

LinuX

Anwenderhandbuch

und

Leitfaden für die Systemverwaltung

Hetze, Sebastian; Müller, Martin u.a.:

Linux Anwenderhandbuch und Leitfaden für die Systemverwaltung

lunetIX Softfair 1993

ISBN 3-929764-00-8

Das Buch wurde unter \LaTeX mit deutschen Anpassungen gesetzt und mit dvips nach Postscript konvertiert.

Grundlage der Kommandobeschreibungen im Handbuch sind die englischen Manualpages. Wir betrachten sie als auf den Programmen basierende Arbeit im Sinne der GNU General Public License.

Der Text dieses Buches wurde mit größter Sorgfalt geschrieben. Der Verlag, die Autoren und die Übersetzer können für Fehler und daraus resultierende Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Die Fehler, von denen wir wissen daß sie da sind, aber nicht wissen wo, werden wir (vielleicht) in kommenden Auflagen des Handbuches verbessern. Für Hinweise sind wir dankbar. Wir bilden uns nicht ein, alles über Linux oder gar über Unix zu wissen. Wo wir inhaltlich falsch liegen, lassen wir uns gerne belehren.

Dieses Buch unterliegt der GNU General Public License, deren Text im Anhang des Handbuches abgedruckt ist. Das Copyleft liegt bei den Autoren oder bei der Free Software Foundation. Die \LaTeX Datei ist unter den Bedingungen der GNU General Public License erhältlich.

ISBN 3-929764-00-8

Vorwort

Wer heute einen "kleinen" Computer kauft, privat für Zuhause oder als Arbeitsplatzrechner für allgemeine Aufgaben, der kauft Technik mit einem Leistungspotential, das noch zu Beginn der achtziger Jahre den Großrechnern der Konzerne vorbehalten war. Standardmäßig werden diese PC's noch immer mit dem aus dem Jahre 1981 stammenden MS-DOS betrieben. Die damit verbundenen Einschränkungen sind langsam zu Fesseln geworden, von denen sich viele Benutzer befreien wollen. Kein Wunder also, daß der Kampf der Softwaregiganten um die Nachfolge von MS-DOS in vollem Gange ist.

Als Außenseiter in dieser Konkurrenz um Profit und Marktanteile tritt Linux auf, ein Nachbau des altbewährten Unix Betriebssystems. Linux ist freie Software, es steht unter der GNU General Public License. Diese Lizenz garantiert jedem Benutzer das Recht, das Programm beliebig oft zu kopieren, sowie den freien Zugang zu den Quelltexten.

Als Unix-Clone ist Linux eine vollwertige Basis für die meisten Unixprogramme. Der gcc C-Compiler der Free Software Foundation ist der Schlüssel zum gesamten Angebot an freier Unix Software. Durch diesen Rückgriff auf den bestehenden Pool freier Software ist ein "Basissystem" von Linux heute ebenso vielseitig und umfangreich wie die "professionelle" Konkurrenz.

Etwas, was Linux wirklich fehlt, sind die Handbücher, die normalerweise zum Lieferumfang eines Betriebssystems gehören. Es gibt die englischen Manualpages und eine Reihe weiterer ebenfalls in Englisch abgefaßter Hilfstexte, die online gelesen werden können. Außerdem wird von der internationalen Linuxgemeinde an einem Dokumentationsprojekt gearbeitet. Um eine Lücke zu schließen, die unserer Meinung nach bei Unixeinsteigern vor allem auch wegen der Sprachbarriere entsteht, haben wir dieses Buch geschrieben.

Wir stellen uns vor, daß der Leser dieses Buches bereits grundlegende Fertigkeiten im Umgang mit Computern besitzt, die er beispielsweise unter MS-DOS erworben hat. Weiter gehen wir davon aus, daß der Leser keine oder nur wenige Kenntnisse von Unix hat, und daß er kein allgemeines Unixbuch im Regal stehen hat.

Wir wollen einen Anfänger in die Lage versetzen sein Linux Basissystem zu konfigurieren, mit dem Dateisystem umzugehen, Benutzer und Benutzergruppen zu verwalten. Und wir wollen dazu ein wenig Hintergrundinformation zum Betriebssystem geben. Wir haben nicht den Anspruch die Tiefen von Linux zu ergründen. Das halten wir für eine Überforderung des Lesers; abgesehen davon erschien uns der Aufwand dafür zu groß. An einem derartigen Projekt wird wie bereits gesagt international gearbeitet, und der ernsthaft ambitionierte Leser wird früher oder später um die Lektüre der englischen Originaltexte nicht umhinkommen.

Berlin, den 1.3.1993

lunetIX Softfair

Martin Müller &
Sebastian Hetze GbR
Lichtenrader Str. 41
1000 Berlin 44

Tel.: 030/622 73 00
Fax: 030/622 10 75

Wir nehmen Stellung:

Gegen Faschismus und Rassismus!

Diese Ideologien verdrängen die inneren Probleme unserer Gesellschaft nach außen, anstatt sie zu lösen.

Es gibt keine einfachen Rezepte, schon gar keine nationalistische Lösung. Eine grundlegende Neuorientierung der Gesellschaft ist notwendig. Sie kann aber nur mit den, nie gegen die Menschen und Völker der anderen Länder und Kontinente entwickelt werden.

Die sogenannte Asylproblematik wird nicht durch Vertreibung gelöst, sie entsteht durch Vertreibung!

Es gibt viele Arten einen Menschen zu töten.

Nur wenige davon zählen in diesem Land als Asylgrund.

In tiefer Trauer um einen von stolzen, deutschen Faschisten ermordeten Freund, und um alle anderen Opfer faschistischer und rassistischer Gewalttaten.

Die Dummheit der Täter und das Elend ihres Lebens entschuldigt diese Taten nicht.

Für uns gilt weiterhin:

Schaut nicht weg!

Greift ein!

Berlin, den 12.12.1992

Sebastian Hetze und Martin Müller

lunetIX Softfair

Inhaltsverzeichnis

1	Von GNU's, Muscheln und anderen Tieren	9
1.1	Erklärung der Erklärung	9
1.2	ar	9
1.3	basename	10
1.4	bash	11
1.4.1	Optionen	11
1.4.2	Argumente beim Aufruf der Shell	12
1.4.3	Definitionen	12
1.4.4	Reservierte Worte	12
1.4.5	Shell Grammatik	13
1.4.6	Kommentare	14
1.4.7	Quotierung	14
1.4.8	Parameter (Shellvariable)	15
1.4.9	Erweiterung	18
1.4.10	Ein/Ausgabe Umleitung	21
1.4.11	Scriptfunktionen	23
1.4.12	Synonyme	23
1.4.13	Hintergrundprozesse	24
1.4.14	Signale	24
1.4.15	Kommandoausführung	25
1.4.16	Umgebung	25
1.4.17	Eingabeaufforderung	25
1.4.18	Kommandozeileneditor	26
1.4.19	Kommandozeilenspeicher	30
1.4.20	Rechnen in der Shell	31
1.4.21	Eingebaute Shellkommandos	32
1.4.22	Login- und andere Shells	43
1.4.23	Dateien	43
1.4.24	Autoren	43
1.5	cat	44
1.6	chgrp	44
1.7	chmod	45
1.8	chsh	46
1.9	cksum	47
1.10	cmp	47
1.11	comm	48

1.12	compress	49
1.13	cp	50
1.14	csplit	51
1.15	cut	51
1.16	date	52
1.17	dd	54
1.18	df	55
1.19	dirname	56
1.20	doshell	56
1.21	du	57
1.22	egrep	58
1.23	elvis	60
	1.23.1 Überblick	60
	1.23.2 Der ‘visual mode’	60
	1.23.3 Der ‘colon mode’	66
	1.23.4 Reguläre Ausdrücke	72
	1.23.5 Die Optionen von elvis	74
	1.23.6 Zwischenspeicher (Puffer)	79
1.24	env	80
1.25	expand	81
1.26	expr	82
1.27	fdformat	83
1.28	file	84
1.29	find	84
1.30	fold	90
1.31	free	90
1.32	grep	91
1.33	groff	93
1.34	groups	95
1.35	gzip	95
1.36	head	97
1.37	hexdump	97
1.38	hostname	98
1.39	id	99
1.40	install	99
1.41	join	100
1.42	kill	101
1.43	less	101
1.44	ln	107
1.45	login	108
1.46	logname	108
1.47	ls	109
1.48	man	110
1.49	mcoppy	112
1.50	mdel	112
1.51	mdir	113

1.52	mformat	113
1.53	mkdir	114
1.54	mkfifo	114
1.55	mmd	115
1.56	more	115
1.57	mrdd	117
1.58	mread	117
1.59	mttools	118
1.60	mv	118
1.61	newgrp	119
1.62	nice	119
1.63	nl	120
1.64	nohup	121
1.65	od	122
1.66	passwd	125
1.67	paste	125
1.68	pr	126
1.69	printenv	127
1.70	ps	127
1.71	pwd	129
1.72	rm	130
1.73	rmdir	130
1.74	sed	131
1.75	sleep	133
1.76	sort	134
1.77	split	135
1.78	stty	136
1.79	su	140
1.80	sum	141
1.81	sync	141
1.82	tac	142
1.83	tail	142
1.84	tar	143
1.85	tee	145
1.86	touch	145
1.87	tty	146
1.88	uname	147
1.89	uniq	147
1.90	wc	148
1.91	who	149
1.92	write	150

2	Die Kommandos für root	151
2.1	chown	151
2.2	fdisk	152
2.3	fsck und efsck	152
2.4	mkfs	153
2.5	mkefs	154
2.6	mknod	154
2.7	mkswap	156
2.8	mount	156
2.9	rdev	158
2.10	shutdown	159
2.11	umount	159
3	Reise durch's Dateisystem	161
3.1	Namen und Pfade	162
3.2	Das Wurzelverzeichnis	163
3.3	Das Verzeichnis /etc	163
3.4	Das Verzeichnis /dev	170
3.5	Das Verzeichnis /usr	172
3.6	Das Verzeichnis /home oder /usr/home	174
4	Systemverwaltung	175
4.1	Das System starten	175
4.1.1	Von Diskette booten	175
4.1.2	Von Festplatte booten	175
4.1.3	Die Vorgänge bei der Systeminitialisierung	177
4.1.4	init	178
4.2	Das System abschalten	179
4.3	Prozeßordnung	180
4.3.1	Abstürzende Programme und hängende Prozesse	180
4.4	Systemabsturz	181
4.5	Die Konsistenz des Dateisystems prüfen	181
4.5.1	Das Rootfilesystem reparieren	182
4.6	Benutzer eintragen	182
4.6.1	Eintrag in /etc/passwd	182
4.6.2	Gruppenzwang	183
4.6.3	Das Heimatverzeichnis anlegen	184
4.7	Die "Sicherheit" des Systems	184
4.7.1	Eigentum und Zugriffsrechte	184
5	Recompilieren des Kernels	187
5.1	Einführung	187
5.2	Entpacken der Quelltexte	187
5.3	Konfigurieren des Kernels	187
5.3.1	Das Konfigurationsscript	187
5.3.2	Änderungen im Makefile	190
5.3.3	Konfiguration der Ethernetkarte	190
5.4	Das Übersetzen der Quellcodes	190

6	Die Installation von Linux	191
6.1	Die Festplatte partitionieren	191
6.1.1	Die Partitionstabelle	192
6.1.2	Planung	192
6.1.3	Benutzung von fdisk	193
6.1.4	Die Bedeutung des Boot-Flags	195
6.1.5	Die Bedeutung des Partitionstyps	195
6.1.6	SCSI Festplatten einrichten	196
6.2	Eine Distribution einspielen	196
6.2.1	Die SLS Distribution	197
6.2.2	Die LSD Distribution	205
6.3	X-Windows konfigurieren	206
A	GNU GENERAL PUBLIC LICENSE	213

Kapitel 1

Von GNU's, Muscheln und anderen Tieren

1.1 Erklärung der Erklärung

Wir haben versucht, bei den Kommandobeschreibungen die Syntax möglichst eindeutig und konsistent anzugeben. Dazu haben wir aus den zur Verfügung stehenden Stilmitteln die folgenden ausgewählt:

Sans Serif wird für **Kommandonamen**, **-Optionen** und **Parameter** verwendet, die vom Benutzer wörtlich eingegeben werden müssen. Speziell die **-Optionen** beginnen in der Regel mit einem Minuszeichen. Als Option kann jedes darauffolgende Zeichen einzeln verwendet werden.

