

Steffen Wendzel, Johannes Plötner

Einstieg in Linux

Eine distributionsunabhängige Einführung

Auf einen Blick

1	Einleitung	19
2	Installationsvorbereitung	35
3	Linux-Installation	45
4	Die grundlegende Funktionsweise von Linux	65
5	Der Bootstrap-Vorgang	95
6	Programme und Prozesse	111
7	Grundlegende Administration	137
8	Die Shell	203
9	Der vi-Editor	281
10	X11 – Die grafische Oberfläche	291
11	Einführung in Netzwerke	323
12	Serverdienste	355
13	Drucken und Textverarbeitung	395
14	Speichermedien unter Linux	403
15	Multimedia unter Linux	413

Anhang

A	Quellcode	431
B	Die Buch-DVD	437
C	Literatur	439

Inhalt

Vorwort	17
---------------	----

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 Die Kernelversionen	27
1.4 Die Anforderungen an Ihren Rechner	29
1.4.1 Hardwarekompatibilität	29
1.5 Über dieses Buch	29
1.5.1 Was Sie in diesem Buch erwartet	29
1.5.2 Wie Sie dieses Buch lesen sollten	31
1.5.3 Wo Sie weitere Informationen bekommen	32

2 Installationsvorbereitung 35

2.1 Die Anforderungen an Ihre Hardware	36
2.2 Hardwareunterstützung	36
2.2.1 Hardwarekompatibilitäts-Listen der Hersteller ...	37
2.2.2 X11 und Grafikkarten	37
2.2.3 Linux auf Laptops	39
2.2.4 Andere Geräte	39
2.3 Festplatten und Partitionen	39
2.3.1 Vorinstallierte Systeme	42
2.3.2 Windows und Linux	43
2.3.3 UNIX und Linux	43
2.3.4 Erstellen eines Backups	44
2.4 Installationsmedien	44

3 Linux-Installation 45

3.1 Slackware-Installation	45
3.1.1 Die Tastaturbelegung	45
3.1.2 Das erste Login	46
3.1.3 Partitionierung der Festplatte	47

Inhalt

3.1.4	Das Tool cfdisk	49
3.1.5	Setup – die eigentliche Installation	50
3.1.6	Test der Installation	57
3.2	(Open)SUSE-Installation	58
3.2.1	Installation per Mausklick	58
3.2.2	Partitionierung	58
3.2.3	Paketinstallation	59
3.2.4	Installation der Software	59
3.2.5	Konfiguration	59
3.3	Ubuntu, Xubuntu, Kubuntu etc.	60
3.4	Debian-Installation	60
3.4.1	Die Releases von Debian	61
3.4.2	Die Installations-CD	61
3.5	Zusammenfassung	63
4	Die grundlegende Funktionsweise von Linux	65
4.1	Singleuser, Multiuser	66
4.2	Singletasking, Multitasking	66
4.3	Ressourcenverwaltung	66
4.3.1	Speicherverwaltung	66
4.3.2	Swapping	68
4.3.3	Speicherplatz der Festplatte	68
4.3.4	Verwaltung weiterer Ressourcen	69
4.3.5	Schnittstellenbezeichnung unter Linux	69
4.3.6	pseudo devices	70
4.4	Zugriffsrechte	70
4.4.1	Standardrechte	71
4.4.2	Erweiterte Zugriffsrechte	77
4.4.3	Access Control Lists	78
4.5	Das virtuelle Dateisystem	80
4.5.1	Die Verzeichnisstruktur	80
4.5.2	Dateinamen	82
4.5.3	Dateitypen	83
4.5.4	Einhängen von Dateisystemen	87
5	Der Bootstrap-Vorgang	95
5.1	Der MBR	95
5.1.1	Die Partitionstabelle	95
5.1.2	Vom LILO bis zum init-Prozess	97

5.1.3	init	100
5.2	Runlevel-Skripte	100
5.2.1	Wechseln des Runlevels	102
5.2.2	Die Datei /etc/inittab	102
5.2.3	Die Rc-Skripte	104
5.3	getty und der Anmeldevorgang am System	105
5.3.1	(a)getty	106
5.3.2	login	106
5.3.3	Shellstart	107
5.4	Beenden einer Terminalsitzung	107
5.5	Herunterfahren und Neustarten	108
5.5.1	Die Auswahl	108
5.5.2	shutdown	108
6	Programme und Prozesse	111
6.1	Was ist ein Prozess?	111
6.1.1	Das Starten eines Programms	112
6.1.2	Eltern- und Kind-Prozesse	112
6.2	Der Kernel und seine Prozesse	113
6.2.1	Die Prozesstabelle	114
6.2.2	Der Prozessstatus	114
6.3	Prozess-Environment	115
6.4	Sessions und Prozessgruppen	116
6.5	Vorder- und Hintergrundprozesse	118
6.5.1	Wechseln zwischen Vorder- und Hintergrund	120
6.5.2	Jobs – behalten Sie sie im Auge	121
6.5.3	Hintergrundprozesse und Fehlermeldungen	122
6.5.4	Wann ist es denn endlich vorbei?	123
6.6	Das Kill-Kommando und Signale	124
6.6.1	Welche Signale gibt es?	125
6.6.2	Beispiel: Anhalten und Fortsetzen eines Prozesses	126
6.7	Prozessadministration	127
6.7.1	Prozesspriorität	127
6.7.2	pstree	128
6.7.3	Prozessaufistung mit Details via ps	131
6.7.4	top	133
6.7.5	Timing für Prozesse	134

7	Grundlegende Administration	137
7.1	Benutzerverwaltung	137
7.1.1	Linux und Multiusersysteme	137
7.1.2	Das Verwalten der Benutzerkonten	139
7.1.3	Benutzer und Gruppen	142
7.2	Installation neuer Software	144
7.2.1	Das Debian-Paketsystem	145
7.2.2	Das RedHat-Paketsystem	150
7.2.3	Das Slackware-Paketsystem	151
7.2.4	Paketsysteme ohne Grenzen	155
7.2.5	Softwareinstallation ohne Pakete	156
7.3	Backups erstellen	159
7.3.1	Die Sinnfrage	159
7.3.2	Backup eines ganzen Datenträgers	160
7.3.3	Backup ausgewählter Daten	162
7.4	Logdateien und dmesg	165
7.4.1	/var/log/messages	166
7.4.2	/var/log/wtmp	167
7.4.3	/var/log/Xorg.log	168
7.4.4	syslogd	168
7.4.5	logrotate	168
7.4.6	tail und head	169
7.5	Kernel-Konfiguration	170
7.5.1	Die Kernelsourcen	171
7.5.2	Los geht's!	171
7.5.3	Start der Konfiguration	173
7.5.4	Kernelerstellung	177
7.5.5	LILO	178
7.5.6	Grub	179
7.5.7	Ladbare Kernelmodule (LKMs)	179
7.6	Weitere nützliche Programme	183
7.6.1	Speicherverwaltung	183
7.6.2	Festplatten verwalten	184
7.6.3	Benutzer überwachen	186
7.6.4	Der Systemstatus	190
7.6.5	Offene Dateideskriptoren mit lsof	190
7.7	Grundlegende Systemdienste	191
7.7.1	cron	192
7.7.2	at	193
7.8	Manpages	193
7.9	Dateien finden mit find	195

