelt es sich um eine Anweisung zur Bearbeitung der Zeilen, die auf die Adresse zutreffen.

Hier einige Beispiele:

- `sed '3 p' file.txt`

  Eine einzige Zahl adressiert eine einzige Zeile. In diesem Fall wird nur die dritte Zeile der Datei 'file.txt' verarbeitet. Die Aktions-Parameter werden separat besprochen.
- `sed '3-5 p' file.txt`

  Dieses Beispiel adressiert die Zeilen 3 bis 5.
- `sed '$ p' file.txt`

  Durch das Dollarzeichen ($) wird die letzte Zeile adressiert.
- `sed '/string/ p' file.txt`

  Hier werden alle Zeilen, die den String »string« enthalten, adressiert.
- `sed '/^#/d' /etc/rc`

  Filtert alle auskommentierten Zeilen mit Hilfe des regulären Ausdrucks heraus, wobei nur Zeilen als Kommentar gewertet werden, die mit einer Raute beginnen.

Befehle für sed

Nun fehlen eigentlich nur noch zwei Dinge, um die Grundlagen von *sed* zu erläutern: die Aktionsbefehle und die Optionen. Die Aktionen werden, wie bereits oben gezeigt, durch Einzelbuchstaben dargestellt, die entweder vor oder nach dem Suchmuster geschrieben werden.[21]

- **d:** `sed '50 d' file`

  d löscht die Zeilen, die der Adressierung entsprechen. Im obigen Fall wird die fünfzigste Zeile von *file* entfernt.

- **p:** `sed '1-20 p' file`

  Die Aktion **p** gibt die adressierten Zeilen auf dem Bildschirm aus.

- **s:** `sed '/s/meins/deins/' file`

  Mittels **s** wird eine Änderung im Text vorgenommen. Im obigen Fall wird der String »meins« durch »deins« ersetzt.

- **c:** `sed '/Linuks ist toll./ c\`
  `Linux ist toll.' *.tex`

  Die Aktion **c** ersetzt komplette Zeilen. Nach dem Aktionszeichen muss ein Backslash und eine neue Zeile folgen, damit *sed* diesen Ausdruck akzeptiert.

- **a:** `sed '/String/ a\`
  `Das hier ist eine zusätzliche Zeile.' *.tex`

  a steht für append, also das Anhängen von Text. In diesem Fall ist das Anhängen einer gesamten Zeile gemeint.

- **i:** `sed '/String/ i\`
  `Dies hier ist eine zusätzliche Zeile.' *.tex`

  Im Gegensatz zu append fügt **i** die Zeile nicht nach, sondern vor der angegebenen Adresse ein.

- **r:** `sed '/String/ r insert.txt' *.tex`

  r baut die nachstehende Datei – in diesem Fall *insert.txt* – nach der Adresse ein.

sed kann mit einigen Optionen aufgerufen werden. Versuchten Sie vielleicht bereits, bestimmte Zeilen mit der Aktion **p** auszugeben und bekamen kein vernünftiges Ergebnis? Das liegt daran, das die Option `-n`

21 Die Schreibweise ist vom jeweiligen Befehl abhängig und der Aufzählung zu entnehmen.

nicht gegeben war. Diese unterdrückt die Ausgabe des eigentlichen Datenstroms.

Möchten Sie mehrere Kommandos in *sed* in einem einzigen Aufruf integrieren, so kann dies mit `-e` realisiert werden: `sed -e '...' -e '...' file`.

`sed`-Kommandos können außerdem innerhalb einer Datei, besser gesagt, eines Skriptes, abgelegt werden. Dieses wird dann mit der Option `-f` angegeben.

8.10.12 grep

Eines der meistgenutzten Programme ist `grep`. Es wird dazu verwendet, bestimmte Zeilen aus ASCII-Streams herauszufiltern, womit gemeint ist, diese entweder zu unterdrücken oder diese nur anzuzeigen.

Listing 8.114 Verwendung von grep

```
# Alle Dateien, die mit 'kap' beginnen, dann zwei
# belibige Zeichen enthalten und auf '.tex' enden
# aus der Pipe des ls--Komandos herausfiltern:
user$ ls | grep 'kap..\.tex'
kap01.tex
kap02.tex
kap03.tex
kap04.tex
...

# Alle Nachrichten die am 22 November zwischen
# 16:00:00 und 16:59:59 geloggt wurden, ausgeben:
user$ grep 'Nov 22 16:..:.. ' /var/log/messages
Nov 22 16:30:13 eygo /bsd: fupids: user 1000: 1...
Nov 22 16:34:17 eygo /bsd: fupids: new programm...
...
```

Parameter-Aufruf

Der Parameter-Aufruf für `grep` bietet eine Menge nützlicher Befehle. Die wichtigsten und meistgenutzten sind folgende:

-v Löscht alle Zeilen heraus, die nicht mit dem regulären Ausdruck übereinstimmen: `grep -v 'regex'`.

-c / –count Dieser Parameter zeigt die gefilterten Zeilen nicht an, sondern zählt diese. In Verwendung mit dem `-v`-Parameter werden die Zeilen gezählt, welche dem Ausdruck nicht entsprechen.

Listing 8.115 Wie viele Dateien sind es denn?

```
user$ ls | grep -c '^kap..\.tex'
15
user$ ls kap[01][0-9].tex | wc -l
      15
```

-n / –line-number Dieser Parameter gibt die Zeilennummer für die gefundenen Zeilen aus.

Listing 8.116 Zeilenzahl

```
user$ grep -n Slackware kap05.tex
130:Diese Ausgabe stammt vom 2.4.5'er stable-Kernel der
    Slackware 8.0 Distribution.
283:von Slackware Linux befinden sich diese Skripte im
    Verzeichnis /etc/rc.d.
291:  0 & Dieser Runlevel hält das System an. Unter
    Slackware-Linux ist
...
```

Ein kurzes Wort zu egrep

`egrep` verfügt im Gegensatz zu `grep` über die Möglichkeit, mehrere Ausdrücke gleichzeitig zu suchen. Die Ausdrücke werden dabei über eine Pipe separiert und in Anführungszeichen oder Hochkommas eingeschlossen:

Listing 8.117

```
user$ ls -1 | egrep 'November|Dezember'
November01.log
November02.log
November03.log
...
Dezember01.log
Dezember02.log
Dezember03.log
...
```

8.11 Ein paar Tipps zum Schluß

- Das zuletzt ausgeführte Kommando kann in der bash durch die Eingabe von zwei Ausrufezeichen erneut gestartet werden.