Kursiv wird für Benutzereingaben wie zum Beispiel *Dateinamen* oder *Suchmuster* verwendet, die normalerweise nicht wörtlich eingegeben werden, sondern bei der Benutzung mit sinnvollen Worten ersetzt werden.

[] eckige Klammern schließen in der Regel Kommandoteile ein, die nicht immer eingegeben werden müssen. Mit solchen Erweiterungen wird das Verhalten eines Kommandos bestimmt und verändert. In den wenigen Fällen, wo die eckigen Klammern selbst Teil des Kommandos sind (wie zum Beispiel beim **test** Kommando) wird der andere Charakter dieser Klammer durch besonders fetten Druck dargestellt.

Wenn mehrere gleichartige Elemente eines Kommandos mehrfach vorkommen können, wird das durch Fortsetzungspunkte '...' gekennzeichnet.

Gelegentlich wird die Auswahlmöglichkeit aus einer genau bestimmten Menge von Parametern, von denen genau einer wörtlich angegeben werden muß, durch geschweifte Klammern { } dargestellt. Die Elemente der Menge sind dann durch Kommata voneinander getrennt. Auch hier wird in den Fällen wo die geschweiften Klammern selbst Teil des Kommandos sind, dies durch besonders fetten Druck gekennzeichnet.

Bestimmte Tasten werden in kleinen Großbuchstaben (KAPITÄLCHEN) benannt. Das sind vor allem das LEERZEICHEN oder SPACE, der TABULATOR oder TAB, der RÜCKSCHRITT oder BACKSPACE, das ZEILENENDE das meist als RETURN manchmal aber auch als NEWLINE angesprochen wird, sowie die Umschalttasten ALT, CONTROL und ESCAPE

Am Ende vieler Kommandobeschreibungen wird in einer Sektion "Siehe auch:" auf andere ähnliche Kommandos hingewiesen. Dabei wird entweder ausdrücklich auf eine Seite im Handbuch verwiesen, oder es ist eine Zahl bzw. das Wort (info) in Klammern dem Kommandonamen nachgestellt. Diese Kommandos sind dann nicht im Handbuch beschrieben, sondern sind als englische Manualpages mit dem **man** Kommando in der entsprechenden Sektion, oder als **T_EXinfo** mit dem **info** Programm nachzulesen.

1.2 ar

Funktion:

ar ist ein Werkzeug zum Erstellen und Verwalten von Archiven oder Bibliotheken, insbesondere für die Objektdaten des C-Compilers

Syntax:

ar [dmpqrtx] [[abi] [*Positionsname*] [cilosuv]] *Archiv Datei* ...

Beschreibung:

Mit **ar** werden beliebig viele Dateien zu einer einzigen Zusammengefaßt. Eine Kompression findet nicht statt. *Archiv* ist der Name der zu bearbeitenden Archivdatei. Die Archivdateien haben per Konvention die Endung **‘.a’**. *Datei* ist der Name einer Datei die in das Archiv eingefügt oder aus ihm gelöscht werden soll.

ar wird vor allem zur Verwaltung von Bibliotheken für den C-Compiler verwendet. Der Linker benötigt die spezielle Datei **‘__SYMDEF’**, in der die Symboltabellen aller Objektdateien zusammengefaßt sind. Mit der Option **‘-s’** wird diese Datei erstellt und in das Archiv eingefügt.

Optionen:

d (delete) löscht *Datei* aus dem *Archiv*

m (move) verschiebt die *Datei* nach (**a**fter) vor (**b**efore) oder ersetzt sie anstelle (**i**nstead) der Datei *Positionname*

p (print) gibt die *Datei* auf die Standardausgabe. Die Zusatzoption **v** (für verbose) gibt den Dateinamen vor jeder Datei aus.

q (quick) fügt die *Datei* in das *Archiv* ein, ohne zu prüfen ob die Datei schon vorhanden ist; durch die Zusatzoption **c** wird die Warnung beim Erzeugen eines neuen Archivs unterdrückt

r (replace) *Datei* ersetzt den gleichnamigen Eintrag in *archiv*. Wenn noch kein Eintrag dieses Namens existiert wird, ein neuer Angelegt.

t (table) gibt das Inhaltsverzeichnis aus; die zusätzliche Option **v** läßt die ausführliche Version anzeigen

x (extract) *Datei* wird aus dem *Archiv* herauskopiert

s sorgt für Erstellung und Aktualisierung der Symboltabelle **‘__SYMDEF’** für den Linker, wie das Kommando **ranlib**

Autor:

viele, Free Software Foundation

1.3 basename

Funktion:

basename liefert den Dateinamen ohne Pfadanteil

Syntax:

basename *Name* [*Suffix*]

Beschreibung:

basename schneidet bei einem Dateinamen mit absoluter Pfadangabe den Pfadanteil des Namens ab und liefert den bloßen Dateinamen. Bei optionaler Angabe einer Dateiendung (Suffix) wird auch diese abgeschnitten. **basename** wird vor allem bei der Shellprogrammierung verwendet.

Autor:

Free Software Foundation

1.4 bash

Funktion

bash – die Wiedergeburtsmuschel. Stark entwickelter Nachfahre eines unsterblichen Wesens aus dem Unix (Kreidezeit).

Syntax

bash [*Optionen*] [*Datei*]

Beschreibung

bash liest und interpretiert Kommandozeilen. Sie ist als Standardshell von Linux die Benutzeroberfläche für ASCII Terminals. (Sie spielt eine ähnliche Rolle wie `command.com` bei MS-DOS.)

Die **bash** bietet eine Vielzahl Funktionen, die das Arbeiten mit der Shell sehr einfach und bequem machen. Zum Beispiel steht ein Kommandozeilenspeicher (→ Seite 27) zur Verfügung, der über die Cursortasten HOCH und RUNTER eine einstellbare Anzahl früherer Kommandos bereithält. Diese Kommandos können mit einem Kommandozeileneditor (→ Seite 26) bearbeitet werden. Eine Herausragende Funktion dieses Kommandozeileneditors wird durch die Tabulatortaste ausgeführt (→ Seite 29). Mit dieser Taste kann ein Kommando, ein Verzeichnis oder ein Dateiname automatisch von der Shell ergänzt werden, soweit er eindeutig ist.

Die **bash** ist kompatibel zu der Bourne-Shell. Sie bietet zusätzlich nützliche Eigenschaften der Korn-Shell (**ksh**) und der C-Shell (**csh**). Die Shell liest Kommandozeilen von der Standardeingabe (interaktiv) oder aus einer Datei (dem Shellsript).

Wenn die Shell interaktiv arbeitet, steht dem Benutzer ein Kommandozeileneditor zur Verfügung. Die Kommandozeile wird mit einem Zeilenende (RETURN) abgeschlossen. Die nichtinteraktive Shell liest eine Skriptdatei zeilenweise, und interpretiert deren Inhalt.

Die Interpretation erfolgt, indem die Zeile in einzelne Worte geteilt wird. Diese Worte werden sequenziell (der Reihe nach) auf bestimmte Worte, Symbole oder Sonderzeichen durchsucht. Entsprechend dieser Zeichen werden die Worte von der Shell verändert (ersetzt). Bei dieser Ersetzung können neue Worte entstehen, die wieder nach den gleichen Regeln verändert werden. Das geschieht so lange, bis die Shell keine speziellen Zeichen mehr findet.

Erst dann wird die Kommandozeile ausgeführt. Wenn eine Kommandozeile eine Liste von Kommandos, oder eine Pipeline enthält, wird jedes Kommando in einer eigenen Subshell gestartet. Jedes einfache Kommando erhält die veränderten Worte (bis zum nächsten Kommando einer Liste) als Argumente übergeben.

1.4.1 Optionen

Zusätzlich zu den beim Shellkommando **set** (→ Seite 38) beschriebenen Optionen und Schaltern, die auch beim Aufruf der Bash in der Kommandozeile verwendet werden können, versteht Bash folgende Optionen:

- c *Zeichenkette*** veranlaßt die Shell, nur die Kommandos in der *Zeichenkette* zu bearbeiten und danach automatisch zu beenden
- i** startet die Shell interaktiv, alle Befehle werden von der Standardeingabe gelesen
- s** zwingt die Shell interaktiv zu starten, auch wenn nach den Optionen weitere Argumente folgen; dadurch können Shellvariable (Positionsvariable) über die Kommandozeile gesetzt werden

Außerdem versteht Bash auch noch eine Reihe von Klartextoptionen; diese Optionen müssen unbedingt vor allen einfachen Buchstabenoptionen gesetzt werden

- norc** unterdrückt die Bearbeitung der Initialisierungsdatei `~/.bashrc`; das ist die Voreinstellung, wenn Bash unter dem Namen **sh** aufgerufen wird
- nopprofile** unterdrückt die Bearbeitung der Dateien `/etc/profile` und `~/.bash_profile` zur Initialisierung als Loginshell
- rcfile *Datei*** verwendet die *Datei* anstelle der `~/.bashrc` zur Initialisierung
- version** zeigt die Versionsnummer der Shell beim Start
- quiet** unterdrückt alle Informationen zum Programmstart
- login** veranlaßt den Start wie als Loginshell
- nobraceexpansion** unterdrückt die Bearbeitung geschweifter Klammern im C-Shell Stil
- nolineediting** unterdrückt die Funktionen des Kommandozeileneditors

1.4.2 Argumente beim Aufruf der Shell

Wenn nach allen Optionen weitere Argumente in der Kommandozeile stehen, und weder die ‘-c’ noch die ‘-s’ Option gesetzt sind, wird das erste verbleibende Argument als Dateiname interpretiert, aus dem die weiteren Kommandos gelesen werden (Shellscript). Dieser Dateiname ist dann in der Shellvariablen ‘0’, alle weiteren Argumente in den folgenden Positionsvariablen für die Bearbeitung im Shellscript zugänglich. Wenn alle Kommandos aus der Datei abgearbeitet sind, wird die Bash automatisch beendet.

1.4.3 Definitionen

Folgende Begriffe werden zur Erklärung der Grammatik benutzt:

Blank ist der leere Zwischenraum von Worten. Neben dem Leerzeichen (SPACE) ist auch ein Tabulator (TAB) möglich

Wort ist eine durch Blank oder Sonderzeichen eingeschlossene Folge von Zeichen. Die Shell teilt die Eingabezeile zur Bearbeitung in Worte auf. Ein weiterer Begriff für Wort ist **token**.

Name (für eine Variable oder eine Scriptfunktion) kann nur ein Wort aus “alphanumerischen Zeichen” sein. Das sind genauer Folgen von Buchstaben oder Zahlen, die mit einem Buchstaben beginnen. Der Unterstrich ‘_’ zählt zu den Buchstaben.