7.9.1	Festlegung eines Auswahlkriteriums	196
7.9.2	Festlegung einer Aktion	198
7.9.3	Fehlermeldungen vermeiden	199
7.10	Der Midnight Commander	199
7.10.1	Die Bedienung	200
7.10.2	Verschiedene Ansichten	201

8 Die Shell 203

8.1	Grundlegendes	203
8.1.1	Was ist eine Shell?	203
8.1.2	Welche Shells gibt es?	204
8.1.3	Die Shell als Programm	205
8.1.4	Die Login-Shell wechseln	205
8.1.5	Der Prompt	206
8.1.6	Shell-intern vs. Programm	208
8.1.7	Kommandos aneinanderreihen	209
8.1.8	Mehrzeilige Kommandos	211
8.2	Arbeiten mit Verzeichnissen	211
8.2.1	Pfade	211
8.2.2	Das aktuelle Verzeichnis	212
8.2.3	Verzeichniswechsel	212
8.2.4	Und das Ganze mit Pfaden	213
8.3	Die elementaren Programme	214
8.3.1	echo und Kommandosubstitution	214
8.3.2	sleep	216
8.3.3	Erstellen eines Alias	216
8.3.4	cat	217
8.4	Programme für das Dateisystem	218
8.4.1	mkdir – Erstellen eines Verzeichnisses	218
8.4.2	rmdir – Löschen von Verzeichnissen	218
8.4.3	cp – Kopieren von Dateien	219
8.4.4	mv – Verschieben einer Datei	220
8.4.5	rm – Löschen von Dateien	220
8.4.6	touch – Zugriffszeiten von Dateien setzen	221
8.4.7	cut – Dateiinhalte abschneiden	221
8.4.8	paste – Dateien zusammenfügen	222
8.4.9	tac – Dateiinhalt umdrehen	223
8.4.10	nl – Zeilennummern für Dateien	223
8.4.11	wc – Zählen von Zeichen, Zeilen und Wörtern ...	223
8.4.12	od – Dateien zur Zahlenbasis x ausgeben	224

Inhalt

8.4.13	Mehr oder weniger, das ist hier die Frage!	225
8.4.14	head und tail	225
8.4.15	sort und uniq	226
8.4.16	Dateien aufspalten	227
8.4.17	Zeichenvertauschung	228
8.5	Startskripte	229
8.6	Ein- und Ausgabeumlenkung	230
8.6.1	Fehlerausgabe und Verknüpfung von Ausgaben	231
8.6.2	Anhängen von Ausgaben	232
8.6.3	Gruppierung der Umlenkung	232
8.7	Pipes	233
8.7.1	Um- und weiterleiten mit tee	233
8.7.2	Named Pipes (FIFOs)	234
8.8	Grundlagen der Shellskript-Programmierung	235
8.8.1	Was genau ist ein Shellskript?	235
8.8.2	Wie legt man los?	236
8.8.3	Das erste Shellskript	236
8.8.4	Kommentare	237
8.8.5	Variablen	237
8.8.6	Rechnen mit Variablen	238
8.8.7	Benutzereingaben für Variablen	240
8.8.8	Arrays	240
8.8.9	Kommandosubstitution und Schreibweisen	241
8.8.10	Argumentübergabe	242
8.8.11	Funktionen	243
8.8.12	Bedingungen	246
8.8.13	Bedingte Anweisungen – Teil 2	249
8.8.14	Die while-Schleife	250
8.8.15	Die for-Schleife	251
8.8.16	Menüs bilden mit select	253
8.8.17	Das Auge isst mit: der Schreibstil	254
8.9	Reguläre Ausdrücke: awk und sed	255
8.9.1	awk – Grundlagen und reguläre Ausdrücke	257
8.9.2	Arbeitsweise von awk	257
8.9.3	Reguläre Ausdrücke anwenden	258
8.9.4	awk – etwas detaillierter	261
8.9.5	awk und Variablen	264
8.9.6	Bedingte Anweisungen	266
8.9.7	Funktionen in awk	269
8.9.8	Builtin-Funktionen	270
8.9.9	Arrays und String-Operationen	273

8.9.10	Was noch fehlt	274
8.9.11	sed	274
8.9.12	grep	276
8.10	Ein paar Tipps zum Schluss	278
8.11	Weitere Fähigkeiten der Shell	279

9 Der vi-Editor 281

9.1	vi	281
9.1.1	Den vi starten	281
9.1.2	Kommando- und Eingabemodus	282
9.1.3	Dateien speichern	283
9.1.4	Arbeiten mit dem Eingabemodus	283
9.1.5	Navigation	284
9.1.6	Löschen von Textstellen	284
9.1.7	Textbereiche ersetzen	285
9.1.8	Kopieren von Textbereichen	286
9.1.9	Shiften	286
9.1.10	Die Suchfunktion	286
9.1.11	Konfiguration	287
9.2	vim	288
9.2.1	gvim	288

10 X11 – Die grafische Oberfläche 291

10.1	Funktionsweise	291
10.1.1	Geschichte	291
10.1.2	Client und Server	292
10.1.3	Das Display	293
10.1.4	XFree86 und X.org	294
10.2	Die Konfiguration	295
10.2.1	Die /etc/X11/xorg.conf	295
10.2.2	xf86config und xorgconfig	299
10.2.3	X -configure	299
10.2.4	Tipps und Tricks	300
10.2.5	Testen der Konfiguration	300
10.3	Window-Manager	301
10.3.1	Warum Window-Manager?	301
10.3.2	Klassische Window-Manager	303
10.3.3	Desktop-Umgebungen	305
10.4	X11 starten	310

Inhalt

10.4.1	Aus dem Textmodus	310
10.4.2	Grafische Login-Manager	312
10.4.3	Startskripte für xdm	313
10.5	Die wichtigsten Programme	313
10.5.1	Eterm, xterm und Co.	313
10.5.2	Mozilla: Browser, Mail- und Usenet-Client	314
10.5.3	The GIMP	315
10.5.4	xchat	317
10.6	Tuning	319
10.6.1	Xinerama und DualHead	319
10.6.2	X11 in einem Fenster	320
10.6.3	Mehrere X-Sessions	321
11	Einführung in Netzwerke	323
11.1	Etwas Theorie	323
11.1.1	TCP/IP	323
11.1.2	Ihr Heimnetzwerk	325
11.2	Konfiguration einer Netzwerkschnittstelle	327
11.2.1	Konfiguration von Netzwerkkarten mit ifconfig ..	327
11.2.2	DHCP	330
11.3	Routing	331
11.3.1	Was ist Routing?	331
11.3.2	route	332
11.3.3	iproute2	334
11.4	Netzwerke benutzerfreundlich – DNS	334
11.4.1	DNS	334
11.4.2	DNS und Linux	336
11.4.3	Windows und die Namensauflösung	338
11.5	Mit Linux ins Internet	339
11.5.1	Einwahl mit DSL	339
11.6	Firewalling und NAT	341
11.6.1	Network Address Translation	341
11.6.2	Firewalling mit iptables	343
11.6.3	Firewalling mit dem TCP-Wrapper	345
11.7	Nützliche Netzwerktools	348
11.7.1	ping	348
11.7.2	netstat	350
11.7.3	nmap	352
11.7.4	tcpdump	353