Listing 8.118 Programme aus der History

```
user$ ls
...
user$ !!
...
user$ echo !!
echo ls
ls
```

- Der zuletzt angegebene Parameter für ein Kommando wird durch `!$` angesprochen: `ls buch.ps; ls !$`.
- Die aktuelle Prozess-ID liefert die `bash` über die Variable $$.

8.12 Was man sonst noch mit der Shell anstellen kann

Mal von Aufgaben in der Administration abgesehen, ist es relativ problemlos möglich, CGI-Skripte für Webserver in der Shell zu realisieren. Shellskripte können im Prinzip zu fast allen Aufgaben genutzt werden. Die Nachteile sind:

- Sie können keine Systemfunktionen (Syscalls) direkt ausführen.
- Die Shell benötigt einen Interpreter.
- Wenn es um Millisekunden geht, ist die Shell alles andere als geeignet.
- Viele Aufgaben können nur durch externe Programme, die eventuell nicht perfekt Ihren Anforderungen entsprechen, gelöst werden.

Wer also etwas umfangreichere Applikationen entwickeln möchte, sollte zumindest auf eine Sprache wie Ruby oder Perl zurückgreifen. Im Anhang haben wir für Sie ein paar Informationen zur Programmierung unter Linux zusammengetragen – schließlich lebt Linux von freiwilligen Helfern.

15 Multimedia unter Linux

»Die Menschen drängen sich zum Lichte, nicht um besser zu sehen, sondern um besser zu glänzen.«
Fiedrich Nietzsche

Dieses Kapitel beschäftigt sich nun endlich mit der Thematik »Multimedia« und was Sie mit dieser unter Linux so alles anstellen können. Dazu gehört zum einen natürlich Software, zum anderen aber auch Hardwarekonfiguration, damit man die entsprechenden Geräte auch nutzen kann.

Hierzu gehört selbstverständlich auch die grafische Oberfläche X11. Lesen Sie also Kapitel 10 sorgfältig, bevor Sie mit Multimedia anfangen. Sie können mit entsprechenden Programmen zwar auch von der Kommandozeile aus MP3s abspielen, jedoch stellen sich die meisten Menschen zu Recht oft etwas anderes unter dem Stichwort *Multimedia* vor.

15.1 Multimedia unter den Distributionen

Je nach Distribution kann Multimedia gut oder eben weniger gut unterstützt sein. Wie bereits mehrfach erläutert, verfolgen die einzelnen Distributionen schließlich unterschiedliche Philosophien bzw. haben unterschiedliche Zielgruppen im Visier.

Weg des geringsten Widerstands

Wie immer gilt: Sollte Ihre Distribution einen Weg anbieten, der vielleicht leichter oder auch nur anders als der hier beschriebene ist, so versuchen Sie diesen zuerst. Es ist immer praktischer, Software im Paketsystem zu verwalten und so mit Updates klarzukommen, als diese von Hand zu installieren.

15.1.1 Der distributionsunabhängige Weg

Trotzdem wollen wir mit dem quasi »allgemeingültigen«, distributionsunabhängigen Weg anfangen, bevor wir dann kurz auf die Besonderheiten der einzelnen Distributionen eingehen.

Schritt 1: Kernel-Support

Wenn Sie gewisse Hardware unterstützen wollen, benötigen Sie unter Umständen Kernel-Support. Das bedeutet im günstigsten Fall, dass Sie mit `modprobe` ein paar Module laden müssen bzw. im schlechtesten Fall

die Ehre haben, einen neuen Kernel zu kompilieren oder gar Patches einzuspielen.

richtig suchen

Um eine Hilfestellung bzw. erstmal eine Idee von dem zu erhalten, was Sie machen müssen, sollte zunächst die Suchmaschine Ihrer Wahl zu Hilfe gezogen werden. Vielleicht müssen Sie etwas suchen bzw. die Stichworte variieren; die Erfahrung hat aber gezeigt, dass diese Eigeninitiative oft der beste Weg zur Lösung eines solchen Problems ist.

Wenn Sie dann das entsprechende Modul geladen haben – denken Sie an die Kontrolle mit `lsmod` – sollte auch ein entsprechendes Device im `/dev`-Verzeichnis vorhanden sein.

Schritt 2: Die Software

Nun brauchen Sie nur noch entsprechende Software, um die Schnittstellen, die der Treiber bietet, auch optimal nutzen zu können. Wenn Sie im Software-Archiv Ihrer Distribution nicht fündig werden, sind die beiden Webseiten

- `http://www.sourceforge.net`
- `http://www.freshmeat.net`

ein sehr guter Ausgangspunkt, um entsprechende Programme zu suchen.

Vorsicht Falle

Eigentlich sind alle auch nur halbwegs relevanten Projekte auf diesen Seiten gelistet. Achten Sie aber darauf, ob ein gut klingendes Projekt nicht erst noch in der Planungsphase steht bzw. generell schon benutzbar ist. Meistens finden Sie auf den Webseiten der Projekte dann auch Dokumentationen und Anleitungen zum Aufsetzen der eventuell benötigten Hardwareunterstützung.

Wie Sie merken werden, bedarf es der Investition von Zeit, um so langsam auf einen grünen Zweig zu kommen. Wenn Sie sich allerdings diese Zeit nehmen und am Ball bleiben, werden Sie vielleicht irgendwann vollkommen auf Linux umsteigen können und vor allem wollen.

15.1.2 Debian

Was Multimedia anbelangt, ist die Debian-Distribution ein wenig das Sorgenkind. Die Philosophie dieses Projektes besagt, dass am besten nur vollständig freie Software in die Distribution aufgenommen wird. Außerdem hat man beim jeweiligen `stable`-Release nicht unbedingt immer die neuesten Versionen, was sich vor allem in Sachen Multimedia oft schmerzlich bemerkbar macht.

Auf `www.apt-get.org` und `marillat.free.fr` kann man »inoffiziel-le« Debian-Pakete, unter anderem zum Multimediabereich, finden. Mit einer entsprechend konfigurierten */etc/apt/sources.list* kann man dann ganz normal per `apt-get install` seinen Debian-Rechner zu einer Multimediastation machen.

15.1.3 SuSE

Im Gegensatz zu Debian ist SuSE wohl das Musterbeispiel für standardmäßig guten Multimedia-Support. Mit `yast` bzw. `yast2` kann man die zum großen Teil schon während der Installation automatisch erkannte Hardware einfach konfigurieren.