Sonderzeichen sind Trenner zwischen einzelnen *Worten*, die zusätzliche Aufgaben erfüllen, wenn sie nicht in Anführungszeichen eingeschlossen sind.

| & ; () < > und das Zeilenende (RETURN) werden als Sonderzeichen interpretiert

Kontrolloperator ist ein Sonderzeichen mit spezieller Kontrollfunktion. Folgende Kontrolloperatoren sind Definiert: || & && ; :: () |

1.4.4 Reservierte Worte

Wie bei jeder anderen Programmiersprache gibt es für die Sprache zur Shellprogrammierung reservierte Worte. Diese Worte dürfen nicht für Variablennamen oder zum Benennen von Scriptfunktionen benutzt werden. Sie werden nur erkannt, wenn sie ohne Anführungszeichen, und als erstes Wort eines einfachen Kommandos oder als drittes Wort eines **case** oder **for** Kommandos auftreten.

Folgende Worte sind reserviert:

! case do done elif else esac fi for function if in then until while { }

1.4.5 Shell Grammatik

Einfache Kommandos

Ein einfaches Kommando ist eine Zuweisung an eine Shellvariable oder eine durch Blanks getrennte Folge von Worten. Ein einfaches Kommando wird durch ein Zeilenende oder einen Kontrolloperator abgeschlossen. Das erste Wort eines einfachen Kommandos benennt die auszuführende Funktion. Die anderen Worte werden diesem Kommando als Argumente übergeben.

Jede Funktion und jedes Kommando kann eine einzige Zahl an die aufrufende Shell oder allgemeiner an das aufrufende Programm zurückgeben. Dieser Rückgabewert wird als **Status** bezeichnet. Ein Status von Null bedeutet in der Regel, daß kein Fehler aufgetreten ist; ein Status ungleich Null signalisiert dagegen häufig einen Fehler.

Wenn ein Kommando durch ein Signal beendet wurde, ist der Rückgabewert (Status) der Wert des Signals + 128.

Pipelines

[!] **Kommando** [| **Kommando** ...]

Eine Pipeline ist die Zusammenfassung von einem oder mehr einfachen Kommandos durch ein '|' Zeichen (Pipe). Die Standardausgabe des ersten Kommandos wird mit der Standardeingabe des zweiten Kommandos verbunden. Diese Verbindung wird vor einer Ein/Ausgabe-Umlenkung eingerichtet.

Der Status der gesamten Pipeline ist gleich dem Status des letzten Kommandos in der Pipeline. Wenn der gesamten Pipeline ein Ausrufezeichen '!' vorangestellt ist, gibt die Pipeline das logische Gegenteil (Komplement) als Status zurück.

Jedes Kommando der Pipeline wird als separater Prozeß mit einer eigenen Subshell ausgeführt.

Listen

Eine Folge von Kommandos kann durch die Kontrolloperatoren &&, ||, & oder ; zu einer Liste zusammengefaßt werden. Die Liste wird durch ein &, ; oder ein Zeilenende (RETURN) abgeschlossen.

Die Kontrolloperatoren && und || erzeugen eine logische UND bzw. ODER Verknüpfung der Kommandos:

Kommando1 [&& **Kommando2** ...]

Kommando2 wird genau dann ausgeführt, wenn Kommando1 einen Rückgabewert (Status) von Null hat, also fehlerfrei abgeschlossen wurde.

Kommando1 [|| **Kommando2** ...]

Kommando2 wird genau dann ausgeführt, wenn Kommando1 einen Rückgabewert (Status) ungleich Null liefert.

Kommandogruppen

Kommandos können auf verschiedene Weisen zu Gruppen zusammengefaßt werden:

(Liste) Ausführung mit eigener Umgebung. Die Kommandos in der Liste werden in einer Subshell ausgeführt. Änderungen in der Umgebung – z.B. an Shellvariablen – sind nur für die Kommandos in der Liste verfügbar.

{Liste;} einfache Gruppierung. Die Kommandos in der Liste werden einfach in der gegebenen Shellumgebung ausgeführt.

for Name [in Wort] do Liste done ist eine Schleifenkonstruktion. Ein auf 'in' folgendes Wort wird zu Token erweitert (siehe unter Erweiterung). Die Liste von Kommandos wird dann so oft durchlaufen, wie Token erzeugt worden sind. Für jeden Durchlauf wird ein Token in der Variablen *Name* übergeben. Wenn der 'in Wort' Teil fehlt, wird die Liste für jeden gesetzten Positionsparameter (siehe unter Parameter) einmal ausgeführt.

Als Status wird der Rückgabewert des letzten ausgeführten Kommandos zurückgegeben. Wenn kein Kommando ausgeführt wurde ist der Status Null.

case *Wort* in [*Muster* [*Muster* ...]) *Liste* ;; ...] **esac** ist eine Verzweigungs-konstruktion. Das Wort wird zuerst zu Token erweitert, dann wird jedes Token mit den Mustern verglichen und bei Übereinstimmung die Liste von Kommandos ausgeführt.

Wenn ein übereinstimmendes Muster gefunden wurde, wird nicht nach weiteren Übereinstimmungen gesucht.

Der Status ist Null, wenn keine Übereinstimmung gefunden wurde. Sonst wird der Status des zuletzt ausgeführten Kommandos der Liste übergeben.

if *Liste* **then** *Liste* [**elif** *Liste* **then** *Liste* ...][**else** *Liste*] **fi** ist eine einfache Verzweigung. Zuerst werden die Kommandos der 'if *Liste*' ausgeführt. Wenn der Status dieser Liste Null ist, wird der then Teil abgearbeitet. Mit **elif** können beliebig lange if-else Ketten erzeugt werden. Wenn der 'if *Liste*' Teil des letzten **elif** nicht Null ist, wird der abschließende **else** Teil bearbeitet.

Der Status der gesamten if-Konstruktion ist gleich dem Status des zuletzt ausgeführten Kommandos, oder Null, wenn kein Kommando ausgeführt wurde.

while *Liste* **do** *Liste* **done** bearbeitet die 'do *Liste*' so lange wie das letzte Kommando der 'while *Liste*' einen Status von Null liefert. Der Status der while-Schleife ist gleich dem Status des letzten Kommandos des 'do-Teils', oder Null, wenn kein Kommando ausgeführt wurde.

until *Liste* **do** *Liste* **done** die until-Schleife entspricht der while-Schleife, mit dem Unterschied, daß der do-Teil ausgeführt wird, wenn die 'until *Liste*' einen Status ungleich Null liefert.

[**funktion**] *Name* () { *Liste* } definiert eine neue Scriptfunktion *Name*. Die im Funktionskörper definierten Kommandos werden ausgeführt, wenn der Funktionsname in einem einfachen Kommando eingesetzt wird. Der Rückgabewert der Scriptfunktion ist gleich dem Status des zuletzt innerhalb der Funktion ausgeführten Kommandos (siehe auch den Abschnitt über Scriptfunktionen auf Seite 23).

1.4.6 Kommentare

In einem Shellsript werden Zeilen, die mit einem '#' beginnen als Kommentar bewertet. Diese Zeilen werden bei der Bearbeitung ignoriert.

1.4.7 Quotierung

Quotierung wird benutzt, um die spezielle Bedeutung von Kontrollzeichen, reservierten Worten oder Namen auszuschalten. Auf diese Weise können Parameter an Funktionen übergeben werden, die Sonderzeichen, Namen oder reservierte Worte enthalten, ohne daß die Shell eine Veränderung an den Parametern vornimmt.

Es gibt drei Formen der Quotierung:

1. durch das Escape-Zeichen \ (Backslash)
2. durch Hochkomma ' (Quote)
3. durch Anführungszeichen " (Doublequote)

Das Escape-Zeichen "entwertet" das unmittelbar folgende Sonderzeichen. Ein durch das Escape-Zeichen entwertetes Zeilenende wird ignoriert.

Die in Hochkommata eingeschlossenen Worte werden von der Shell nicht weiter bearbeitet. Lediglich ein Hochkomma darf nicht in Hochkommata eingeschlossen werden; auch nicht wenn es durch ein Escape-Zeichen eingeleitet wird.

Von den in Anführungszeichen eingeschlossenen Worten erkennt die Shell nur die Sonderzeichen \$, ' und \ als solche. Alle anderen Worte bleiben unbearbeitet. Das Escape-Zeichen behält seine Bedeutung aber nur,

wenn es von einem der Zeichen \$, ', " , \ oder dem Zeilenende (RETURN) gefolgt wird. Ein Anführungszeichen darf zwischen zwei Anführungszeichen stehen, wenn es durch ein Escape-Zeichen eingeleitet wird.

Die speziellen Parameter * und @ haben eine besondere Bedeutung, wenn sie zwischen Anführungszeichen auftauchen (siehe bei Spezialparameter, Seite 15).

1.4.8 Parameter (Shellvariable)

Ein Parameter ist ein Platzhalter für bestimmte feste oder variable Werte bei der Shellprogrammierung. Als Parameter sind Namen, Zahlen oder die unten aufgeführten Spezialparameter zugelassen. Eine einfache Variable wird mit einem Namen bezeichnet.

Parameter können in Shellprogrammen als Zwischenspeicher für Werte verwendet werden; mit Parametern kann der Ablauf eines Shellprogramms gesteuert werden, und es können gespeicherte Werte beim Aufruf als Teil der Kommandozeile an Programme übergeben werden.

Der Zugriff auf den Wert eines Parameters bzw. die Übergabe eines Parameters erfolgt durch den Operator '\$' gefolgt von der Parameterbezeichnung.

Ein Parameter gilt als gesetzt, wenn ihm ein Wert zugewiesen wurde. Eine leere Zuweisung – der Nullstring "" – gilt als Wert in diesem Sinne. Wenn eine Variable gesetzt ist, kann sie nur durch das unset Kommando entfernt werden.

Eine Variable kann durch eine Zuweisung der folgenden Form erzeugt werden:

Name = [**Wert**]

Wenn kein Wert angegeben ist, wird der Variablen die leere Zeichenkette (Nullstring) zugewiesen.

Außerdem kann eine Variable mit verschiedenen Shellfunktionen erzeugt und initialisiert werden. (Siehe bei den Shellfunktionen declare und export auf den Seiten 33 und 35.)

Positionsparameter

Ein Positionsparameter wird durch eine positive ganze Zahl bezeichnet. Die Positionsparameter werden beim Aufruf der Shell in der Kommandozeile, oder durch das set Shellkommando (mit den Optionen '-' oder '--', → Seite 38) gesetzt.

Wenn ein Positionsparameter mit mehr als einer Stelle (>9) bezeichnet werden soll, muß er in geschweifte Klammern gesetzt werden.

Spezialparameter

Einige Parameter haben besondere Bedeutung. Diese Parameter können nur gelesen werden. Eine Zuweisung an diese Parameter ist verboten.

- * steht (als Parameter) für alle Positionsparameter von 1 an. In Anführungszeichen gesetzt steht "\$*" für ein einziges Wort, bestehend aus dem Inhalt aller Positionsparameter mit dem ersten IFS als Trennzeichen.
- @ steht ebenfalls für alle Positionsparameter von 1 an. In Anführungszeichen gesetzt wird es aber durch die Werte der einzelnen Positionsparameter (jeweils ein einzelnes Wort) ersetzt.
- # steht für die Anzahl der Positionsparameter.
- ? liefert den Rückgabewert (Status) des zuletzt ausgeführten Kommandos.
- steht für die Optionsflags (von set oder aus der Kommandozeile).
- \$ steht für die Prozeßnummer der Shell.
- ! steht für die Prozeßnummer des zuletzt im Hintergrund aufgerufenen Kommandos.
- 0 steht für den Namen des Shellscripts oder der Shell selbst, wenn kein Shellsript ausgeführt wird.

- (Unterstrich) steht für das letzte Argument des zuletzt ausgeführten Kommandos (nach allen Erweiterungen).