12 Serverdienste	355
12.1 Grundlegende Konzepte	355
12.1.1 Peer-to-Peer-Netzwerke	355
12.1.2 Das Client-Server-Prinzip	356
12.1.3 Und das Ganze mit TCP/IP	357
12.2 inetd	358
12.2.1 Die /etc/inetd.conf	359
12.2.2 TCP-Wrapper	360
12.2.3 update-inetd	360
12.3 Standarddienste	361
12.3.1 finger	362
12.3.2 telnet	362
12.3.3 Die r-Tools	363
12.3.4 Weitere kleine Server	364
12.4 Secure Shell	364
12.4.1 Das SSH-Protokoll	365
12.4.2 Secure Shell nutzen	367
12.4.3 Der Secure-Shell-Server	371
12.5 Das World Wide Web	372
12.5.1 Das HTTP-Protokoll	372
12.5.2 Einrichten eines Apache-Webservers	375
12.5.3 Den Apache verwalten	379
12.6 Samba	380
12.6.1 Windows-Freigaben mounten	380
12.6.2 Dateien freigeben	381
12.7 Dateien tauschen mit FTP	381
12.7.1 Das FTP-Protokoll	381
12.7.2 FTP nutzen	383
12.7.3 Einen Server konfigurieren	386
12.8 E-Mail unter Linux	386
12.8.1 Grundlegende Begriffe	387
12.8.2 fetchmail	388
12.8.3 procmail	390
12.8.4 MTAs	391
12.9 Das Usenet	392
12.9.1 Newsgroups	392
12.9.2 Clients	393
12.9.3 Server	394

13 Drucken und Textverarbeitung	395
13.1 Druckerkonfiguration	395
13.1.1 CUPS – Common UNIX Printing System	396
13.1.2 Den Drucker benutzen	399
13.2 Textverarbeitungsprogramme	399
13.2.1 OpenOffice.org	400
13.2.2 KOffice	401
14 Speichermedien unter Linux	403
14.1 Neue Festplatten integrieren	403
14.2 Eine Datei als Dateisystem	404
14.2.1 Loop-Device	404
14.2.2 Und das Ganze mit dem RAM	406
14.3 CDs brennen	406
14.3.1 ISO-Dateien erzeugen	406
14.3.2 cdrecord	407
14.3.3 Die benutzerfreundliche Variante: k3b	408
14.4 USB-Sticks und Co.	409
14.4.1 USB-Treiber	409
14.4.2 Das Device ansprechen	410
14.5 SoftRAID und LVM	410
15 Multimedia unter Linux	413
15.1 Multimedia und die Distributionen	413
15.1.1 Der distributionsunabhängige Weg	413
15.1.2 Debian	415
15.1.3 SUSE	415
15.2 Konfiguration der Soundkarte	415
15.2.1 Bis Kernel 2.6 – OSS	415
15.2.2 Ab Kernel 2.6 – ALSA	417
15.3 Audiowiedergabe	418
15.3.1 Ausgabemöglichkeiten	418
15.3.2 MP3-Player und Co.	419
15.3.3 Text-to-Speech	420
15.4 Videos und DVDs	420
15.4.1 DVDs, DivX und Co.	420
15.4.2 MPlayer	421
15.4.3 XINE	423

15.5	Installation einer TV-Karte	424
15.6	Webcams und Webcam-Software	425
15.6.1	Beispiel: USB-IBM-Cam einrichten	425
15.6.2	Webcam-Software	426
Anhang		429
A	Quellcode	431
A.1	Samba-Konfiguration	431
A.2	ProFTPd-Konfiguration	434
B	Die Buch-DVD	437
B.1	Welche Versionen sind auf der DVD?	437
B.2	Benutzung der DVD	437
C	Literatur	439
Index		441

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

4 Die grundlegende Funktionsweise von Linux

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?

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 der zugehörigen Software, die, wie bereits erläutert wurde, in Form von Distributionen vertrieben wird.

Aufbau des Systems

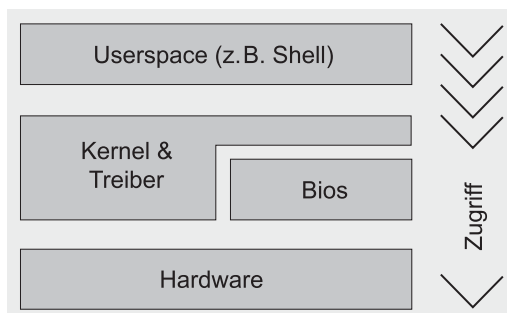


Abbildung 4.1 Aufbau

Der Kernel hat die Aufgabe, die Hardware zu verwalten und Applikationen deren Nutzung zu ermöglichen. Darüber hinaus muss der Kernel das problemlose Nebeneinanderlaufen aller Applikationen gewährleisten, er

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

4.1 Singleuser, Multiuser

UNIX wurde relativ früh in der Entwicklungszeit als ein Betriebssystem mit Multiuserfähigkeiten konzipiert. Betriebssysteme, die multiuser- bzw. mehrbenutzerfähig sind, unterstützen das gleichzeitige Arbeiten mehrerer Benutzer am System. Anders als Betriebssysteme wie Windows 98, OS/2 oder MS-DOS – die als Singleusersysteme bezeichnet werden – verfügen die Benutzer unter UNIX-Systemen (damit auch unter Linux) über sogenannte Accounts. Hat man also so einen Account, 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, das 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 sogenannten *Scheduler* nur ein sehr kleiner Teil der CPU-Zeit zugewiesen, anschließend wird der nächste Prozess in die Verarbeitung geschickt.

Prioritäten Ein Prozess ist dabei *ein Programm in Ausführung*. Er verfügt unter Linux über eine Reihe von Eigenschaften. 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 Prozessen mit geringer Priorität.

4.3 Ressourcenverwaltung

4.3.1 Speicherverwaltung

Kernel- und Userspace Linux verwaltet den Hauptspeicher in zwei getrennten Bereichen: Es gibt den Kernel- und den Userspace. Der Userspace wird dabei von Anwen-

dungen genutzt, wobei der Kernspace 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 Kernspace nur dem Kernel selbst erlaubt. Den vom Benutzer ausgeführten Programmen 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.

Denkt man diesen Gedanken weiter, erscheint es nur folgerichtig, dass auch die Speicherbereiche der einzelnen Programme¹ logisch voneinander getrennt werden. Linux nutzt dafür das Konzept des sogenannten *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.