Auch die entsprechende Software ist teilweise schon vorhanden und entsprechend installiert. Ansonsten gibt es auch oft auf den Seiten der durch Freshmeat bzw. SourceForge gefundenen Projekte entsprechende Pakete.

15.2 Konfiguration der Soundkarte

Kommen wir nun zum Thema Soundkartenkonfiguration. Am besten wenden wir dabei das eben Angesprochene an. Zuerst kümmern wir uns um die entsprechende Hardware – die Soundkarte. Und um die richtig einzubinden, müssen wir wissen, wie die Soundunterstützung des Kernels aussieht.

15.2.1 Bis Kernel 2.6 – OSS

Open Sound System

Die Kernel der 2.4er Entwicklungsreihe waren die letzten, die standardmäßig mit OSS, dem Open Sound System, als Unterstützung für Soundkarten ausgeliefert wurden. Man unterscheidet dabei zwischen der OSS-API, einem Interface zu Audiotreibern, das neben Linux auch noch von vielen anderen Unixen unterstützt wird, und den OSS-Treibern, die eben dieses Interface für den Kernel implementierten.

Treiber laden

Die Vorbereitungen bei der Kernel-Konfiguration sahen dann so aus, dass zuerst die entsprechenden Treibermodule mit dem Kernel kompiliert werden mussten. Die Unterstützung für eine Karte wurde dann durch das Laden der entsprechenden Kernel-Module aktiviert. Nach einem Laden mit `modprobe` sah man dann meist so etwas:

Listing 15.1 OSS im Kernel

```
# lsmod
```

```
...
es1370                  30348   1
gameport                 1388   0   [es1370]
soundcore                3428   4   [es1370]
...
```

In diesem Beispiel wurde eine Ensoniq-1370 kompatible Soundkarte benutzt, deren Modul noch das `soundcore`-Modul als Basis sowie die Gameport-Unterstützung nachgeladen hat.

Wenn der Treiber erfolgreich geladen wurde, ist das entsprechende Device */dev/dsp* aktiviert. Wird aber der falsche Treiber geladen, schlägt dies fehl und eine entsprechende Meldung wird ausgegeben:

Listing 15.2 Treiber für die falsche Karte wird geladen

```
# modprobe i810_audio
/lib/modules/2.4.22/kernel/drivers/sound/i810_audio.o:
  init_module: No such device
...
```

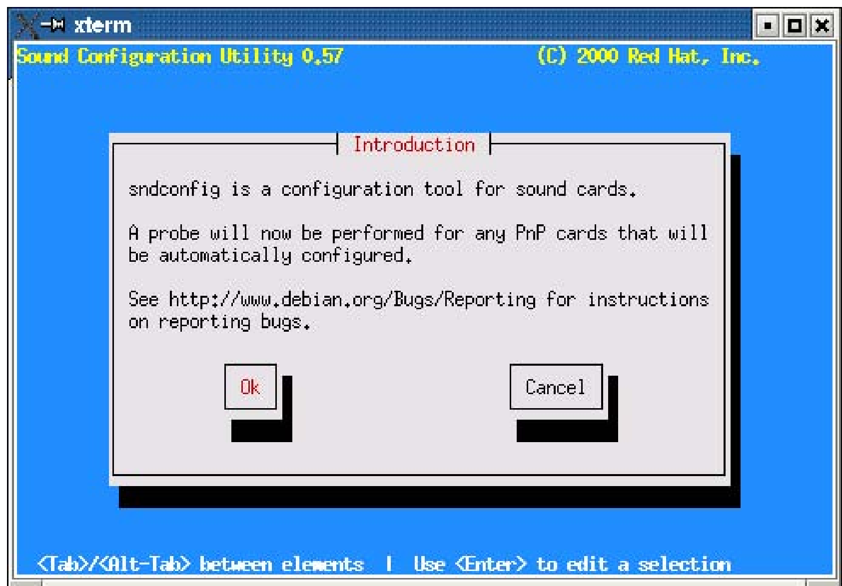


Abbildung 15.1 sndconfig

alles automatisch Auf dieser Basis wird, wie bei SuSE oft, schon während der Installation die Soundkarte automatisch bestimmt. Ist das, wie bei Debian, nicht der Fall,

kann man sich ein Tool wie `sndconfig` herunterladen und versuchen, die Soundkarte automatisch zu bestimmen.

Dieses Tool stammt ursprünglich aus der RedHat-Distribution, ist aber mittlerweile auch für andere Distributionen wie Debian verfügbar.

Test der Konfiguration

Eine hübsche Spielerei zum Testen einer funktionierenden Soundinstallation ist die so genannte *voice of god*:

Listing 15.3 voice of god

```
# cat /vmlinuz > /dev/dsp
```

Bei diesem Beispiel leiten wir einfach die binären Daten des Kernels per Ausgabeumleitung auf die Soundkarte – wenn Sie nun ein lautes Rauschen hören, funktioniert Ihre Soundkarte und Sie wissen, wie sich der Kernel anhört :-)

15.2.2 Ab Kernel 2.6 – ALSA

ALSA steht für Advanced Linux Sound Architecture, ist seit Kernel 2.6.0 standardmäßig dabei und hat damit das Open Sound System ersetzt. Das hat vor allem folgende Gründe:

- Guten Support für alle Arten von Audioschnittstellen, wie normale Soundkarten oder professionelle multichannel Soundkarten
- Modularisierte Soundtreiber
- Ein Multiprozessor- und threadsicheres Design
- Eine Userpace Bibliothek (alsa-lib), die die Anwendungsentwicklung vereinfacht und eine höhere Funktionalität bietet
- Unterstützung für das alte OSS-Interface, für Kompatiblität mit den meisten OSS-Programmen

Außerdem klingen ALSA-Treiber oft auch noch besser als ihre entsprechenden OSS-Varianten.