Shellvariable

Die folgenden Variablen werden durch die Shell gesetzt:

PPID ist die Prozeßnummer des Elternprozesses der Shell.

PWD ist das aktuelle Verzeichnis, wie es vom `cd` Shellkommando gesetzt wird.

OLDPWD ist das zuletzt aktuelle Verzeichnis (wird ebenfalls vom `cd` Shellkommando gesetzt).

REPLY wird vom Shellkommando `read` gesetzt, wenn keine andere Variable als Rückgabeparameter benannt ist; → Seite 37.

UID ist die Benutzerkennung des aktuellen Anwenders. Diese Kennung ist in der Datei `/etc/passwd` dem Benutzernamen zugeordnet.

EUID ist die effektive Benutzerkennung des Anwenders. Während der Ausführung von Programmen, bei denen das `SUID` Bit gesetzt ist, wird die effektive Benutzerkennung des Eigentümers der Programmdatei gesetzt.

BASH beinhaltet den kompletten Pfadnamen der aktuellen Shell.

BASH_VERSION beinhaltet die Versionsnummer der Shell.

SHLVL steht für das Shelllevel. Bei jedem Aufruf einer neuen Shell in der Shell in der Shell wird der Shelllevel um eins erhöht. Eine Möglichkeit zwischen den Levels zu wechseln besteht nicht.

RANDOM liefert bei jeder Abfrage einen neuen Pseudo-Zufallswert. Die Folge von Pseudo-Zufallszahlen kann durch eine Zuweisung an `RANDOM` initialisiert werden. Gleiche Initialwerte führen zu gleichen Zahlenfolgen.

SECONDS liefert die Anzahl von Sekunden seit dem Start der aktuellen Shell. Wenn `SECONDS` ein Wert zugewiesen wird, erhöht sich dieser Wert entsprechend jede Sekunde automatisch um eins.

Die folgenden Variablen werden von der Shell ausgewertet. In einigen Fällen wird die Variable mit einem Standardwert initialisiert.

IFS ist der "interne Feld-Separator". Alle Zeichen dieser Shellvariablen werden als Trenner von Worten erkannt. Die Standardbelegung für `IFS` ist Leerzeichen Tabulator und Zeilenende.

PATH enthält eine durch Doppelpunkt getrennte Liste von Verzeichnissen. Wenn ein einfaches Kommando nicht als internes Shellkommando erkannt wird, und nicht mit komplettem Pfadnamen (das ist der Pfad vom aktuellen Verzeichnis oder vom Wurzelverzeichnis aus) angegeben wird, dann sucht die Shell in allen Verzeichnissen der `PATH` Variablen nach einem Programm mit passendem Namen und führt es aus.

Ein einzelner Punkt anstelle des Wurzelverzeichnisses steht für das aktuelle Verzeichnis. Eine Tilde `'~'` steht für das Heimatverzeichnis des Anwenders.

HOME ist das Heimatverzeichnis des aktuellen Benutzers. Dieses Verzeichnis wird in der Datei `/etc/passwd` bestimmt und von `login` automatisch in die Variable `HOME` geschrieben. Das in `HOME` gespeicherte Verzeichnis ist der Standardwert für das `cd` Shellkommando; → Seite 33.

CDPATH ist eine durch Doppelpunkt getrennte Liste von Verzeichnissen. Wenn beim `cd` Shellkommando kein absoluter Verzeichnisname und kein Verzeichnis im aktuellen Verzeichnis benannt wird, werden alle Verzeichnisse im `CDPATH` nach einem passenden Verzeichnis durchsucht. In das erste passende Verzeichnis wird gewechselt.

- ENV** enthält den Namen einer Datei, die Kommandos zur Initialisierung für die Shellumgebung beim Bearbeiten von Shellscrippts enthält. (z.B. die Datei `~/.bashrc`)
- MAIL** enthält den Namen der Datei in der die elektronische Post für den Benutzer gespeichert wird. Wenn diese Datei nicht leer ist wird der Anwender darauf hingewiesen, daß Post für ihn da ist.
- MAILCHECK** spezifiziert die Zeitintervalle nach denen die Shell prüft ob Post da ist. Die Voreinstellung ist 60 Sekunden.
- MAILPATH** ist eine durch Doppelpunkt getrennte Liste von (absoluten) Dateinamen, in denen Post für den Benutzer ankommen kann. Zu jeder Datei kann eine durch '?' eingeleitete Zeichenkette angegeben werden, die auf dem Bildschirm ausgegeben wird, wenn in der entsprechenden Datei Post angekommen ist.
- MAIL_WARNING** ist ein Schalter. Wenn die Variable gesetzt ist (egal welcher Wert), dann wird eine Warnung ausgegeben, wenn auf die Postdatei zugegriffen wurde.
- PS1** ist die (rohe) Zeichenkette, die als Eingabeaufforderung (Prompt) die Arbeitsbereitschaft der Shell anzeigt. Die Variable kann eine Reihe weiterer Parameter enthalten, die vor der Ausgabe nach den im Abschnitt Eingabeaufforderung (Seite 25) erläuterten Regeln erweitert werden. Die Voreinstellung ist `'\$'`.
- PS2** enthält die Zeichenkette für die sekundäre Eingabeaufforderung. Sie wird genau wie PS1 erweitert. Die sekundäre Eingabeaufforderung erscheint, wenn zu einem gegebenen Kommando interaktiv über die Shell weitere Kommandos oder Parameter von der Tastatur gelesen werden sollen. (Zum Beispiel nach dem Kommando `for`) Die Voreinstellung ist `'>'`.
- PS4** wird zur Anzeige der erweiterten Kommandos mit der Option `'-x'` benutzt (siehe beim Shellkommando `set`, Seite 38). Voreinstellung ist `'+'`.
- NO_PROMPT_VARS** ist ein Schalter. Wenn die Variable gesetzt ist, werden nur die im Abschnitt über die Eingabeaufforderung (Seite 25) beschriebenen speziellen Parameter erweitert.
- HISTSIZE** setzt die Anzahl der im Kommandozeilenspeicher erinnerten Zeilen fest.
- HISTFILE** benennt eine Datei, in die der Inhalt des Kommandozeilenspeichers beim Beenden der Shell automatisch gesichert wird. Der Kommandozeilenspeicher wird beim Start einer neuen Shell aus dieser Datei aufgefüllt.
- PROMPT_COMMAND** benennt ein Kommando, das vor jeder Eingabeaufforderung automatisch ausgeführt wird.
- IGNOREEOF** hat nur eine Bedeutung, wenn die Shell interaktiv läuft. Wenn die Variable eine ganze Zahl enthält, so wird diese Anzahl von 'EOF'-Zeichen (CTRL-D) für das Dateiende bzw. das Ende der Eingabe erwartet, bevor die Shell verlassen wird. Zu jedem 'EOF' wird der Hinweis ausgegeben, daß die Shell mit `logout` verlassen werden soll.
- Wenn die Variable leer ist oder keine Zahl enthält, so ist die Voreinstellung 10. Wenn die Variable nicht gesetzt ist, führt jedes EOF-Zeichen sofort zum Verlassen der Shell.
- HOSTTYPE** enthält eine Kennung zur Identifikation des Rechnertyps. Für Linux kommen nur die Typen `'i386'` und `'i486'` in Frage.
- TMOUT** mit dieser Variablen kann die Shell veranlaßt werden, nach einer bestimmten Zeit ohne Benutzeraktivität, also ohne Eingabe, automatisch zu beenden. Nur wenn die Variable eine Zahl enthält, wird diese Anzahl Sekunden auf die Eingabe gewartet.
- FCEDIT** benennt einen anderen Editor als `vi` als Standardeditor für das `fc` Shellkommando; → Seite 35.

notify ist ein Schalter. Wenn die Variable gesetzt ist, wartet die Shell mit der Nachricht über die Beendigung eines Hintergrundprozesses nicht bis zur nächsten Eingabeaufforderung, sondern platzt sofort mit der Meldung heraus.

history_control bestimmt, welche Kommandos in den Kommandozeilenspeicher geschrieben werden. Wenn die Variable das Wort **'ignorespace'** enthält, werden nur die Zeilen in den Speicher geschrieben, die nicht mit einem Leerzeichen beginnen. Wenn die Variable das Wort **'ignoredups'** enthält, werden alle Zeilen, die der zuletzt gespeicherten Zeile entsprechen, nicht gespeichert. Beide Worte zusammen werden nicht erkannt. Wenn die Variable nicht gesetzt ist oder irgendeinen anderen Wert enthält, werden alle Kommandozeilen in den Kommandozeilenspeicher geschrieben.

glob_dot_filenames ist ein Schalter. Wenn die Variable gesetzt ist, werden auch Dateinamen, die mit einem Punkt beginnen, in die Pfadnamenerweiterung einbezogen.

allow_null_glob_expansion ist ein Schalter. Wenn die Variable gesetzt ist, werden Muster, die bei der Pfadnamenerweiterung nicht erweitert werden konnten, zu einer leeren Zeichenkette erweitert anstatt unverändert zu bleiben.

histchars enthält zwei Zeichen zur Kontrolle der Wiederholung von Kommandos aus dem Kommandozeilenspeicher. Das erste Zeichen leitet eine Kommandozeilenerweiterung aus dem Kommandozeilenspeicher ein. Die Voreinstellung ist '!'. Das zweite Zeichen kennzeichnet einen Kommentar, wenn es als erstes Zeichen eines Wortes auftaucht. Ein solcher Kommentar wird bei der Ausführung eines Kommandos ignoriert.

nolinks ist ein Schalter. Wenn die Variable gesetzt ist, folgt die Shell beim wechseln des aktuellen Verzeichnisses mit dem `cd` Shellkommando nicht den symbolischen Links.

hostname_completion_file ist der Name einer Datei mit dem gleichen Format wie `/etc/hosts`. Die Einträge dieser Datei werden zur Hostnamenerweiterung verwendet.

noclobber ist ein Schalter. Wenn die Variable gesetzt ist, können existierende Dateien nicht durch die Ausgabeumlenkungen `>`, `>&` und `<>` überschrieben werden. Anhängen von Daten an eine existierende Datei ist auch bei gesetztem **noclobber** Schalter möglich. Wie bei der C-Shell gibt es keine Möglichkeit diesen Schalter zu umgehen.

auto_resume ist ein Schalter. Wenn die Variable gesetzt ist, vergleicht die Shell ein einfaches Kommando ohne Ein/Ausgabe-Umlenkung mit den im Hintergrund laufenden Prozessen. Wenn ein Prozeß im Hintergrund einen passenden Namen hat, wird der erste passende Prozeß in den Vordergrund geholt.

no_exit_on_failed_exec ist ein Schalter. Wenn diese Variable gesetzt ist, wird die Shell nicht durch ein abgebrochenes mit `exec` aufgerufenes Kommando abgebrochen.

cdable_vars ist ein Schalter. Nur wenn diese Variable gesetzt ist, kann dem `cd` Shellkommando das Verzeichnis in das gewechselt werden soll auch in einer Variablen übergeben werden.

pushd_silent ist ein Schalter. Wenn diese Variable gesetzt ist, geben die `pushd` und `popd` Shellkommandos nicht den kompletten Verzeichnisstapel aus, nachdem sie erfolgreich ausgeführt wurden; → Seite 37.

1.4.9 Erweiterung

Die von der Shell gelesenen Kommandozeilen werden zuerst in Worte zerlegt. Diese Worte werden dann verschiedenen Erweiterungsprozeduren unterworfen.

Durch Worttrennung oder Pfadnamenerweiterung kann die Anzahl der Worte in der Kommandozeile dabei verändert werden.