Virtueller Speicher

Die Programme besitzen also selbst keine Möglichkeit, aus ihrer *virtuellen* Umgebung auszurechnen, aber 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 Bereichsgrenzen im Speicher hinausgeschrieben, sodass 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 genauso 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, hardware- und systemnaher und damit deutlich effizienter zu programmieren. Und da der virtuelle Speicher die Anwendungen ja schließlich voneinander schützt, ist es für das Sys-

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 Speicherabbild, ein »core dump«, auf die Festplatte geschrieben.

tem als Ganzes egal, wenn einzelne Anwendungen einmal abstürzen. Der Rest kann, ohne etwas zu »merken«, unbeeinflusst weiterlaufen.

Der Speicher eines Programms selbst ist nun wiederum in verschiedene Bereiche aufgeteilt. So liegen im virtuellen Speicher eines Prozesses beispielsweise ein sogenanntes *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 von Desktop-Systemen oder kurzzeitig sehr stark ausgelasteten Servern 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 sogenannte *Swapping*. Beim Swapping werden im Moment nicht genutzte Daten aus dem Hauptspeicher auf einen speziellen Teil der Festplatte verschoben, die sogenannte *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 hat 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.

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

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 die Ressource und gestalten dann, beispielsweise mittels einer Warteschlange und verschiedener Programme, den Benutzern den Zugriff auf ihren Dienst.

Als Beispiel für diese Art Dämonprozess sei der Lineprinter-Daemon `lpd` genannt. 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. [zB]

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 mithilfe von Syscalls⁴ ihre Anforderungen (z.B. »Gib mir die für mich neu eingetroffenen IP-Pakete ...«) an den Kernel senden.

Dienste

Dämonprozesse werden oft auch als **Dienste** bezeichnet.

4.3.5 Schnittstellenbezeichnung unter Linux

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

Gerätedateien werden demnach von Userspace-Applikationen angesprochen und benutzt – sofern diese über die entsprechenden Zugriffsberechtigungen verfügen. 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 drei Syscalls durch. Zuerst wird die Gerätedatei geöffnet,

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

dann von dieser Datei gelesen bzw. in sie geschrieben. Nachdem alle Daten gesendet bzw. gelesen worden sind, wird die Schnittstelle wieder geschlossen. Tut dies die Anwendung nicht selbst, so schließt der Kernel sie automatisch bei der Beendigung des Programms.

Deskriptoren Intern werden geöffnete Dateien über sogenannte *Deskriptoren* verwaltet, 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, die nicht in Form von Hardware vorliegen und nur als Software implementiert wurden. Ein Beispiel dafür ist das sogenannte Datengrab */dev/null*.

[zB] 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 Sie in Kapitel 8, »Die 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 Bildschirm bleibt wie gewünscht leer.

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

Listing 4.1 Wir wollen keine Ausgabe sehen.

4.4 Zugriffsrechte

UID und GID Vor die Datei haben die Götter 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 sogenannte UID (User ID). Zudem 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 lesen⁵ oder in es wechseln darf, ist jeweils über ein einzelnes Attribut geregelt.

Für jede Datei können dabei die Rechte für den Eigentümer, dessen Gruppe und den Rest unterschiedlich eingestellt werden. Ändern können die Rechte nur der Eigentümer und `root`. Dabei ist `root`⁶ der Systemadministrator, der von vornherein alles darf. Für ihn gelten keine Einschränkungen durch Rechte, entsprechend groß ist seine Macht.⁷

Der Superuser

Der `root`-Account auf einem UNIX-System 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.

Doch kommen wir nun wieder von der angewandten Gesellschaftskunde der Informatik zurück zum Rechtesystem unter Linux.

Ich, meine Freunde und der Rest der Welt

Wie bereits angesprochen wurde, kann der Eigentümer⁸ 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

⁵ In diesem Zusammenhang ist das Anzeigen der vorhandenen Dateien gemeint.

⁶ Manchmal wird `root` auch als Superuser bezeichnet.

⁷ Daher ist die Übernahme dieses Accounts auch Ziel der Hacker.

⁸ Und natürlich `root`, den wir an dieser Stelle nicht mehr explizit miterwähnen wollen.

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 hat. Dieser Test wird meist positiv verlaufen.⁹ 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 die Nichtmitglieder 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 zumindest 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 mithilfe von Shellkommandos beschreiben, auch wenn Sie an dieser Stelle des Buches noch keinen tieferen Einblick in die Shell haben.¹⁰

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.

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

Listing 4.2 Das Kommando ls

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

```
$ 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
```

Listing 4.3 Ein langes Listing

⁹ Und selbst wenn nicht – der Eigentümer kann sich selbst *jedes* Recht auf eine Datei geben.

¹⁰ An dieser Stelle soll die Beschreibung genügen, dass die bzw. eine Shell eine Art Kommandointerpreter ist, der getippte Befehle ausführt.

In dieser Ausgabe finden wir schon alle Informationen, die wir brauchen. Beide Dateien gehören der Gruppe `users` und dem Benutzer `hannes`. r-w-x

Die Datei `hallo` hat eine Größe von 2344 Bytes und die Datei `test.txt` ist 23 Bytes groß.

Nun wird es etwas komplizierter: Die zweite Spalte gibt die Anzahl der so genannten Hardlinks einer Datei an. Über Hardlinks werden Sie in diesem Buch natürlich auch noch mehr lesen.

Ganz links stehen sonderbare Zeichen und Striche. Diese sind folgendermaßen zu deuten: Das erste Zeichen gibt den Dateityp an. Bei einer regulären Datei wird ein Strich (-) dargestellt. Bei einem Verzeichnis würde ein 'd' und bei einem Unix-Domain-Socket ein 's' dargestellt werden. Außerdem gibt es noch Pipes (p), Block-Devices (b), Character-Devices (c) und symbolische Links (l). Natürlich werden auch all diese Themen im Buch besprochen.

Neben dem ersten Zeichen folgen die Zugriffsrechte, von denen schon oft die Rede war. Dabei stehen die Rechte in der Reihenfolge: Eigentümer, Gruppe und Andere zu jeweils 3 Zeichen. In unserem Beispiel kann der Eigentümer die Datei `hallo` lesen (r), schreiben (w) und ausführen (x). Alle anderen – Gruppenmitglieder und Andere sind gleich – dürfen nur lesen (r) und ausführen (x).

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. Aber keine Angst: Wie immer wird nichts so heiß gegessen, wie es gekocht wird.

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 (8 ist 2^3), also wie gesagt durch drei Bits mit jeweils zwei Zuständen darstellbar.¹¹ 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 hö-

Oktales
Zahlensystem

¹¹ 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.

herwertigen Bits gesetzt, was in diesem Fall *lesen* und *schreiben* bedeutet – *ausführen* ist nicht erlaubt.¹²

Oktalzahl	Übersetzung	Interpretation
777	rw-rw-rwx	Alle dürfen lesen, schreiben und ausführen.
644	rw-r--r--	Der Eigentümer darf lesen und schreiben, alle anderen dürfen 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

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

```
$ 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
```

Listing 4.4 Ändern der Zugriffsrechte

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

In Kapitel 8, »Die Shell«, werden wir uns noch etwas näher mit diesem Kommando beschäftigen, und so soll diese Einführung im Augenblick erst einmal genügen.

umask

Freakfaktor hin oder her, manche Leute können dem Herumrechnen mit binären Repräsentationen diverser Oktalzahlen überhaupt nichts abgewinnen. Für diese Menschen gibt es den Befehl `umask`. Mit ihm wird eine Art Voreinstellung für Rechte vorgenommen, sodass man `chmod` auch mit intuitiveren Parametern sinnvoll aufrufen kann.