Die Konfiguration von ALSA läuft dabei gar nicht so anders als die von OSS. Sie müssen auch die entsprechenden Module samt Paramatern laden. Um die ALSA-Module dabei von ihren entsprechenden OSS-Äquivalenten zu unterscheiden, haben diese alle ein `snd-` vor ihrem Namen. So können Sie nach einer erfolgreichen Installation prüfen, ob Ihr System eventuell schon ALSA unterstützt:

Listing 15.4 ALSA im Kernel

```
# lsmod
Module                    Size  Used by
...
snd_via82xx              25184  3
snd_seq_oss              34560  0
snd_seq_midi_event        7936  1 snd_seq_oss
...
snd                      52516  17 ...
```

ausführliche Hilfen

Die Module per Hand zu laden ist zwar recht umständlich, bei den meisten Distributionen allerdings zum jetzigen Zeitpunkt noch immer der einzige Weg. Das wird sich zwar noch ändern, aber da wir nicht wissen wie ALSA und der 2.6er Kernel in die Distributionen integriert sein werden, legen wir Ihnen so lange `http://www.alsa-project.org` ans Herz. Dort erfahren Sie, ob Ihre Karte unterstützt wird, und finden außerdem noch Installationshinweise speziell für Ihre Hardware und Distribution!

15.3 Audiowiedergabe

Im letzten Abschnitt haben wir nur die Soundkarte konfiguriert – und egal ob OSS oder ALSA, bis jetzt haben wir nur das Device. Was uns noch fehlt sind Applikationen, die es benutzen[1]. An dieser Stelle möchten wir noch einmal an Kapitel 10 erinnern, in welchem bereits die Sound-Dämonen angesprochen wurden.

15.3.1 Ausgabemöglichkeiten

Blockade unter OSS

Diese wurden eingeführt, damit Applikationen bei der Benutzung der OSS-Treiber nicht unnötig das Sounddevice (meist */dev/dsp*) blockieren, da unter OSS nur ein exklusiver Zugriff möglich war. Entsprechend den unterschiedlichen Sound-Diensten gibt es für die meisten Programme unterschiedliche Ausgabe-Plugins:

1 Die *voice of god* zählen wir mal nicht dazu.

Schnittstelle	Beschreibung
OSS	Die OSS-API wird direkt zur Ausgabe genutzt.
ALSA	Die ALSA-Schnittstelle wird direkt genutzt.
SDL	Die SDL – eine Medienbibliothek – als Ausgabeschnittstelle nutzen. Diese setzt wiederum auf beispielsweise OSS oder ALSA auf.
aRts	Der KDE Sound-Dämon. Dieser wird normalerweise beim KDE-Start aktiviert.
esd	Der Gnome Sound-Dämon.

Tabelle 15.1 Die wichtigsten Soundausgabemöglichkeiten

Nun stellt sich natürlich die Frage, wie es unter ALSA weitergeht. Mit diesen Treibern können wir blockadefrei auch mit mehreren Programmen gleichzeitig die Audioschnittstelle nutzen.

Das Problem hierbei ist nun viel eher, dass viele Programme noch kein Ausgabeplugin für ALSA besitzen. Von daher werden wir die Sound-Dienste wohl auch in Zukunft noch weiterhin benötigen.

15.3.2 MP3-Player und Co.

Jetzt kommen wir endlich zu den Playern. Von diesen möchten wir als wichtigsten nur den `xmms` vorstellen.

XMMS

MP3!

Das X MultiMedia-System ist recht weit verbreitet und unterstützt alle möglichen Audio- und zum Teil auch Videoformate. Außerdem ist für fast alle Sound-APIs ein Ausgabe-Plugin zu bekommen. Alles in allem erinnert der XMMS in seinem Aussehen und Verhalten an den WinAMP 2.0, mit dem Unterschied, dass XMMS mittlerweile mehr Features unterstützt als das einstige Vorbild.

Was lässt sich zum XMMS noch sagen? Wir schlagen vor, Sie probieren ihn mit Ihren Lieblings-MP3s einfach mal aus. Im Kontrast dazu können Sie ja zum Beispiel die bei KDE mitgelieferten Player ausprobieren.

SoX

die Shell und Sound

Als nicht GUI-Softwarepaket sei hier noch kurz `sox`, das *Schweizer Taschenmesser* für Audio, erwähnt. Auf der Kommandozeile kann man mit SoX verschiendenste Formate ineinander konvertieren und diverse Filter auf die Audiodaten anwenden.

15.3.3 Text-to-Speech

Sie wollen sich einmal einen Text von Ihrem Computer vorlesen lassen? Probieren Sie doch einmal ein Text-to-Speech Tool wie *Festival* aus. Mittlerweile gibt es auch eine Reihe netter Plugins für alle möglichen Anwendungen wie z. B. Browser, mit denen Sie sich sogar Webseiten vorlesen lassen können.

Sollten Sie sich allerdings auf solche Abenteuer einlassen, bringen Sie etwas Zeit und Ruhe mit. Nach der einen oder anderen Bastelstunde kann ein interessantes Ergebnis wahrgenommen werden, mit dem Sie auch so manchen Ihrer Freunde beeindrucken können.

15.4 Videos und DVDs

Im letzten Abschnitt haben wir uns um den Ton gekümmert – der nächste Schritt ist das zugehörige Bild. Wer jetzt denkt, dass man unter Linux mit viel Glück nur die wichtigsten AVIs oder MPEG-Videos abspielen kann, hat weit gefehlt. Dieser Zustand war einmal. Mittlerweile steht Linux mit der entsprechenden Software Windows in nichts nach und teilweise kann man sogar mit einfachen Mitteln mehr anstellen als unter dem Betriebssystem aus Redmond. Doch schauen wir uns dazu erstmal die entsprechende Software an.

15.4.1 DVDs, DivX und Co.

Vorher wollen wir aber noch ein paar Begriffe und Gegebenheiten klären, damit Sie später nicht ins Schleudern kommen.

Codec

Videos entschlüsseln

Videos sind, wie überhaupt jede Multimediadatei, immer in einem speziellen Format hinterlegt. Um das Format dann wieder *lesen* zu können, braucht man sinnvollerweise einen Decoder, der uns die Daten dann entsprechend wieder ausgibt. Ein Codec (**co**der/**dec**oder) ist nun eine Art Software-Bibliothek mit entsprechenden Funktionen.

DivX

DivX ist ein besonderer Codec (genauer: eine Codec-Familie) für Videodateien. Oft sind Filme, die man aus dem Netz laden kann, platzsparend als DivX kodiert. Diese Raubkopien haben DivX erst richtig berühmt gemacht – immerhin kann eine 6 GB DVD so nahezu ohne Qualitätsverluste

auf eine 600 MB Videodatei gebracht werden, die dann auch gebrannt werden kann.

DVD

Die *Digital Versatile Disc*, kurz DVD, ist ähnlich wie die CD auch eine Art »Datenscheibe« – allerdings mit einer viel größeren Kapazität. Diese Kapazität wird heutzutage oft für Filme genutzt, auch wenn die DVD prinzipiell jede Art von Daten speichern kann.