Tildenerweiterung

Wenn ein Wort mit einer Tilde ('~') beginnt, werden alle Buchstaben vor dem ersten Schrägstrich '/' (Slash) als Benutzername interpretiert. Diese werden dann durch den absoluten Pfad des Heimatverzeichnisses dieses Benutzers ersetzt. Wenn kein Benutzername angegeben ist, wird die Tilde durch den Inhalt der Shellvariablen HOME bzw. das Heimatverzeichnis des aktuellen Anwenders ersetzt.

Wenn die Tilde von einem '+' gefolgt wird, so wird das aktuelle Verzeichnis aus der Shellvariablen PWD eingesetzt. Wird die Tilde von einem '-' gefolgt, so wird mit dem letzten aktuellen Verzeichnis aus der Variablen OLDPWD erweitert.

Die Shell führt eine Tildenerweiterung auch in allen benutzten Shellvariablen durch. Dabei werden neben den Tilden am Wortanfang auch solche erkannt und ersetzt, die auf den Doppelpunkt ':' folgen. Auf diese Weise kann Tildenerweiterung auch für die Shellvariablen PATH, CDPATH und MAILPATH durchgeführt werden.

Parametererweiterung

Das '\$' Zeichen leitet Parametererweiterung, Kommandosubstitution und arithmetische Ersetzung ein. Der zu erweiternde Parameter kann in geschweiften Klammern eingeschlossen werden, um ihn von den folgenden Zeichen sicher abzugrenzen, die sonst als Teil des Variablennamens mißverstanden werden könnten.

\${Parameter} Der Wert des Parameters wird eingesetzt. Die geschweiften Klammern sind zwingend, wenn ein Positionsparameter mit mehr als einer Ziffer benannt ist, oder wenn der Variablenname nicht eindeutig von den darauffolgenden Zeichen getrennt werden kann.

Die folgenden Konstruktionen stellen verschiedene Arten bedingter Parametererweiterung dar. Der mit *Wort* bezeichnete Teil ist seinerseits wieder Gegenstand von Tildenerweiterung, Parametererweiterung, Kommandosubstitution und arithmetischer Erweiterung. In allen Konstruktionen, die einen **Doppelpunkt** enthalten, wird der Parameter daraufhin getestet, ob er leer oder ungesetzt ist. Diese Konstruktionen können alle auch **ohne** den Doppelpunkt verwendet werden. In diesem Fall wird der Parameter nur daraufhin getestet, ob er ungesetzt bzw. gesetzt (auch leer!) ist.

\${Parameter:-Wort} Benutzung von Standardwerten. Wenn der *Parameter* ungesetzt oder leer ist, wird das *Wort* anstelle des gesamten Ausdrucks eingesetzt.

\${Parameter:=Wort} Setzen von Standardwerten. Wenn der Parameter ungesetzt oder leer ist, wird der Inhalt vom Wort dem Parameter zugewiesen, und der neue Parameter eingesetzt. Den Positionsparametern und den Speziellen Parametern kann allerdings auch auf diese Weise kein Wert zugewiesen werden!

\${Parameter:?Wort} gibt eine Fehlermeldung, wenn der Parameter leer oder ungesetzt ist. Der Inhalt von Wort wird als Fehlermeldung auf der Standardfehlerausgabe ausgegeben, und eine nicht-interaktive Shell wird verlassen. Wenn der Parameter gültig gesetzt ist, wird sein Wert eingesetzt.

\${Parameter:+Wort} erzwingt die Benutzung eines anderen Wertes. Wenn der Parameter nicht leer oder ungesetzt ist, wird der Inhalt von Wort eingesetzt. Sonst wird nichts eingesetzt.

\${#Parameter} wird durch die Anzahl der Zeichen im Parameter ersetzt. Wenn als Parameter '*' oder '@' angegeben werden, wird die gesamte Länge des erweiterten Parameters eingesetzt. Das ist die Längensumme aller Positionsparameter inklusive der Worttrenner.

\${Parameter#Wort} und
\${Parameter##Wort}

das *Wort* wird mit dem Anfang vom *Parameter* verglichen. Dabei werden Jokerzeichen (Wildcards) ('*', '?' und [...]) behandelt, wie bei der Pfadnamenerweiterung (Seite 20) beschrieben. Im Fall der Konstruktion mit einem '#' wird das kürzeste passende Muster vom Parameter entfernt, im Fall der zwei '##' wird das längste passende Muster vom Parameter entfernt.

$\${Parameter}\% Wort$ und
 $\${Parameter}\%\% Wort$

das *Wort* wird mit dem Ende des *Parameters* verglichen. Die Jokerzeichen werden auch hier wie bei der Pfadnamenerweiterung behandelt. Bei der Konstruktion mit einem ‘%’ wird das kürzeste passende Muster vom Parameter entfernt, bei der mit zwei ‘%%’ wird das längste passende Muster entfernt.

Kommandosubstitution

Die Kommandosubstitution ermöglicht es, die Ausgabe eines Kommandos direkt einer Shellvariablen zuzuweisen. Es gibt zwei mögliche Formen der Kommandosubstitution:

$\$(Kommando)$ und
'Kommando'

Die Kommandosubstitution wird durchgeführt, indem die Standardausgabe des Kommandos anstelle des Parameters eingesetzt wird.

Wenn die Substitution in der “alten” Form mit den Apostrophe (Backquote, nicht Hochkomma) durchgeführt wird, behält das Escape-Zeichen ‘\’ nur dann seine Sonderfunktion, wenn es vor den Zeichen ‘\$’, ‘ ’ oder ‘\’ steht. Wenn das Kommando in der Klammerform aufgerufen wird, werden alle Zeichen unverändert belassen. Kommandosubstitution kann auch verschachtelt werden. In der “alten” Form muß vor den inneren Apostrophe ein Escape-Zeichen stehen.

Wenn eine Kommandosubstitution innerhalb von Anführungszeichen durchgeführt wird, wird die Ausgabe des Kommandos nicht mehr in einzelne Worte geteilt.

Arithmetische Erweiterung

Durch die arithmetische Erweiterung ist es möglich, die Ergebnisse arithmetischer Berechnungen für die Shellprogrammierung zu verwenden. Durch die Konstruktion

$\$[Ausdruck]$

wird anstelle des *Ausdrucks* das Ergebnis der im Abschnitt über das Rechnen in der Shell beschriebenen Berechnungen eingesetzt (→ Seite 31).

Der Ausdruck wird behandelt, als wenn er in Anführungszeichen stehen würde. Zusätzliche Anführungszeichen im Ausdruck werden ignoriert. Alle Worte des Ausdrucks werden vor der Berechnung einer Parametererweiterung, Kommandosubstitution und Quotenreduktion unterzogen.

Worttrennung

Die bisher beschriebenen Erweiterungen (Parametererweiterung, Kommandosubstitution und arithmetische Erweiterung) werden vor der weiteren Bearbeitung jeweils einer Worttrennung unterzogen, wenn die erweiterten Parameter nicht in Anführungszeichen stehen.

Bei der Worttrennung wird jedes der in der Shellvariablen IFS – dem internen Feld Separator – festgelegten Zeichen als Trenner behandelt. Wenn der Inhalt der IFS Variablen exakt SPACE TAB RETURN (die Voreinstellung) ist, wird eine beliebige Folge dieser Zeichen als ein einziger Trenner behandelt. Anderenfalls führt jeder Separator zu einem neuen Wort. Wenn die IFS Variable leer ist, wird keine Worttrennung durchgeführt.

Leere Worte, wie sie zum Beispiel durch die Erweiterung leerer Parameter entstehen, werden entfernt, wenn sie nicht ausdrücklich als leere Zeichenkette in Anführungszeichen gesetzt sind.

Wenn keine Erweiterung durchgeführt wurde, findet auch keine Worttrennung statt.

Pfadnamenerweiterung

Nach der Worttrennung werden alle Worte der Kommandozeile nach den Zeichen *, ?, und [durchsucht. Wenn ein solches Zeichen gefunden wird, wird das gesamte Wort als Muster für eine Pfadnamenerweiterung benutzt. Es werden alle Dateien und Verzeichnisse des aktuellen Verzeichnisses mit dem Muster verglichen und anstelle des Musters eine alphabetisch geordnete Liste aller passenden Pfadnamen eingesetzt.

Wenn kein passender Pfadname gefunden wurde, und die Shellvariable `allow_null_glob_expansion` nicht gesetzt ist, bleibt das Wort unverändert. Anderenfalls wird es aus der Kommandozeile entfernt.

Bei der Pfadnamenerweiterung werden Datei- oder Verzeichnisnamen, die mit einem Punkt beginnen nur dann berücksichtigt, wenn die Shellvariable `glob_dot_filenames` gesetzt ist. Auch der die Verzeichnisnamen trennende Schrägstrich ‘/’ (Slash) kann durch kein Jokerzeichen ersetzt werden, sondern muß immer ausdrücklich angegeben werden.

Die Jokerzeichen (Wildcards) zur Pfadnamenerweiterung haben die folgende Bedeutung:

***** paßt auf alle Zeichenketten, einschließlich der leeren Zeichenkette

? paßt auf jedes einzelne Zeichen, außer dem Zeilenende

[...] paßt auf alle der in den eckigen Klammern eingeschlossenen Zeichen. Ein paar von Zeichen mit einem Minuszeichen dazwischen ist ein Bereich. Dieser Bereich enthält alle Zeichen die nach lexikalischer Ordnung zwischen den beiden begrenzenden Zeichen angeordnet sind. Wenn das erste Zeichen ein Ausrufezeichen (!) oder ein Caret (^) ist, paßt das Muster auf alle Zeichen die nicht eingeschlossen sind. Das Minuszeichen oder die eckige Klammer selbst kann in den Bereich eingeschlossen werden, indem es zusätzlich als einzelnes Zeichen vor oder nach dem Bereich angegeben wird.

Wenn die Option `braceexpand` gesetzt ist, werden in geschweiften Klammern eingeschlossene, durch Komma getrennte Listen von Wortteilen zu einer Liste von Worten erweitert, wie in der C-Shell.

Zum Beispiel wird `*.{c,h}` zu `*.c *.h` erweitert.

Quoten-Reduktion

Nach allen Erweiterungen werden die unquotierten Escape-Zeichen und Apostrophe entfernt.

1.4.10 Ein/Ausgabe Umleitung

Bevor eine Kommandozeile ausgeführt wird, können die Eingabe- und Ausgabekanäle umgelenkt werden. Auf diese Weise können Dateien zum Lesen bzw. Schreiben für das Kommando geöffnet werden, die nach Beendigung des Kommandos automatisch von der Shell wieder geschlossen werden.

Wenn mehrere Kanalumlenkungen in einer Kommandozeile auftauchen, werden sie der Reihe nach von links nach rechts ausgewertet.

Wenn in einer der folgenden Beschreibungen die Kanalnummer einer Datei nicht angegeben wird, so wird bei einer Eingabeumleitung die Standardeingabe (Kanal 0) und bei einer Ausgabeumleitung die Standardausgabe (Kanal 1) umgeleitet.

Das auf den Umleitungsoperator folgende Wort wird allen möglichen Parametererweiterungen unterworfen. Wenn durch die Erweiterung mehr als ein Wort entsteht, wird eine Fehlermeldung ausgegeben.

Mit Hilfe des `exec` Shellkommandos kann auch die Eingabe/Ausgabe der aktiven Shell umgelenkt werden, indem `exec` ohne Kommando aber mit entsprechenden Umleitungen aufgerufen wird (siehe beim `exec` Shellkommando, Seite 34).

Wenn in einem Kommando mehrere Umleitungen gelegt werden, ist die Reihenfolge signifikant. Ein Beispiel hierfür ist bei der Verdoppelung der Dateikennung auf Seite 23 gegeben.