¹² Keine Panik, die Rechartmetik ist nicht schwer. Mit etwas Mathe und Gewöhnung sieht alles ganz einfach aus.

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:

```
$ 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
```

Listing 4.5 umask in Aktion

In diesem Beispiel wird die `umask` auf `022` gesetzt. Wenn 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 alle anderen User gesetzt ist. **[zB]**

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, sodass `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.

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

Listing 4.6 Setzen der Eigentumsrechte

Jetzt gehört die Datei `test.txt` dem Benutzer `steffen` und der Gruppe `autoren`. Natürlich kann man denselben Effekt auch mit einem einzigen

Aufruf erzielen, der Befehl `chgrp` ist also nur der Vollständigkeit halber aufgeführt:

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

Listing 4.7 `chown` setzt die Gruppe.

su und sudo

Dass man eine Datei dringend braucht und 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:

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

Listing 4.8 `su` in Aktion

[zB] 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 vim` führt das Programm `vim` 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:

```
...
# Den Benutzern der Gruppe users ohne Passwort alles
# erlauben
%users ALL=(ALL) NOPASSWD: ALL
# Dem Benutzer Test das Mounten/Unmounten des CDROM-
# Laufwerks erlauben (hier wird der Befehl direkt
```

```
# angegeben)
test ALL=/sbin/mount /cdrom,/sbin/umount /cdrom
...
```

Listing 4.9 Die `/etc/sudoers`

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 Programms auswirken. Wenn es nämlich relativ oft gestartet wird, muss es dann nicht jedes Mal von der Festplatte nachgeladen werden.

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-Bit¹³ ist am »t« in den Zugriffsrechten zu erkennen und lässt sich mit dem Kommando `chmod +t` setzen.

Sticky-
Verzeichnisse

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

Listing 4.10 Das Sticky-Bit für Verzeichnisse

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

¹³ Oft findet man in anderer Fachliteratur auch die Bezeichnung »klebriges Bit«.

Dateien oder andere Ressourcen in vertrauenswürdigen Programmen gekapselt wird.

[zB] 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`, das mit dem Suid-Bit ausführbar ist und `root` gehört. Führt ein Benutzer nun `passwd` aus, so kann das Programm mit Root-Rechten Änderungen an der `/etc/shadow` vornehmen. Der Benutzer hat jedoch keine Möglichkeit zur böartigen Manipulation, und da er die Datei nicht mal lesen kann, kann er auch nicht daheim heimlich versuchen, die Passwörter zu entschlüsseln.

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

```
$ 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
```

Listing 4.11 Setzen der Bits SUID und SGID

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 sogenannten *Access Control Lists*, kurz ACLs, implementiert. Früher brauchte man, um ACLs nutzen zu können, noch einen speziellen Kernelpatch, aber ab Version 2.6 des Linux-Kernels sind ACLs auch ohne Patch schon standardmäßig im Kernel integriert. 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 sie sich mit jeder neuen Kernelversion ändert. Die ACLs wollen wir Ihnen aber trotzdem nicht vorenthalten und sie im Folgenden kurz beschreiben.

Access Control Lists sind im Prinzip eine mächtige Erweiterung der Standardrechte. Stellen Sie sich vor, Sie hätten 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 weiterhin auf alle anderen Daten der Gruppe Rechnungswesen zugreifen dürfen. Mit UNIX-Rechten ist diese Situation, wenn überhaupt, nur sehr kompliziert lösbar, mit ACLs ist es jedoch so einfach, wie mit `chmod` ein Recht zu setzen. [zB]

Bei ACLs werden die Rechte nicht mehr nur für den Eigentümer, die Gruppe und alle anderen 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

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

Listing 4.12 ACL-Administration mit `setfacl`

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 sie auf `---` gesetzt wurden, und der Gruppe (`(g:)`) `autoren` wurden 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, so weit vorhanden. Die Rechte aus der ACL können dabei aber nur so weit 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 alle anderen 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.

```
$ getfacl file.txt
#file:file.txt
#owner:jploetner
#group:users
```

4 | Die grundlegende Funktionsweise von Linux

```
user::rw-  
user:swendzel:rw-  
group::r--  
mask::rw-  
other::---
```

Listing 4.13 getfacl

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

4.5 Das virtuelle Dateisystem

In den vorangehenden 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 gleichzusetzen ist oder die Hardware über sogenannte *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 sogenannte *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 Ordnung¹⁴, 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 beim Systemstart unmittelbar benötigt werden. Dort findet man einen oder durchaus auch einmal

¹⁴ Diese Ordnung ist allerdings von UNIX-Derivat zu UNIX-Derivat verschieden. Hat man aber einmal das Prinzip verstanden, so findet man sich auch auf den anderen UNIX-Systemen, die nichts mit Linux zu tun haben, schnell zurecht.

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 Zylinder¹⁵ befand, denn sonst konnte das System nicht gebootet werden.

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 persönlichen Dateien, Schriftstücke und Bilder. Im Normalfall sollte sich ein User auch nur für dieses Verzeichnis interessieren – der Rest der Festplatte geht ihn einfach nichts an. Die Chance, dass ein normaler Benutzer dort etwas kaputt macht, ist einfach zu groß. Demzufolge sind die Rechte normaler Benutzer auf andere Verzeichnisse sehr eingeschränkt, und sie dürfen oft, wenn überhaupt, nur deren Inhalt auflisten. Es reicht ja auch aus, Programme auszuführen. Veränderungen, wie das Einspielen neuer Versionen, sind nur dem Systemadministrator erlaubt.

In */root* hat der Systemadministrator sein Heimatverzeichnis. Allerdings sollte mit dem Root-Account 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.

Root's home

Wichtig ist vielleicht noch anzumerken, dass unter Linux' eigenen Dateisystemen standardmäßig 5% des Plattenplatzes¹⁶ für den Superuser reserviert bleiben. 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.

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

/var

Temporäre Dateien werden im Verzeichnis */tmp* abgelegt. Dabei ist */tmp*, wie in diesem Kapitel bereits erwähnt wurde, das einzige Verzeichnis (abgesehen vom jeweiligen */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

Temporäre Dateien

¹⁵ Eine Maßeinheit, die den geometrischen Aufbau einer Festplatte beschreibt.

¹⁶ Dieser Wert ist veränderbar. Mehr dazu folgt in Kapitel 12, »Serverdienste«.

preisgegeben werden. Das */tmp*-Verzeichnis ist oft auch gar nicht physikalisch auf der Festplatte vorhanden, sondern einfach nur ein Bereich im RAM, der mal eben als RAM-Disk 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 zum Beispiel wichtige Programme in */usr/bin*, Include-Dateien für die C(++)-Programmierung in */usr/include*, die Administrations-Binarys in */usr/sbin/*, 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, sodass selbst ein Hacker mit Rootrechten keine Programme zu seinen Gunsten verändern kann.