Wichtig bei Video-DVDs sind vor allem die Ländercodes. Sie sollen verhindern, dass eine DVD überall abgespielt werden kann und der Filmindustrie damit Verluste entstehen. Folgende Ländercodes gibt es:

Code	Region
0	Überall spielbar
1	USA, Kanada und US-Kolonien
2	Europa, Grönland, Südafrika, Japan, Ägypten und der mittlere Osten
3	Südost-Asien, Südkorea, Hongkong, Indonesien, Phillipinen und Taiwan
4	Australien, Neuseeland, Mexiko, Zentralamerika und Südamerika
5	Russland und andere Länder der ehemaligen UDSSR, Osteuropa, Indien und Afrika
6	VR China
7	Reserviert für zukünftige Nutzung
8	Internationales Gelände, zum Beispiel in Flugzeugen oder auf Schiffen

Tabelle 15.2 DVD Ländercodes

Außerdem ist die »Verschlüsselung« mit dem CSS-Verfahren zu beachten. Ursprünglich sollte dieses Feature das Abspielen auf nicht-lizensierten Playern verhindern, allerdings wurde die aufgrund von Exportbeschränkungen schwache Verschlüsselung recht bald geknackt.

15.4.2 MPlayer

Der MPlayer (MoviePlayer) ist unter Linux das Nonplusultra, was Videos anbelangt Schauen wir uns einmal die Features an:

- Support für MPEG 1/2/4, DivX 3/4/5, Windows Media 7/8/9, RealAudio/Video bis Version 9, Quicktime 5/6, und Vivo 1/2
- Viele für MMX/SSE(2)/3DNow(Ex) optimierte native Audio- und Videocodecs

- Support für XAnim- und binäre Realplayer Codec Plugins
- Support für Windows Codec DLLs (!)
- Grundlegende VCD/DVD Abspielfunktionalität (inkl. Untertitel)
- Videoausgabe auf allen möglichen und unmöglichen Schnittstellen
- Jedes unterstützte Dateiformat kann in Raw/DivX/MPEG4 AVI (mit pcm/mp3 Audio) konvertiert werden
- Zugriff auf V4L-Geräte wie beispielsweise Webcams
- und anderes mehr

Unterstützung für Windows Codes

Wie Sie sehen, kann der MPlayer als besonderen Clou sogar mit Windows-Codes umgehen. Aus diesem Grund ist auch eine brandneue DivX-Version für den MPlayer absolut kein Problem. Eventuell sollten Sie aber beachten, dass Sie die Win32-Codes als eigenes Paket in Ihrer Distribution finden, so dass Sie unter Umständen dieses noch installieren müssen, bevor alles wirklich so funktioniert, wie es soll.

Konfiguration

Videotreiber

Wie bei allen anderen Multimediaplayern gibt es auch beim MPlayer das Problem, den passenden Ausgabe-Plugin zu wählen – nur gilt es, statt den bekannten Audioschnittstellen auch noch einen passenden Videotreiber auszuwählen. Letzteres geht aber meistens recht schnell, wenn Sie ein Paket Ihrer Lieblingsdistribution nutzen.

Alle entsprechenden Plugins müssen nämlich mit einkompiliert werden, wozu die entsprechenden Bibliotheken auf Ihrem Rechner vorhanden sein sollten.[2] Bei einem Paket sind aber schon die wichtigsten Plugins enthalten. Sie können einfach eins nach dem anderen ausprobieren.

Exkurs: Bugfixes in freier Software

Eigentlich gehört es ja nicht hierher. Bei der Recherche für MPlayer sind wir aber auf der MPlayer-Homepage `www.mplayerhq.hu` auf folgenden interessanten Ablauf eines Bugfixes gestoßen:

Ein großer Bug hatte sich in eine Beta-Release des MPlayers geschlichen – er ließ sich auf einigen Architekturen nicht kompilieren.

Der Ablauf:

---

2 Die meiste Software, wie der MPlayer auch, überprüft während des Übersetzungsvorgangs, welche der Möglichkeiten auf Ihrem Rechner vorhanden ist – daher sollten die entsprechenden Bibltiotheken vorher installiert bzw. die Ausgabe dieser Überprüfung kurz begutachtet werden.

- **09.12.2003 05:24 GMT**
  Der Bug wurde entdeckt.
- **09.12.2003 09:15 GMT**
  Der Bugfix war im CVS.
- **09.12.2003 10:00 GMT**
  Auf dem FTP waren die neuen Dateien zu bekommen.

Binnen fünf Stunden war also ein wichtiger Bug gefixt. Da kommen zwangsläufig Fragen auf: Warum ging das so schnell? Warum kommt so ein Bug überhaupt in ein (Beta-)Release?

Um diese Fragen beantworten zu können, muss man freie Software verstehen. Die Entwickler sind eben darauf angewiesen, dass die Software getestet wird und es Rückmeldungen durch die Nutzer gibt. Immerhin investieren Sie Ihre Freizeit und viel Energie in ihre Projekte – das sollte man respektieren und eben auch unterstützen. Freie Software würde anders nicht funktionieren.

Und dass mit dem Bugfix dann alles doch so schnell ging, ist natürlich Ehrensache der Entwickler.

Der Haken ...

Die Frage bleibt vielleicht, wo der Haken an dieser tollen Software ist. Tatsächlich muss man lange suchen, bis man schließlich etwas findet: MPlayer unterstützt in seiner aktuellen Version keine DVD-Menüs, obwohl man sich die Scheiben trotzdem ohne Einschränkungen ansehen kann. Wen das trotzdem stört, der kann sich ja mal den nächsten Videoplayer anschauen . . .

15.4.3 XINE

XINE unterstützt unter anderem DVD-Menüs. Auch sonst ist die Feature-Liste ähnlich lang wie beim MPlayer. Daher wurden von uns einige Punkte zusammengefasst:

- Abspielen von CDs, DVDs und VCDs
- Support für alle möglichen und unmöglichen Video- und Audiodateien
- Support für Multimediastreams aus dem Netz

Ob man sich nun für XINE oder den MPlayer entscheidet, ist weitestgehend Geschmackssache. Wer will, kann natürlich auch beide installieren und ausgiebig testen. Meistens lässt sich einer von beiden leichter kon-

figurieren, in jedem Fall hilft aber die entsprechende Dokumentation – wenn man nicht zu faul ist, diese zu lesen.