Eingabeumleitung

[n]< Wort

erzeugt eine Eingabeumleitung von der mit dem (erweiterten) Wort bezeichneten Datei auf den Kanal mit der Nummer n, oder auf die Standardeingabe (Kanal 0) wenn keine Zahl n angegeben wird.

Ausgabeumleitung

[n]> Wort

lenkt den Ausgabekanal mit der Nummer *n* auf die mit dem Wort bezeichnete Datei um. Wenn keine Zahl für die Kanalnummer angegeben ist, wird die Standardausgabe (Kanal 1) angenommen.

Wenn die angegebene Datei nicht existiert, wird sie erzeugt. Wenn eine Datei dieses Namens existiert, wird sie überschrieben, solange die Umgebungsvariable `noclobber` nicht gesetzt ist.

Anfügen der Ausgabe an eine existierende Datei

[n]>> Wort

öffnet die Datei mit dem angegebenen Namen zum anhängen von Daten. Die Daten aus dem Ausgabekanal mit der Nummer *n* werden an die Datei angehängt. Wenn die Kanalnummer fehlt, wird die Standardausgabe umgelenkt.

Wenn die Datei nicht existiert, wird sie erzeugt.

Zusammenfassung der Standardausgabe mit der Standardfehlerausgabe

&> Wort oder

>& Wort

legt die Kanäle für die Standardausgabe und die Standardfehlerausgabe zusammen, und schreibt sie in eine Datei namens *Wort*. Von den beiden Formen sollte die erste bevorzugt werden.

Shellscript-Dokumente

<<[-] Wort

Dokument

Begrenzer

Diese Art der Kanalumlenkung wird benutzt, um einen Teil des Shellscrips direkt als Standardeingabe für ein Kommando zu benutzen. Es werden alle Zeilen des Dokumentes gelesen und als Eingabe an das Kommando weitergeleitet, bis der Begrenzer in einer Zeile auftaucht. Das in der ersten Zeile festgelegte Wort wird keiner Erweiterung unterzogen. Wenn es aber irgendeine Art der Quotierung enthält, ist der Begrenzer das Wort ohne die Quotierung. In diesem Fall wird aber das Dokument ohne jede Erweiterung an das Kommando weitergegeben. Anderenfalls (wenn das Wort keine Quotierung enthält) werden alle Parameter im Dokumenttext erweitert, und es wird die Kommandosubstitution durchgeführt.

In dem zweiten Fall wird ein durch das Escape-Zeichen ‘\’ eingeleitetes Zeilenende ignoriert. Um das Escape-Zeichen selbst, sowie die Zeichen ‘\$’ und ‘ ’ im Dokument darstellen zu können, müssen sie ebenfalls durch ein Escape-Zeichen eingeleitet werden.

Wenn das optionale Minuszeichen bei der Einleitung des Shellscript-Dokumentes auftaucht, werden alle Leerzeichen und Tabulatoren am Anfang der Dokumentzeilen ignoriert. Dadurch kann das Dokument durch Einrückung deutlich von dem übrigen Shellscript abgesetzt werden, ohne daß diese Einrückung auch in der Ausgabe erscheint.

Verdoppelung der Dateikennung

[n]<& Wort

kopiert die in *Wort* enthaltene (ganzzahlige) Eingabedateikennung. Es wird die neue Dateikennung *n* als Eingabekanal erzeugt, oder eine existierende Kennung überschrieben.

Wenn im *Wort* anstelle einer Zahl ein ‘-’ steht, wird der Kanal *n* geschlossen. Wenn das Wort leer ist, wird es durch die Standardeingabe ersetzt.

[n]>& Wort

verdoppelt die Ausgabedateikennung in *Wort*. Wenn das Wort leer ist wird hier die Standardausgabe eingesetzt. Ansonsten ist die Funktion die gleiche wie bei der Verdoppelung der Eingabekennung.

Ein **Beispiel** soll die Funktion der Verdoppelung verdeutlichen:

Die Konstruktion

```
ls > inhalt 2>&1
```

lenkt die Standardausgabe und die Standardfehlerausgabe in die Datei ‘inhalt’ um.

Die Konstruktion

```
ls 2>&1 > inhalt
```

lenkt dagegen nur die Standardausgabe in die Datei ‘inhalt’ um.

Beim ersten Beispiel wird zuerst die Standardausgabe in die Datei umgelenkt, und danach wird der Standardausgabekanal verdoppelt, und dabei die Standardfehlerausgabe ersetzt. Im zweiten Beispiel wird zuerst die Standardfehlerausgabe durch die Standardausgabe ersetzt. Das ist in diesem Moment aber noch der Bildschirm. Erst danach wird die Standardausgabe mit der Datei verbunden. Die Kopie des Standardausgabekanals (die Standardfehlerausgabe) wird von dieser Umlenkung aber nicht betroffen!

Öffnen einer Datei zum lesen und schreiben

`[n]<> Wort`

öffnet die im *Wort* benannte Datei zum lesen und schreiben.

1.4.11 Scriptfunktionen

Scriptfunktionen, wie sie im Abschnitt über die Shellgrammatik eingeführt wurden, sind Teile eines Shellscripts oder einer Initialisierungsdatei für eine interaktive Shell, die komplett in der Shellumgebung gespeichert werden. Wenn ein Kommando in der Kommandozeile mit einer solchen Funktion übereinstimmt, wird der unter dem Funktionsnamen gespeicherte Teil des Shellscripts ausgeführt. Es wird dazu kein neuer Prozeß erzeugt.

Wenn die Scriptfunktion mit Argumenten aufgerufen wird, werden diese Argumente der Funktion als Positionsparameter übergeben. Der Spezialparameter ‘#’ wird aktualisiert. Der Positionsparameter ‘0’ bleibt allerdings unverändert.

Mit dem `local` Shellkommando ist es möglich, lokale Variable für Scriptfunktionen zu erzeugen. Normale Shellvariable sind “global”, sind also in der gesamten Shell uneingeschränkt sichtbar.

Wenn in einer Scriptfunktion das Shellkommando `return` auftaucht, wird die Funktion beendet und die dem Aufruf folgende Zeile des Shellscripts oder die interaktive Eingabe fortgesetzt. Die Positionsparameter und der Spezialparameter ‘#’ werden auf ihre Werte vor dem Funktionsaufruf zurückgesetzt.

Eine Liste der definierten Scriptfunktionen erhält man in einer interaktiven Shell mit der ‘-f’ Option zu den Shellkommandos `declare` oder `typeset`. Die Scriptfunktionen werden automatisch an alle Unterschells weitergereicht (und stehen nur dann auch in diesen Shells zur Verfügung), wenn sie mit der Shellfunktion `export` unter der Option ‘-f’ für den Export bestimmt wurden.

Scriptfunktionen können rekursiv aufgerufen werden. Es gibt keine Begrenzung für die Anzahl der rekursiven Anrufe.

1.4.12 Synonyme

Die Bash unterhält eine Liste von Synonymen, die mit den Shellkommandos `alias` und `unalias` verwaltet werden kann. Das erste Wort jeder Kommandozeile wird daraufhin untersucht, ob es mit einem Synonym übereinstimmt. Wenn das der Fall ist, wird das Wort durch das Synonym ersetzt. Der Synonymname muß ein gültiger Name sein, der Ersetzungstext muß den Regeln der Shellgrammatik entsprechen.

Das erste Wort des Ersetzungstextes wird wieder auf ein Synonym untersucht. Es wird aber das selbe Synonym nicht rekursiv ersetzt, sodaß ein bereits existierendes Kommando über ein Synonym unter dem normalen Namen mit speziellen Optionen aufgerufen werden kann. Beispielsweise kann mit ‘`alias ls='ls -F'`’ bei jedem Aufruf von `ls` die Option ‘-F’ automatisch gesetzt werden.

Wenn das letzte Zeichen des Ersetzungstextes ein Blank ist, wird das dem Synonym folgende Wort wieder auf ein Synonym untersucht.

Es gibt keine Möglichkeit, in dem Ersetzungstext Argumente zu verwenden. Wenn Argumente gebraucht werden, sollte anstelle eines Synonyms eine Scriptfunktion benutzt werden.

Synonyme werden mit dem `alias` Shellkommando erzeugt und aufgelistet. Mit dem `unalias` Shellkommando werden sie wieder gelöscht. Siehe dazu auch die Kommandobeschreibungen auf den Seiten 32 und 42.

Synonyme sollten nicht innerhalb von Scriptfunktionen definiert werden. Außerhalb der Scriptfunktion definierte Synonyme können aber durchaus in der Scriptfunktion verwendet werden.

1.4.13 Hintergrundprozesse

Nachdem die Kommandozeile von der Shell bearbeitet ist, werden die darin enthaltenen Kommandos ausgeführt. Ein Kommando oder eine Pipeline wird dann zu einem Job. Dieser Job läuft weitgehend unabhängig von der Shell; trotzdem steht die Shell weiterhin mit allen Jobs in Verbindung, und kann über Signale mit ihnen kommunizieren. Zu diesem Zweck unterhält die Bash eine Tabelle aller aktuellen Jobs.

Wenn ein Job mit dem `&` Zeichen im Hintergrund gestartet wird, gibt die Bash eine Zeile mit der Jobnummer und der Prozeßnummer für diesen Job aus. Wenn ein Hintergrundjob beendet ist, wird wieder eine Meldung mit dem Namen, der Jobnummer und dem Status des Jobs ausgegeben.

Mit der Tastenkombination `^Z` ist es jederzeit möglich, einen im Vordergrund laufenden Job anzuhalten. Es erscheint dann eine Meldung mit der Jobnummer und dem Namen des angehaltenen Jobs. Danach erscheint die Eingabeaufforderung der Shell.

Ein angehaltener Job kann mit dem `bg` oder dem `fg` Shellkommando im Hintergrund oder im Vordergrund gestartet werden. Mit dem `kill` Kommando wird er abgebrochen. Siehe dazu auch die Kommandobeschreibungen auf den Seiten 32, 35 und 36.

Im Zusammenhang mit den genannten Kommandos kann ein Job mit seiner Jobspezifikation, und seiner Prozeßnummer angesprochen werden.

Als **Jobspezifikation** werden die Jobnummer oder der Anfang des Jobnamens erkannt, indem sie durch ein Prozentzeichen `%` eingeleitet werden. Die Jobnummer ist die laufende Nummer unter der der Job in der Jobtabelle geführt wird. Der zuletzt angehaltene Job wird auch als aktueller Job bezeichnet, und kann mit `%%` angesprochen werden. Der zuvorletzt angehaltene Job wird auch als der letzte Job bezeichnet und kann mit `%-` benannt werden. Wenn ein angegebener Anfang auf mehrere Jobs paßt, wird eine Fehlermeldung ausgegeben. Ein Job kann auch einfach durch Angabe seiner Jobspezifikation (ohne Kommando) aus dem Hintergrund in den Vordergrund geholt werden. Zum Beispiel bringt `%1` den angehaltenen oder im Hintergrund laufenden Prozess mit der Jobnummer 1 im Vordergrund zu laufen, so wie `fg %1`.

Bei der Ausgabe der Jobtabelleneinträge wird der aktuelle Job mit einem `+` gekennzeichnet, und der letzte Job mit einem `-`.

Wenn die Shell beendet werden soll, während sich angehaltene Jobs im Hintergrund befinden, wird eine Warnung ausgegeben, und die Shell nicht beendet. Erst wenn die Shell unmittelbar darauf, oder nach einem einzigen `jobs` Shellkommando, ein zweites Mal beendet wird, werden alle verbleibenden Jobs automatisch terminiert.