4.5.2 Dateinamen

Bei der Betitelung von Dateien sollten Sie einige Regeln beachten. Generell sollten Sie von der Verwendung der Sonderzeichen absehen. 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.¹⁷

Versteckte Dateien Dateinamen, die mit einem Punkt beginnen, gelten als versteckt und werden bei einem normalen `ls`-Aufruf ohne entsprechende Parameter nicht angezeigt. Das ist beispielsweise bei der */.cshrc* oder der */.login* der Fall. Mit dem Parameter `-a` 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 den Überblick über die wirklich wichtigen Dateien verstellen. Aus diesem Grund sind meist nur benutzerspezifische Konfigurationsdateien im */home*-Verzeichnis des entsprechenden Users versteckt.

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

Listing 4.14 Anzeige der schüchternen Dateinamen

¹⁷ Es sei denn, Sie mounten ein Windows-Dateisystem ...

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 auseinanderzusetzen.

Reguläre Dateien

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

Verzeichnisse

Wie versprochen, sprechen wir bei Dateitypen natürlich auch die Verzeichnisse an. Sie existieren im hierarchischen 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, die aus verschiedenen Größen¹⁸ bestehen. Ein Verzeichnis beinhaltet die Nummern der Inode-Einträge aller Dateien, die 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.

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

Listing 4.15 Nicht-rekursive Verzeichnisanzeige

¹⁸ Generell gilt: Je mehr Inode-Nummern vorhanden sind, desto größer muss der Block sein, um diese zu speichern. Typische Blockeinheiten haben eine Größe von 512 Byte, 1 KByte, 2 KByte oder 4 KByte.

Gerätedateien

Gerätedateien haben Sie in diesem Kapitel bereits kennengelernt. Diese besonderen Dateien ermöglichen, wie Sie bereits wissen, den Userspace-Zugriff auf Kernspace-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 hingegen 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. Bei einem Character-Device wie z. B. einer seriellen Schnittstelle ist dieser Puffermechanismus nicht vorhanden, und alle Daten werden unmittelbar übertragen.

Major und Minor Jedem Gerät ist eine sogenannte *Major-* und *Minor-Number* zugeordnet. Die Major-Number ist dem Treiber zugeordnet, die Minor-Number selbst gibt die Gerätenummer für den Treiber an. So werden also die abstrakten und eigentlich willkürlichen Gerätenamen einem entsprechenden Treiber zugeordnet.

Ein Aufruf des `ls`-Kommandos zeigt beim Gerätedateien-Listing entweder `b` für Block-Device oder `c` für Character-Device an.

```
$ ls -l /dev/hda*
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
...
```

Listing 4.16 Die Primärfestplatte

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 Geräte 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

Gerätefile 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 Gerätefile, 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 ein sogenannter 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 z. B. 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 (sogenannte *FIFOs*) dienen zur Kommunikation zwischen Prozessen.¹⁹ Wir werden uns in Kapitel 8 näher mit ihnen beschäftigen.

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

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

Listing 4.17 Erstellung und Darstellung einer FIFO

¹⁹ Dafür wird die Bezeichnung *Interprocess Communication*, IPC, verwendet.

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.

[zB] 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.

[zB] Im folgenden Beispiel wird über das `touch`-Kommando eine Datei namens *myfile* und anschließend ein symbolischer Link *link* auf diese Datei erstellt.

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

Listing 4.18 Ein symbolischer Link

Hardlinks Ein Hardlink unterscheidet sich jedoch von einem 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 befindet sie sich 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.

```
$ 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
```

Listing 4.19 Hardlinks

4.5.4 Einhängen von Dateisystemen

Sie haben das Buch noch nicht entnervt beiseite gelegt? Das freut uns! Kommen wir nun zu einem sehr wichtigen Thema, nämlich zum Einhängen von Dateisystemen. In der Fachsprache benutzt man hierfür das Wort »mount«. Sprechen Sie folglich vom »Mounnten« 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 Einhängpunkt) für andere Dateisysteme dienen. Oft wird beispielsweise das CD-Laufwerk in das Verzeichnis */cdrom* gemountet.

mount

Um ein Dateisystem einzuhängen, wird also das Kommando `mount` benutzt. Dabei werden 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 Netzwerkressource (Network Filesystem) oder Ähnliches sein.

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

Listing 4.20 Beispiel eines Mountings

Hier wurde die erste Partition der zweiten Festplatte²⁰, 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:

```
# 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)
```

Listing 4.21 Was haben wir denn Feines eingehängt?

Mit dem Kommando `umount` wird ein Dateisystem wieder ausgehängt. **[+]** Einsteigern bereitet dieses Kommando jedoch oft Kopfzerbrechen, da sich so manches Dateisystem nicht ohne Weiteres unmounten lässt. Dies liegt oft daran, dass noch ein Prozess in diesem Dateisystem aktiv ist – beispielsweise befindet man sich gerade selbst im Mountpoint.

²⁰ Genauer gesagt: der Primary Slave des IDE-Hostadapters

```
# umount /public
```

Listing 4.22 Unmounten einer Partition

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 unmountet werden, bevor diese physikalisch entfernt werden dürfen, ansonsten kann Datenverlust auftreten!

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

Listing 4.23 Immer raus damit!

df und du

Die beiden Befehle `df` und `du` liefern 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) Ausgabereinheit anzeigen zu lassen.²¹

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

Listing 4.24 Das `du`-Kommando

Eine Übersicht über die Belegung der Dateisysteme gibt das Kommando `df`. Auch hierbei werden ohne entsprechende Angabe des `-h`-Parameters Blockeinheiten als Belegungseinheit²² gewählt.

```
$ 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
```

Listing 4.25 `df` zeigt die Dateisystem-Belegung

²¹ Das `-h` steht für *human readable*.

²² Es existieren noch weitere Parameter wie `-k` für eine Kilobyte-Angabe.

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.

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

Listing 4.26 Inhalt der `/etc/fstab`

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, das gemountet 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 gekommenes 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 damit natürlich in erster Linie zu empfehlen. Sie sind abwärtskompatibel, man kann demnach eine `ext3`-Partition mit einem `ext2`-Treiber

mounten, und alles läuft glatt. Darüber hinaus 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 das Journaling.
- ▶ **'reiserfs'**
Das ReiserFS ist ein relativ neues und sehr weit verbreitetes Journaling-Dateisystem, das binäre Bäume als Grundlage seiner Datenverwaltung benutzt. Als das ext3-System noch nicht fertig war, wurde ReiserFS aufgrund 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 NFS²³ wird für die Speicherung von Dateien auf Fileservern genutzt. Ein so von einem anderen Rechner gemountetes Dateisystem ist für den Benutzer bis auf Performance-Aspekte 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

23 Network Filesystem

alles erst während Ihres Zugriffs zur Laufzeit für Sie erstellt wird. 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 gemountet werden soll. Wenn man ein Dateisystem nicht beim Booten mountet, reicht später ein einfaches `mount` mit 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 ein 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. Demnach sollte das Dateisystem, auf dem sich das Verzeichnis `/dev` befindet, 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, die an dieser Stelle nicht erläutert werden sollen. Falls Sie sich für diese Optionen interessieren, so hilft Ihnen die `mount`-Manpage²⁴ weiter.