Konfiguration

Für die Konfiguration von XINE gilt, was die Ausgabe-Plugins anbelangt, dasselbe wie für die Konfiguration des MPlayer. Falls es mit der XINE-Konfiguration nicht so richtig klappen will, ist oft die Webseite des Projekts (`www.xinehq.de`) ein guter Startpunkt für die Problemlösung. Teilweise muss man noch für DVD-Support und Ähnliches extra Bibliotheken installieren, damit alles funktioniert.

Beispielsweise benötigen Sie zum Abspielen von DVDs die

- libdvd
- libdvdread
- libdvdnav

15.5 Installation einer TV–Karte

Die Installation einer TV-Karte ist nun ein weiterer Schritt in Richtung Multimedia-PC. Wie auch bei den Soundkarten ist in erster Linie der auf der Karte verwendete Chip wichtig, für den die Unterstützung dann im Kernel aktiviert werden muss. Achten Sie beim eventuellen Kompilieren des Kernels also auf folgende Punkte:

- `<*> Enable loadable module support`
- Video For Linux[3]
  - `[M] Video For Linux`
  - `[M] BT848 Video For Linux`

Der hier selektierte Chipsatz ist für die meisten marktüblichen TV-Karten der richtige. Da aber viele Karten diesen Chip besitzen, müssen Sie beim Laden der Module noch entsprechende, die Karte identifiziernde Optionen angeben. Lesen Sie dazu bitte unbedingt die Dokumentation im Verzeichnis des Kernel-Quellcodes (bzw. unter */usr/share/doc* falls Sie vorkompilierte Pakete verwenden) oder eben im Netz.

3 Video4Linux (v4l) ist die Video-API des Kernels.

Listing 15.5 Installation einer WinTV PCI

```
# insmod videodev.o
# insmod i2c.o verbose=0 scan=1 i2c_debug=0
# insmod tuner.o debug=1 type=1
# insmod msp3400.o debug=0
# insmod bttv.o card=10 pll=1 radio=0
```

erst die Treiber, dann die Tools

Danach sollte die Karte ohne Probleme funktionieren und das Programm kann mit Tools wie `xawtv` unter X11 genossen werden.

15.6 Webcams und Webcam-Software

Für Webcams gilt Ähnliches wie für die TV-Karten: Vor der Nutzung müssen die entsprechenden V4L-Treiber eingebunden werden. Nun benötigt man aber für unterschiedliche Modelle unterschiedliche Module, daher raten wir: Suchen Sie zuerst im Internet Hilfe, bevor Sie sich ausgiebig mit der Thematik auseinander setzen.

Wir werden hier nur ein Beispiel exemplarisch behandeln, damit Sie die Handlungsabläufe prinzipiell verstehen.

15.6.1 Beispiel: USB IBM Cam einrichten

Um den Support für diese Webcam einzubinden, benötigen Sie zunächst folgende Module:

- **videodev**
  Video For Linux Modul
- **usbcore**
  USB Core Modul. Wird auf jeden Fall benötigt.
- **input**
  Input Modul, wird im Allgemeinen auch benötigt.
- **usb-uhci**
  Dieses Modul braucht man, um USB auf dem Motherboard zu aktivieren. Wenn man beim Laden dieses Moduls Fehler angezeigt bekommt, sollte man `usb-ohci` oder die `uhci` Treiber versuchen.
- **ibmcam**
  Das webcam-spezifische Modul. Andere mögliche Webcam-Module (für andere Modelle) könnten sein:

- ov511 z.B. für Creative WebCam III
- dc2xx z.B. für verschiedene Kodak Modelle
- cpia, cpia_usb z.B. für Creative WebCam II

Eventuell muss man nur noch die entsprechenden Einträge im */dev*-Verzeichnis erstellen.[4] Dazu sollte einfach `/dev/MAKEDEV video` – natürlich als »root« – aufgerufen werden.

15.6.2 Webcamsoftware

Nun sollten Sie mit der entsprechenden Software schon ein Bild sehen. Ein paar wichtige Programme wollen wir Ihnen nun noch vorstellen.

xawtv

noch mehr xawtv

Man kann `xawtv` nicht nur zum Fernsehen, sondern auch für den Zugriff auf alle möglichen Multimediadevices – wie eben auch Webcams – nutzen.

Listing 15.6 Verfügbare Video-Devices suchen

```
$ xawtv -hwscan
This is xawtv-3.71, running on Linux/i686 (2.4.20)
looking for available devices

/dev/v4l/video0: OK       [ -device /dev/v4l/video0 ]
type : v4l
name : BT878(Hauppauge (bt878))
flags: overlay capture tuner

/dev/v4l/video1: OK       [ -device /dev/v4l/video1 ]
type : v4l
name : IBM USB Camera
flags: capture
```

Hier haben wir eine WinTV-Karte sowie eine IBM USB Webcam gefunden. Mit folgender Option kann man dann auf die Webcam zugreifen: `xawtv -c /dev/video1`.

4 Ab Kernel 2.6 entfällt dieser Schritt allerdings aufgrund des devfs.

streamer

Mit streamer hat man ein hübsches Programm, um von der Kommandozeile aus diverse Webcams anzusteuern. Man kann dabei einzelne Bilder oder auch Videostreams aufnehmen.

Listing 15.7 Bild aufnehmen

```
$ streamer -c /dev/video1 -b 16 -o bild.jpg
```

Videos in der Shell

Wie auch bei xawtv, gibt man hier mit der -c-Option das Video-Device an. Mit -b stellt man schließlich die Farbtiefe ein, und -o legt die Ausgabedatei fest. Ganz ähnlich funktioniert das auch mit einem Video-Stream:

Listing 15.8 Ein Video aufnehmen

```
$ streamer -c /dev/video1 -f rgb24 -r 4 -t 00:05:00
  -o video.avi -q
```

Dieser Aufruf würde ein 5 Minuten langes (-t 00:05:00) TrueColor AVI (-f rgb24) mit 4 Frames pro Sekunde (-r 4) aufnehmen.

Gqcam

Gqcam wurde ursprünglich für Connectix Quickcams entwickelt, kann mittlerweile aber so ziemlich alle Video4Linux-Devices ansprechen. Das Programm zeichnet sich durch seine intuitive Oberfläche und einfache Bedienung aus.

Motion

Bewegungserkennung

Motion ist ein nettes Programm, welches Bewegungen auf dem Kamerabild erkennt und dann verschiedene Handlungen ausführen kann – beispielsweise das Versenden einer E-Mail, die Aufnahme eines Videostreams oder auch das Hochladen der Daten auf einen Webserver.