Wenn zum Zeitpunkt des ausloggens noch Jobs im Hintergrund laufen werden diese Jobs beim Verlassen der Shell automatisch an den `init` Prozess übereignet, und laufen so weiter.

1.4.14 Signale

Wenn die Bash interaktiv arbeitet, ignoriert sie `SIGTERM` und `SIGQUIT`. `SIGINT` wird für das `trap` Shellkommando abgefangen aber, nicht von der Shell selbst bearbeitet (→ Seite 41).

Die Signale zur Jobkontrolle `SIGTTIN`, `SIGTTOU` und `SIGTSTP` werden von der Shell selbst ebenfalls ignoriert.

Mit dem `suspend` Shellkommando kann die Shell bis zu einem `SIGCONT` Signal angehalten werden.

Hintergrundjobs ignorieren die Signale `SIGINT` und `SIGQUIT`.

Jobs, die durch Kommandosubstitution von der Shell aufgerufen wurden, reagieren nicht auf die Jobkontrollsignale `SIGTTIN`, `SIGTTOU` und `SIGTSTP`.

1.4.15 Kommandoausführung

Nachdem die Kommandozeile in Worte zerteilt ist, werden für ein einfaches Kommando folgende Schritte ausgeführt:

Wenn der Kommandoname keine Schrägstriche ‘/’ enthält (also kein Verzeichnis angegeben ist), versucht die Shell das Kommando zu lokalisieren. Dazu wird der Kommandoname zuerst in der Hashtabelle gesucht, dann mit den Synonymen, dann mit den Scriptfunktionen und dann mit den Shellfunktionen verglichen. Das erste passende Kommando wird ausgeführt.

Wenn keiner dieser Namen paßt, werden alle in der PATH Shellvariablen aufgeführten Verzeichnisse nach einem Kommando dieses Namens durchsucht. Wenn auch hier kein passendes Kommando gefunden wurde, gibt die Shell eine Fehlermeldung aus.

Wenn der Kommandoname mindestens einen Schrägstrich ‘/’ enthält, wird der Name als Pfadname interpretiert, und dieses Kommando ausgeführt, das genau auf diesem Pfad liegt. Wenn es dieses Kommando nicht gibt, wird wieder eine Fehlermeldung ausgegeben.

Wenn ein passendes externes Kommando gefunden wurde, wird die Positionsvariable 0 mit dem vollständigen Pfadnamen des Kommandos belegt, und die übrigen Argumente (wenn welche vorhanden sind) werden in den Argumentvektor geschrieben, der dann von dem Kommando bearbeitet werden kann.

Wenn die Ausführung fehlschlägt, weil die Datei keine ausführbaren Binärdaten enthält, versucht die Bash automatisch diese Datei als Shellsript auszuführen. Dazu wird eine neue Shell (Subshell) erzeugt, die wie bei einem Aufruf in der Kommandozeile neu initialisiert wird.

Wenn die Datei mit den Zeichen ‘#!’ beginnt, wird der Rest der ersten Zeile als Programm zur Interpretation der Datei aufgerufen. (Diese Funktion wird aber schon vom Kernel ausgeführt)

1.4.16 Umgebung

Wenn ein Kommando von der Bash ausgeführt wird, erhält es eine Reihe von Shellvariablen als “Umgebung”.

Die Variablen, die die Shell nach dem Start in ihrer eigenen Umgebung gefunden hat, werden automatisch an alle Kinder dieser Shell weitergegeben. Zusätzlich können beliebige Variable mit der `export` oder der ‘`declare -x`’ Shellfunktion erzeugt und für den Export bestimmt werden.

1.4.17 Eingabeaufforderung

Wenn die Shell interaktiv arbeitet, und auf die Eingabe einer neuen Kommandozeile wartet, signalisiert sie ihre Bereitschaft durch die Ausgabe eines Zeichens oder einer kurzen Meldung. Diese Meldung wird auch als **Prompt** bezeichnet. Sie kann durch den Anwender selbst gestaltet werden. Dazu muß die gewünschte Meldung in die Shellvariable PS1 oder PS2 geschrieben werden.

Die Shell benutzt zwei Prompts für Eingabeaufforderungen. Der erste erscheint bei jeder interaktiven Eingabeaufforderung. Der zweite erscheint, wenn die Shell zur Vervollständigung eines vorangegangenen Kommandos noch weitere Eingaben vom Benutzer erwartet.

Für die Gestaltung des Prompts stehen die folgenden Sonderzeichen zur Verfügung:

`\t` die aktuelle Zeit

`\d` das aktuelle Datum

`\n` ein Zeilenende

`\s` der Name der aktuellen Shell (Inhalt vom Parameter 0)

`\w` das aktuelle Verzeichnis

`\W` auch das aktuelle Verzeichnis

`\u` der Benutzername

`\h` der Hostname (Netzwerkname des Rechners)

`\#` die (absolute) Nummer des aktuellen Kommandos

`\!` die Nummer unter der das aktuelle Kommando im Kommandozeilenspeicher geführt wird

`\$` der “Standardprompt”. Ein `$` Zeichen für normalsterbliche Anwender, ein `#` für den Superuser (root)

`\nnn` das Zeichen mit dem (oktalen) Code `nnn`

`\\` der Backslash ‘\’

1.4.18 Kommandozeileneditor

Wenn die Shell im interaktiven Betrieb auf die Eingabe wartet und den Prompt ausgegeben hat, ist der Kommandozeileneditor aktiv. Der Kommandozeileneditor benutzt die Funktionen der ‘readline’ Bibliothek. In dem folgenden Abschnitt werden die Editorbefehle für den **emacs** Stil des Kommandozeileneditors erklärt. Mit dem Shellkommando ‘`set -o vi`’ kann der Editor auch in den **vi** Stil umgeschaltet werden.

Die Editorbefehle benutzen zwei Sondertasten: die **CONTROL**- und die **Metataste**. Die **CONTROL**-taste ist auf der PC Tastatur in der linken unteren Ecke (manchmal **STRG**). Eine **Metataste** ist unter diesem Namen in der Regel nicht vorhanden. Meistens wird die linke **ALT**-taste mit dieser Funktion belegt. Wenn keine Taste mit der Metafunktion belegt ist, kann die Escapetaste (**ESC**) benutzt werden. Während **control** und **ALT** gemeinsam mit dem Buchstaben gedrückt werden müssen, wird **ESC** vor dem Buchstaben gedrückt (zwei Anschläge).

In den Erklärungen wird **CONTROL** mit ‘**C-**’ und die **Metataste** mit ‘**M-**’ gekennzeichnet. In manchen Fällen müssen die **CONTROL** und **ALT** zusammen gedrückt werden.

Zusätzlich können alle Funktionen des Editors auch vom Benutzer mit anderen Tasten belegt werden. Diese Belegung kann in einer Datei `~/inputrc` im Heimatverzeichnis des Anwenders festgehalten werden. Diese Datei wird von der **bash** beim Start gelesen.

Für jede Umbelegung muß eine Zeile in der Datei eingetragen werden. Das Format ist

Taste:Kommandobezeichnung

Die Kommandobezeichnungen sind bei den Beschreibungen angegeben.

Außerdem können die folgenden Tasten benannt werden:

RUBOUT für Backspace

DEL für Delete

ESC für Escape

SPACE für Leerzeichen

NEWLINE für Zeilenende

RETURN für Wagenrücklauf

LFD für Zeilenvorschub

TAB für Tabulator

Die aktuelle Tastaturbelegung wird mit dem Kommando ‘**bind -v**’ ausgegeben (siehe auch beim Shellkommando **bind** auf Seite 32).

Positionieren der Einfügemarke

Zeilenanfang (C-a) setzt die Einfügemarke an den Anfang der Kommandozeile. Die Kommandobezeichnung ist **beginning-of-line**.

Zeilenende (C-e) setzt die Einfügemarke an das Ende der Kommandozeile. Die Kommandobezeichnung ist **end-of-line**.

Zeichen vorwärts (C-f) Setzt die Einfügemarke ein Zeichen nach rechts. Diese Funktion ist auch mit der rechten Pfeiltaste im Cursorblock erreichbar. Die Kommandobezeichnung ist **forward-char**.

Zeichen rückwärts (C-b) Setzt die Einfügemarke ein Zeichen nach links. Diese Funktion ist auch mit der linken Pfeiltaste im Cursorblock erreichbar. Die Kommandobezeichnung ist `backward-char`.

Wort vorwärts (M-f) Setzt die Einfügemarke an das Ende des aktuellen Wortes, oder ein Wort weiter (wenn die Einfügemarke zwischen zwei Worten steht) Die Kommandobezeichnung ist `forward-word`.

Wort rückwärts (M-b) Setzt die Einfügemarke an den Anfang des aktuellen Wortes oder des vorhergehenden Wortes (Wenn die Einfügemarke zwischen zwei Worten steht) Die Kommandobezeichnung ist `backward-word`.

Bildschirm löschen (C-l) löscht alle Zeichen vom Bildschirm. Die Einfügemarke erscheint danach auf der ersten Zeile. Die Kommandobezeichnung ist `clear-screen`.

Benutzung des Kommandozeilenspeichers

Abschluß des Kommandozeileneditors (NEWLINE, RETURN) der Kommandozeileneditor wird beendet, und die gesamte Kommandozeile von der Shell ausgewertet. Die Position der Einfügemarke zum Zeitpunkt des Abschlusses spielt keine Rolle. Wenn die Zeile nicht leer ist, wird sie an den Kommandozeilenspeicher angehängt. Wenn die Shellvariable `history_control` das Wort `'ignorespace'` enthält, werden Kommandozeilen, die ein Leerzeichen als erstes Zeichen haben nicht gespeichert. Enthält die Variable das Wort `'ignoredups'` werden nur die Kommandozeilen gespeichert, die sich von der zuletzt gespeicherten unterscheiden. Die Kommandobezeichnung ist `accept-line`.

Kommando zurück (C-p) wandert im Kommandozeilenspeicher eine Position zurück und gibt die komplette Zeile zum editieren auf den Bildschirm. Diese Funktion ist auch an die 'hoch' Taste des Cursorblocks gebunden. Die Kommandobezeichnung ist `previous-history`.

Kommando vorwärts (C-n) wandert im Kommandozeilenspeicher eine Position Vorwärts, und gibt die komplette Zeile zum editieren auf den Bildschirm. Diese Funktion ist auch über die 'runter' Taste des Cursorblocks zu erreichen. Die Kommandobezeichnung ist `next-history`.

zum ersten Kommando (M-<) holt die erste Kommandozeile aus dem Kommandozeilenspeicher in den Editor. Die Kommandobezeichnung ist `beginning-of-history`.

zum letzten Kommando (M->) holt die letzte Kommandozeile aus dem Kommandozeilenspeicher in den Editor. Die Kommandobezeichnung ist `end-of-history`.

Kommando rückwärts suchen (C-r) sucht rückwärts inkrementell im Kommandozeilenspeicher nach einer Kommandozeile mit bestimmtem Anfang. Der gesuchte Anfang wird interaktiv eingegeben. Die Kommandobezeichnung ist `reverse-search-history`.

Kommando vorwärts suchen (C-s) sucht vorwärts im Kommandozeilenspeicher von der aktuellen Zeile an nach einer Kommandozeile mit bestimmtem Anfang. Die Kommandobezeichnung ist `forward-search-history`.

Kommandozeile erweitern (M-C-e) erweitert die Kommandozeile (nur History- und Aliasexpansion) Die Kommandobezeichnung ist `expand-line`.

letztes Argument einfügen (M-., M-_) fügt das letzte Argument des letzten Kommandos an der Einfügemarke ein. Die Kommandobezeichnung ist `insert-last-argument`.