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 `umount`²⁵ das Dateisystem auf Fehler hin. 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 Windows-Anwender bevorzugen es, Linux parallel zu einer bestehenden Windows-Installation auf einem Rechner zu verwenden.

FAT32 Eine Windows-Partition mit FAT32-Dateisystem kann man sehr einfach einbinden. Dies geschieht, wie bereits bekannt sein sollte, mit dem Kommando `mount`. Mit dem `-t`-Parameter ist es dabei 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:

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

Listing 4.27 Mounten einer FAT-Partition

²⁴ Wir werden in Kapitel 7 genauer auf die Manpages eingehen.

²⁵ Beispielsweise beim Absturz des Rechners

Verwendet man hingegen eine neuere Windows-Version wie XP oder Vista, ist die Wahrscheinlichkeit sehr hoch, dass man keine FAT32-Partition, sondern eine NTFS-Partition mounten möchte.

mount bekommt als Dateisystem-Parameter hierbei `ntfs` übergeben.

```
mount -t ntfs /dev/hda1 /mnt/windows
```

Listing 4.28 Mounten einer NTFS-Partition

Index

.bash_logout 229
.bash_profile 229
.exrc 288
.profile 229
/etc/fstab 89, 410
/etc/group 142
/etc/hosts 336
/etc/hosts.allow 346
/etc/hosts.deny 346
/etc/inetd.conf 359
/etc/inittab 102
/etc/lilo.conf 178
/etc/modules 180
/etc/modules.conf 181
/etc/networks 336
/etc/nsswitch.conf 337
/etc/passwd 140
/etc/profile 229
/etc/services 358
/etc/shadow 140
/etc/shells 205
/etc/skel/ 141
/etc/ssh/sshd_config 371
/etc/sudoers 76
/home 81
/var/log/messages 166
/var/log/wtmp 167
/var/log/XFree.log 168
/var/log/Xorg.log 168
\$?-Variable 248
\$HOME 138
\$MANPATH 195
\$TERM 200
~ 138

A

Absoluter Pfad 211
ACL 78
adduser 139
alias 216
ALSA 417
apache 375
 access.log 379

apache2ctl 379
error.log 379
httpd.conf 376
Logdateien 379
Module 378
PHP 378
apache2ctl 379
aptitude 149
Arbeitsverzeichnis 211
at 193
Ausgabeumlenkung 230
awk 255, 257
 Arbeitsweise 257
 Arrays 273
 bedingte Anweisungen 266
 Befehl ausführen 270
 Builtin-Funktionen 270
 Defaultvariablen 261
 delete 274
 for 268
 Funktionen 269
 getline 270
 if 266
 index 271
 integer-Funktion 271
 Kosinus-Funktion 270
 length 271
 Logarithmus 271
 match 271
 printf 270
 Rückgabewerte 270
 Rechenoperationen 265
 Sinus-Funktion 270
 starten 257
 strftime 272
 Strings 258
 sub 272
 system 272
 tolower 272
 toupper 272
 while 268
 Zeitfunktionen 272
 Zufallsfunktionen 271

B

Backup 159
bash 204
Benutzerverwaltung 137
bg 120
Block-Device 84
Bootflag 48
Bootloader 43
Bugfix 422
bzip2 164

C

case 249
cat 217
cd 212
cdrecord 407
CDs kopieren 407
cfdisk 49
Character-Device 84
chgrp 75
chmod 73
chown 75
chsh 205
Client-Server-Prinzip 356
compress 164
Cookie 373
cp 219
cron 192
CUPS 396
 Installation 398
 Konfiguration 398
cut 221

D

Dateideskriptoren 190
Dateien 83
 FIFO 85
 Gerätedatei 84
 kopieren 219
 löschen 220
 Link 86
 Pipe 85
 regulär 83
 Socket 85
 spalten 227
 umbenennen 220

Verzeichnis 83

Dateisysteme 89, 404
dd 161
Debian 22
Default-Gateway 56
deluser 141
df 88, 185
DHCP 330
dhcp-client 331
Distributionen 21
 Debian 22, 60, 415
 Fedora 23
 Gentoo 22
 Knoppix 21
 Mandriva 23
 OpenSUSE 58
 RedHat 22
 Slackware 23, 45
 SUSE 22, 58, 415
 Ubuntu 60
DivX 420
dmesg 165
DNS-Server 56
DocumentRoot 378
Domain 55
Drucker 395
DSL 339
du 88, 185
DVD 421
 brennen 408
 Ländercode 421

E

echo 214
Editor 281
 gvim 288
 sed 274
 vi 281
 vim 288
egrep 278
Eingabeumlenkung 231
eject 88
Eltern-Prozess 112
E-Mail 386
Escapesequenz 239
expr 239

F

FAQs 33
 FAT32 92
 fdisk 47, 186
 Fedora 23
 fetchmail 388
 field separator 262
 FIFO 85, 234
 find 195
 finger 362
 Finger-Server 361
 Firefox 314
 for 251
 free 183
 Freigaben (Win) 380
 fsck 92
 FTP 381
 Client 383
 Protokoll 381
 Funktionscode 244
 Funktionsschachtelung 245
 fvwm2 303
 fwbuilder 344

G

Gateway 326
 Gentoo 22
 Gerätedateien 41, 69
 Geschichte 26
 getty 105
 GIMP 315
 Gnome 308
 GPL 19
 gpm 312
 Gqcam 427
 grep 276
 egrep 278
 GTK 305
 gvim 288
 gzip 165

H

Hardlink 86
 Hardware
 Festplatte 39
 Grafikkarten 37

Laptops 39
 RedHat HCL 37
 Unterstützung 36
 Hash-Verfahren 366
 Hash-Wert 366
 hdparm 184
 head 169, 225
 Heimatverzeichnis 81
 Herunterfahren 108
 Hexdump 224
 Home directory 81
 HOWTOs 33
 HTTP 372

I

if 246
 ifconfig 327
 Include-Dateien 158
 inetd 358
 inetd.conf 359
 init 97, 100
 INN 394
 insmod 179
 Installation 45
 Bootdisk 54
 cfdisk 49
 Hostname 55
 Modemauswahl 54
 Package-Serien 52
 Partitionierung 47
 Root-Passwort 57
 Setup 50
 Tastaturbelegung 45
 Testen 57
 Zeitzone 56
 installpkg 152
 iproute 334
 iptables 343
 IRC 318
 ISO-Dateien 406