Index

.bash_logout 235
.bash_profile 235
.profile 235
/etc/fstab 94, 98, 447
/etc/group 146
/etc/hosts 348
/etc/hosts.allow 369
/etc/hosts.deny 369
/etc/inetd.conf 383
/etc/inittab 107
/etc/lilo.conf 180
/etc/modules 182
/etc/modules.conf 183
/etc/networks 348
/etc/nsswitch.conf 349
/etc/passwd 144
/etc/ppp/options 355
/etc/ppp/pap-secrets 358
/etc/profile 235
/etc/services 382
/etc/shadow 144
/etc/shells 207
/etc/skel/ 145
/etc/ssh/sshd_config 395
/etc/sudoers 82
/var/log/XFree86.log 170
/var/log/messages 168
/var/log/wtmp 170
$?-Variable 254
$HOME 142
$MANPATH 197
$TERM 201
~ 142

A

AbiWord 437
Abschnitte 431
absoluter Pfad 213
Account 493
ACL 84
adduser 143
alias 219
ALSA 453
apache 400
  apachectl 401
  httpd.conf 400
  Module 401
ARP 493
at 194
Ausgabeumlenkung 236
awk 261, 263
  Arbeitsweise 264
  Arrays 280
  bedingte Anweisungen 273
  Befehl ausführen 278
  Builtin-Funktionen 277
  Defaultvariablen 267
  delete 281
  for 275
  Funktionen 276
  getline 277
  if 273
  index 279
  integer-Funktion 278
  Kosinus-Funktion 278
  lenght 279
  Logarithmus 278
  match 279
  printf 277
  Rückgabewerte 277
  Rechenoperationen 272
  Sinus-Funktion 278
  starten 264
  strftime 279
  Strings 265
  sub 279
  systime 279
  tolower 279
  toupper 279
  while 275
  Zeitfunktionen 279
  Zufallsfunktion 278

B

Backup 161
bash 206
Benutzerverwaltung 141
bg 124
Bootdiskette 44

Bootflag 52
Bootloader 42
BSD 493
Bugfix 458
bzip2 167

C

C 493
case 255
cat 220
cd 214
cdrecord 445
CDs kopieren 445
cfdisk 53
chat-Skript 357
chgrp 81
chmod 79
chown 81
chsh 207
CLI 493
Client/Server-Prinzip 380
Compiler 493
compress 167
Cookie 397
cp 222
CPU 493
cron 193
CUPS 426
  Installation 428
  Konfiguration 428
cut 224

D

Dämonprozess 493
Datei
  kopieren 222
  löschen 223
  spalten 231
  umbenenen 222
Dateideskriptoren 192
Dateien 88
  FIFO 91
  Gerätedatei 89
  Link 91
  Pipe 91
  regulär 88
  Socket 91
  Verzeichnis 89
Dateisystem 441
Dateisysteme 95
dd 163
Debian 22
Debugger 480
deluser 145
Device 493
df 94, 187
DHCP 342
dhcp-client 343
Distribution
  RedHat 65
Distributionen 21
  Debian 22, 67, 450
  Gentoo 22
  Knoppix 21
  Mandrake 22
  RedHat 22
  Slackware 23, 49
  SuSE 22, 63, 451
DivX 456
dmesg 168
dos2unix 234
Drucker 425
DSL 363
du 94, 187
DVD 443, 457
  Ländercode 457
DVDs brennen 445
dvips 432

E

E-Mail 410
echo 216
Editoren
  sed 281
egrep 285
Eingabeumlenkung 237
eject 93
elm 413
Eltern-Prozess 116
Escapesequenz 245
expr 245
ext2/3 493

F

FAQs 33
FAT32 439
fdisk 51, 187
fetchmail 417
fg 124
field separator 269
FIFO 91, 240, 493
find 197
finger 386
Finger-Server 385
fips 42
Firewall 493
for 258
free 185
Freigaben (Win) 403
FTP 404, 493
 Client 407
 Protokoll 405
Funktionscode 250
Funktionsschachtelung 251
fvwm2 315
fwbuilder 368

G

gas 479
Gateway 338
gcc 479
gdb 480
Gentoo 22
Gerätedateien 40, 75
Geschichte 26
getty 110
GIMP 327
Gnome 319
GPL 19, 493
gpm 323
Gqcam 463
grep 284
 egrep 285
groff 433
GTK 316
Gtk 486
gzip 167

H

Hardlink 92
Hardware
 CDB 36
 Festplatte 39
 Grafikkarten 37
 Laptops 38
 RedHat HCL 37
 Unterstützung 36
Hash-Verfahren 390
Hashwert 390
hdparm 186
head 171, 229
Heimatverzeichnis 86
Herunterfahren 113
Hexdump 227
HOWTOs 33
HTTP 396, 493

I

if 252
ifconfig 339
Includedateien 160
inetd 382
 inetd.conf 383
Inhaltsverzeichnis 431
init 101, 104
insmod 181
Installation 49
 Bootdisk 58
 cfdisk 53
 Hostname 60
 mehrere Systeme 70
 Modemauswahl 58
 Package–Serien 56
 Partitionierung 50
 RedHat
  Grub 65
 Root Passwort 61
 Setup 54
 Tastaturbelegung 49
 Testen 62
 Zeitzone 61
installpkg 156
Interpreter 493
IPC 493
iproute 346

iptables 367
IRC 330
ISDN 359
  HiSAX 360
ISO 9660 494
ISO-Dateien 444

J

jobs 125, 126
Journalling 494

K

k3b 445
KDE 317
kdevelop 486
kdm 323
Kernel 71, 494
  Code 173
  Energie-Management 177
  erstellen 173
  Kernelversion 28
  Konfiguration 173
  Module 179, 181
  Multitasking 72
  Multiuser 72
  PCMCIA 177
  Singletasking 72
  Singleuser 72
  SMP 176
Kernel-Modul 494
Kernelkonfiguration 173
Kernelmanual 196
Kernelspace 72, 494
kill 128
killall 131
kmail 414
Knoppix 21
KOffice 436
Kommandosubstitution 247
Konvertierung 432
Korn-Shell 206

L

ld 485
less 227
LILO 180
lilo 101
Link 91
LKM 494
Logdateien 168
login 111
Login-Shell 205
Loginshell 111
Loginsystem 169
Loginversuche 170
logrotate 171
Loop Device 441
LP-Tools 426
  lpq 427
  lpr 427
  lprm 427
ls 78
lsmod 181
lsof 139, 192
LVM 448

M

Mail 410
mail (Konsolenprogramm) 411
Mailserver 385
make 483
Makefile 483
man 195
Mandrake 22
Manpage 33
MBR 99
md5sum 390
MDA 410
Memory Management 72
MIME 401
mkdir 220
mke2fs 440
mkisofs 444
mkreiserfs 440
Modems 354
modinfo 181
Modulo 245
more 227
Motif 486
mount 92
Mozilla 325, 414
  Mail 415
    Serverinformation 415
mplayer 457

MTA 410, 420
mtools 233
MUA 410
Multiboot 42
Multitasking 142, 494
Multiuser 494
mutt 414
mv 222

N

NAT 365
  Masquerading 366
NETBIOS 350
netstat 373
Netzmasken 337
Netzwerk 335
Netzwerk-Devices 339
Netzwerkkonfiguration 59
Neustart 113
News 421
Newsgroup 34
NFS 423, 494
nice 132
nl 226
nmap 375
nmbd 404
NNTP 494
Nullmodemkabel 354

O

od 227
oktale Zahlen 79
OpenMotif 486
OpenOffice.org 434
OpenSource 494
OSS (Open Sound System) 451
output-field-separator 270

P

Parentprozess 116
parted 187
Partitionierung 50
Partitionstabelle 99
paste 225
PCMCIA 177
Peer-to-Peer 379
Pfadnamen 213
ping 372
Pipe 91
Pipes 239
pkgtool 154
Portforwarding 365
Portscan 375
PPP 351
  Anmeldung 352
  chat 357
  Einwahl 359
  Modems 354
  options 355
  PAP
    verwenden 358
  Trennung 359
pppd 353
pppoeconf 364
procmail 419
proftpd 409
Programmieren 479
  kdevelop 486
  X11 485
Proxy-Server 494
Proxyserver 397
Prozess 115
  fortsetzen 130
  Gruppierung 120
  Hintergrundprozess 121
  jobs 125
  Kreierung 116
  Priorität 132
  Prozesstabelle 117
  Session 120
  stoppen 130
  timing 139
  Zombie 118
Prozesse 494
Prozessor 102
Prozessstatus 118, 136
Prozesstabelle 138
Prozessumgebung 119
Prozessverwaltung 117
ps 135
ps2pdf 432
pseudo device 76
Pseudo-Dateisystem 494
pstree 133

Q

Qt 316, 486
Quota 494
Quotas 188
Quotasupport 188

R

RAM 494
RAM device 443
Ramdisk 443
rcp 387
reboot 113
RedHat 22
reguläre Ausdrücke 261
ReiserFS 494
removepkg 156
renice 132
rlogin 387
rm 223
rmdir 221
rmmod 182
ROM 494
Rootdisk 46
route 344
Runlevel 105
  wechseln 106

S

Samba 403
scp 391
scsh 206
sed 261, 281
  Befehle 283
select 259
setfacl 85
shared Librarys 484
Shell 205
  -intern 218
  alias 219
  Argumentübergabe 248
  Array-länge 247
  Arrays 246
  bash 206
  bedingte Anweisungen 252
  Benutzereingaben 246
  BuiltIn 210
  Fehlerumlenkung 238
  Funktionen 249
  Kommandogruppierung 239
  Kommandosubstitution 216
  Kommandozeile 211
  Kommentare 243
  Menü 259
  named Pipes 240
  Parameterübergabe 251
  Pipes 239
  Prompt 208
  Rückgabewerte 254
  read 246
  Schleifen 256, 258
  Schreibstil 260
  sh 206
  Skript Interpreter 242
  Skripte 241
  Startskripte 235
  Variablen 243
    Gültigkeit 244
    Rechnungen 244
  wechseln 207
  zsh 206
Shellstart 111
shutdown 113
Signale 128
Slackware 23
sleep 217
SMB 494
smbd 404
SMTP 494
sndconfig 452
Softraid 448
sort 230
Sound 454
Soundkarte 451
sox 455
Speicherverwaltung 72
split 231
SSH 388
  Tunnel 393
  Verschlüsselung 389
ssh-keygen 392
sshd 395
Standard-Ausgabe 127
Standard-Eingabe 127
Startskripte 105
stderr 127

stdin 127
stdout 127
Sticky-Bit 83
strace 481
su 81
Subshell 239, 251, 495
Suchpfad 197
sudo 81
  /etc/sudoers 82
SUID & SGID 83
SuSE 22
SVR4
  Geschichte 25
Swap 74, 495
swapon 185
sylpheed 414
Syscall 495
syslogd 170
System-Administration 196
Systembackup 163

T

tac 225
tail 171, 229
talk 190
tar 165
Tcl 485
TCP-Wrapper 384
TCP/IP 335, 379, 495
  IP-Adressen 336
  IPv6 336
  Netzmasken 337
  Routing 343
tcpdump 377
tee 240
telnet 386
Tex 429
Text-to-Speech 456
Thin Clients 423
time 139
Tk 485
tldp 33
top 137
touch 223
tr 232
TV-Karte 460
twm 315
type 210, 218

U

umask 80
unalias 219
uniq 230
Unix
  BSD 25
  Geschichte 23
unix2dos 234
update-inetd 384
upgradepkg 157
uptime 192
Usenet 421
  Clients 422
  Newsgroups 421
  slrn 422
userdel 143
Usergroup 34
Userspace 72, 495
USV 177

V

Variablen 270
Verzeichnis
  erstellen 220
  löschen 221
Verzeichniswechsel 213
VFS 86, 495
Videoplayer 456
virtuelle Netzwerkschnittstellen 341
Virtuelles Dateisystem 86

W

w 190
wait 128
wc 226
Webcams 461
which 210
while 256
who 190
Window-Manager 312
WindowMaker 315
Windows 42
WLAN 495

write 190
WWW 396, 495

X

X Library 304
X-Sessions 333
X11 449, 495
  .xinitrc 322
  .xsession 324
  Display 305
  Geschichte 303
  Konfiguration 306
  Window-Manager 312
  XF86Config 306
  XFreee86 306
  Zugriffkontrolle 305
xawtv 462
xchat 328
XClient 304
xdm 323
xhost 305
xine 459
Xinerama 331
xmms 455
Xnest 333
XServer 304
xterm 324

Z

Z-Shell 206
ZIP-Laufwerk 439
  mounten 439
Zombie-Prozess 495
Zugriffsrechte 76