J

Jobs 121, 122

K

k3b 408
KDE 305
kdm 312
Kernel 65
 Code 170
 Energie-Management 175
 erstellen 170
 Konfiguration 170, 171
 Module 179
 Multitasking 66
 Multiuser 66
 PCMCIA 175
 Singletasking 66
 Singleuser 66
 SMP 175
 Version 28
Kernelmanual 195
Kernelspace 67
kill 124
killall 127
kmail 387
knode 393
Knoppix 21
KOffice 401
Kommandosubstitution 241
Korn-Shell 204

L

less 225
LILO 97, 178
Link 86
Logdateien 165, 166
Login 167
login 106
Login-Shell 107, 204
Loginsystem 166
Loginversuche 167
logrotate 168
Loop Device 404
LP-Tools 396
 lpq 397
 lpr 397
 lprm 397
ls 72
lsmod 179
lsuf 190
LVM 410

M

Mail 386
 Thunderbird 314
Mailserver 361
Major-Number 84
man 193
Mandriva 23
Manpage 33
MBR 95
md5sum 366
MDA 387
Memory Management 66
Minor-Number 84
mkdir 218
mke2fs 403
mkisofs 406
mkreiserfs 403
modinfo 180
Modulo 239
more 225
mount 87
Mozilla 314, 387
mplayer 421
MTA 387, 391
MUA 387
Multiboot 43
Multitasking 138
mv 220

N

NAT 341
 Masquerading 342
NETBIOS 338
netstat 350
Netzmasken 325
Netzwerk 323
Netzwerk-Devices 327
Netzwerkkonfiguration 55
Neustart 108
News 392
Newsgroup 33
nice 128
nl 223
nmap 352
nmbd 381
NNTP
 Clients 393
 Server 394

NTFS 93

O

od 224
 Oktale Zahlen 73
 OpenOffice.org 400
 OSS (Open-Sound-System) 415
 output-field-separator 263

P

Parent-Prozess 112
 parted 186
 Partition 47
 Partitionierung 47
 Partitionstabelle 95
 paste 222
 PCMCIA 175
 Peer-to-Peer 355
 Pfadnamen 211
 ping 348
 Pipe 85, 233
 pkgtool 151
 Portforwarding 342
 Portscan 352
 pppoeconf 340
 procmail 390
 proftpd 386
 Proxy-Server 372
 Prozess 111
 fortsetzen 126
 Gruppierung 116
 Hintergrundprozess 118
 Jobs 121
 Kreierung 112
 Priorität 127
 Prozesstabelle 114
 Session 116
 Status 131
 stoppen 126
 timing 134
 Zombie 115
 Prozessor 97
 Prozessesstatus 114
 Prozesstabelle 133
 Prozessumgebung 115
 Prozessverwaltung 113
 ps 131
 pseudo device 70

pstree 128
 pwd 212

Q

Qt 305
 Quotas 186
 Quotasupport 187

R

Rückgabewert 210
 Rückgabewerte (awk) 270
 RAM device 406
 Ramdisk 406
 rcp 363
 reboot 108
 RedHat 22
 reguläre Ausdrücke 255
 Relativer Pfad 211
 removepkg 153
 renice 128
 rlogin 363
 rm 220
 rmdir 218
 rmmod 180
 route 332
 Runlevel 101
 wechseln 102

S

Samba 380
 scp 367
 scsh 204
 sed 255, 274
 Befehle 275
 select 253
 setfacl 79
 Shell 203
 alias 216
 Argumentübergabe 242
 Array-Länge 241
 Arrays 240
 bash 204
 bedingte Anweisungen 246
 Benutzereingaben 240
 BuiltIn 208
 Editor 281

Index

Fehlerumlenkung 231
Funktionen 243
Kommandogruppierung 232
Kommandosubstitution 214
Kommandozeile 209
Kommentare 237
Menü 253
named Pipes 234
Parameterübergabe 245
Pipe 233
Prompt 206
Rückgabewerte 248
read 240
Schleifen 250, 251
Schreibstil 254
sh 204
Skript-Interpreter 236
Skripte 235
Startskripte 229
Variablen 237
 Gültigkeit 238
 Rechnungen 238
 wechseln 205
 zsh 204
Shellstart 107
shutdown 108
Signale 124
Slackware 23
sleep 216
slrn 393
smbd 381
sndconfig 417
Socket 85
Softraid 410
sort 227
Sound 418
Soundkarte 415
sox 419
Speicherverwaltung 66
split 227
SSH 364
 Tunnel 369
 Verschlüsselung 365
sshd 371
ssh-keygen 368
SSL 378
Standard-Ausgabe 122
Standard-Eingabe 122
Startskripte 100
stderr 123

stdin 122
stdout 122
Sticky-Bit 77
su 76
Subshell 232, 245
Suchpfad 195
sudo 76
 /etc/sudoers 76
SUID & SGID 77
SUSE 22
SVR4
 Geschichte 25
Swap 68
swapon 184
sylvheed 387, 393
syslogd 168
System-Administration 195
Systembackup 160

T

tac 223
tail 169, 225
talk 189
tar 162
tasksel 62
TCP-Wrapper 360
TCP/IP 323, 355
 IP-Adressen 325
 IPv6 324
 Netzmasken 325
 Routing 331
tcpdump 353
tee 233
telnet 362
Text-to-Speech 420
Thunderbird 314
time 134
Timestamp 272
tldp 32
top 133
touch 221
tr 228
TV-Karte 424
twm 303
type 208

U

umask 74
 unalias 217
 uname 190
 uniq 227
 UNIX
 BSD 25
 Geschichte 23
 update-inetd 360
 upgradepkg 154
 uptime 190
 Usenet 392
 Clients 393
 Newsgroups 392
 Server 394
 Thunderbird 314
 userdel 139
 Usergroup 33
 Userspace 67
 USV 175

V

Variablen 264
 Verzeichnis
 erstellen 218
 löschen 218
 Verzeichniswechsel 211
 VFS 80
 vi 281
 ausschneiden 284
 autoindent 287
 Eingabemodus 282
 ersetzen 285
 Kommandomodus 282
 Konfiguration 287
 Navigation 284
 number 287
 shiften 286
 shiftwidth 288
 showmatch 288
 showmode 288
 speichern 283
 Statuszeile 282
 Suchfunktion 286
 tabstop 288
 Text kopieren 286
 Videoplayer 420
 vim 288

Virtuelle Netzwerkschnittstellen 329
 Virtuelles Dateisystem 80

W

w 188
 wait 123
 wc 223
 Webcams 425
 whence 208
 which 208
 while 250
 who 188
 WindowMaker 304
 Window-Manager 301
 Windows 43
 write 189
 WWW 372

X

X-Sessions 320
 X11 413
 .xinitrc 311
 .xsession 313
 Display 293
 Geschichte 291
 Konfiguration 295
 X.org 294
 XF86Config 295
 XFree86 294
 xorg.conf 295
 Zugriffskontrolle 294
 xawtv 426
 xchat 317
 XClient 292
 xdm 312
 xf86config 299
 xhost 294
 xine 423
 Xinerama 319
 XLib 292
 xmms 419
 Xnest 320
 xorg.conf 295
 xorgconfig 299
 XServer 292
 xterm 313

Index

Z

Z-Shell 204

Zugriffsrechte 70