Änderungen des Textes

Zeichen löschen (C-d) Die Kommandobezeichnung ist `delete-char`.

letztes Zeichen löschen (BACKSPACE) Die Kommandobezeichnung ist `backward-delete-char`.

Fluchtsymbol (C-q, C-v) wird benutzt um Control-Zeichen in die Kommandozeile zu schreiben, die normalerweise durch den Editor abgefangen und bearbeitet werden. Die Kommandozeichnung ist `quoted-insert`.

Tabulator einfügen (M-TAB) schreibt ein TAB in die Kommandozeile. Die Kommandozeichnung ist `tab-insert`.

Text einfügen (a, b, A, 1, !, ...) schreibt die Tastatursymbole (Buchstaben) in die Kommandozeile, wie sie von der Tastatur gelesen werden. Hinter dieser etwas verklausulierten Formulierung verbirgt sich die Eingabe normalen Textes. Die Kommandozeichnung ist `self-insert`.

zwei Zeichen vertauschen (C-t) vertauscht das Zeichen vor der Einfügemarke mit dem Zeichen unter der Einfügemarke. die Einfügemarke wandert außerdem um eine Stelle weiter. Wenn die Einfügemarke am Zeilenende angekommen ist, werden die beiden Zeichen vor der Einfügemarke vertauscht. Die Kommandozeichnung ist `transpose-chars`.

zwei Worte vertauschen (M-t) vertauscht das Wort unter der Einfügemarke mit dem Wort vor der Einfügemarke, bzw. das Wort nach der Einfügemarke mit der Wort vor der Einfügemarke. Die Einfügemarke steht danach hinter dem letzten der vertauschten Worte. Wenn die Einfügemarke bereits am Ende der Kommandozeile steht, werden die beiden Worte vor der Einfügemarke vertauscht. Die Kommandozeichnung ist `transpose-words`.

GROßBUCHSTABEN (M-u) wandelt alle Buchstaben des Wortes von der Einfügemarke an in Großbuchstaben um. Die Kommandozeichnung ist `upcase-word`.

kleinbuchstaben (M-l) wandelt alle Buchstaben des Wortes von der Einfügemarke an in Kleinbuchstaben um. Die Kommandozeichnung ist `downcase-word`.

Großschreibung (M-c) Wandelt den Buchstaben unter der Einfügemarke oder den ersten Buchstaben des folgenden Wortes in Großbuchstaben um, die folgenden Buchstaben bis zum Wortende alle in Kleinbuchstaben. Anschließend steht die Einfügemarke hinter dem umgewandelten Wort. Die Kommandozeichnung ist `capitalize-word`.

Ausschneiden und Einfügen

bis zum Zeilenende ausschneiden (C-k) löscht die komplette Kommandozeile von der Einfügemarke an, und speichert sie im Ausschneideringspeicher. Die Kommandozeichnung ist `kill-line`.

vom Zeilenanfang ausschneiden (ohne Belegung) schneidet den Anfang der Kommandozeile bis zur Einfügemarke aus. Dieses Kommando ist normalerweise nicht mit einer Tastenkombination verbunden. Die Kommandozeichnung ist `backward-kill-line`.

Rest des Wortes ausschneiden (M-d) schneidet von der Einfügemarke an ein Wort aus, und speichert es im Ringspeicher. Als Wort werden hier alle Zeichen von der Einfügemarke an bis zum nächsten Sonderzeichen betrachtet. Eine beliebig lange Folge von Sonderzeichen unter oder unmittelbar nach der Einfügemarke wird nicht als Worttrenner behandelt. Die Kommandozeichnung ist `kill-word`.

Anfang des Wortes ausschneiden (M-BACKSPACE) schneidet alle Zeichen vorder Einfügemarke bis zum nächsten Sonderzeichen aus, und speichert es im Ringspeicher. Das Zeichen unter der Einfügemarke wird nicht ausgeschnitten. Eine beliebig lange Folge von Sonderzeichen unmittelbar vor der Einfügemarke wird nicht als Worttrenner behandelt. Die Kommandozeichnung ist `backward-kill-word`.

vom Zeilenanfang bis zur Einfügemarke ausschneiden (C-u) schneidet den Anfang der Kommandozeile bis zur Einfügemarke aus, und speichert ihn im Ringspeicher. Das Zeichen unter der Einfügemarke wird nicht ausgeschnitten. Die Kommandozeichnung ist `unix-line-discard`.

Anfang des Wortes ausschneiden (C-w) schneidet alle Zeichen vor der Einfügemarke bis zum nächsten Leerzeichen aus und speichert ihn im Ringspeicher. Die Kommandozeichnung ist `unix-word-rubout`.

ausgeschnittenes Stück einfügen (C-y) fügt das zuletzt in den Ausschneideringspeicher geschriebene Stück vor der Einfügemarke ein. Das Stück wird dabei aus dem Ringspeicher entfernt. Die Kommando- bezeichnung ist `yank`.

Ausschneideringspeicher rotieren (M-y) ersetzt das zuletzt eingefügte Stück durch das vor diesem Stück in den Ringspeicher eingefügte Stück der Kommandozeile. Das zuvor ersetzte Stück wird wieder in den Ringspeicher eingefügt, das aktuell ersetzte Stück daraus entfernt. Die Kommando- bezeichnung ist `yank-pop`.

Argumente

numerisches Argument (M-0, M-1, ... , M-—) wird als numerisches Argument für das folgende Editorkommando benutzt. In der Regel wird durch dieses Argument die Anzahl der Wiederholungen dieses Editorkommandos bestimmt. Die Kommando- bezeichnung ist `digit-argument`.

universelles Argument (Nicht Belegt) entspricht dem gleichnamigen Argument von emacs. Es ist allerdings keiner Tastenkombination zugeordnet. Die Kommando- bezeichnung ist `universal-argument`.

Automatische Erweiterung von Kommando- und Dateinamen

erweitern (TAB) versucht die bis zu dem Editorbefehl geschriebene Kommandozeile sinnvoll zu ergänzen. Das geschieht, indem die Zeichenfolge des letzten Wortes bis zur Einfügemarke mit den Namen von Kommandos, Dateien und Verzeichnissen verglichen wird, und das Wort soweit ergänzt wird, wie eine eindeutige Zuordnung möglich ist. Das erste Wort eines einfachen Kommandos wird dabei nur mit den Kommandonamen in den PATH Verzeichnissen verglichen, die weiteren Worte eines einfachen Kommandos werden dagegen nur noch mit Datei- und Verzeichnisnamen verglichen. In einigen Fällen (z.B. bei Pipelines) werden auch die ersten Worte der folgenden Kommandos auf diese Weise richtig zugeordnet.

Wenn ein Wort mehreren bekannten Namen zugeordnet werden kann, wird es nur soweit ergänzt, wie sich die Namen nicht unterscheiden. Wenn dann ein zweites mal das TAB Editorkommando gegeben wird, werden alle erkannten Möglichkeiten angezeigt.

Die Kommando- bezeichnung ist `complete`.

mögliche Erweiterungen (M-?) zeigt alle als sinnvoll erkannten Erweiterungen an. Als sinnvoll gilt hier wieder ein Kommandoname als erstes Wort eines einfachen Kommandos, und ein Datei- oder Verzeichnisname für jedes weitere Wort. Die Kommando- bezeichnung ist `possible-completions`.

nur Dateinamenerweiterung (M-/) vergleicht das Wort bis zur Einfügemarke nur mit Date- und Verzeichnisnamen. Werden passende Namen gefunden, findet eine Erweiterung wie beim TAB Editorkommando statt. Die Kommando- bezeichnung ist `complete-filename`.

mögliche Dateinamen (C-x /) zeigt alle als sinnvoll erkannten Datei- und Verzeichnisnamen, ohne jedoch eine Erweiterung auszuführen. Die Kommando- bezeichnung ist `possible-filename-completions`.

Benutzernamenerweiterung (M-~) versucht das Wort bis zur Einfügemarke zu einem Benutzernamen zu erweitern. Die Kommando- bezeichnung ist `complete-username`.

mögliche Benutzernamen (C-x ~) zeigt alle möglichen Benutzernamen, auf die das Wort bis zur Einfügemarke paßt. Die Kommando- bezeichnung ist `possible-username-completions`.

Variablenerweiterung (M-\$) versucht das Wort bis zur Einfügemarke zu einem Variablennamen zu erweitern. Die Kommando- bezeichnung ist `complete-variable`.

mögliche Variable (C-x \$) zeigt alle möglichen Variablennamen an, auf die das Wort bis zur Einfügemarke paßt. Die Kommando- bezeichnung ist `possible-variable-completion`.

nur Hostnamenerweiterung (M-@) erweitert das Wort unter der Einfügemarke zu einem Hostnamen. Dieser Name wird aus der Datei /etc/hosts genommen, wenn in der Shellvariablen `hostname_completion_file` keine andere Datei angegeben ist. Die Kommandozeichnung ist `complete-hostname`.

mögliche Hostnamen (C-x @) zeigt alle möglichen Hostnamenerweiterungen. Die Kommandozeichnung ist `possible-hostname-completions`.

Verschiedenes

Abbruch (C-g) bricht ein Editorkommando ab. Die Kommandozeichnung ist `abort`.

Zweite Kommandoebene (M-a, M-b, ...) erlaubt eine Doppelbelegung der Tastatur mit Editorkommandos. Die Kommandozeichnung ist `do-uppercase-version`.

Metazeichen (ESC) veranlaßt die Shell, das nächste Zeichen als Metazeichen zu interpretieren. (Wichtig, wenn keine Meta-Taste auf der Tastatur ausgewiesen ist. Bei einer PC-Tastatur ist meist die linke `alt` Taste mit der Meta-Funktion belegt!) Die Kommandozeichnung ist `prefix-meta`.

Kommando zurück (C-_) nimmt das zuletzt ausgeführte Editorkommando zurück. Es werden alle für eine Kommandozeile gegebenen Editorkommandos gespeichert und sind auf diese Weise nacheinander reversibel. Die Kommandozeichnung ist `undo`.

alles zurück (M-r) stellt den ursprünglichen Zustand der Kommandozeile wieder her, nimmt also alle Editorkommandos zurück. Die Kommandozeichnung ist `revert-line`.

1.4.19 Kommandozeilenspeicher

Die Bash benutzt einen Kommandozeilenspeicher mit ähnlichen Funktionen wie der History Mechanismus bei der C-Shell.

Es ist möglich, einzelne Worte in bestimmten Kommandozeilen im Kommandozeilenspeicher in die aktuelle Kommandozeile zu integrieren. Dazu wird zuerst eine Kommandozeile ausgewählt, und danach ein Wort in dieser Kommandozeile angesprochen

Bezugnahme auf eine frühere Kommandozeile

Zum Auswählen einer Kommandozeile im Kommandozeilenspeicher gibt es verschiedene Möglichkeiten. Prinzipiell leitet ein Ausrufezeichen `!` die Kommandozeilensubstitution ein, außer wenn das Ausrufezeichen nach einem Escape-Zeichen `\` steht, oder von einem `SPACE`, `RETURN`, `TAB`, `=`, oder `(` gefolgt wird.

Anstelle des Ausrufezeichens kann in der Shellvariablen `histchar` auch ein anderes Zeichen mit dieser Funktion belegt werden.

!! wählt die letzte Zeile im Kommandozeilenspeicher

!n wählt die Zeile Nummer *n*

!-n wählt die aktuelle Zeile minus *n*

!Zeichenkette wählt die letzte Kommandozeile, deren Anfang mit der *Zeichenkette* übereinstimmt.

!?Zeichenkette[?] wählt die letzte Kommandozeile, in der die *Zeichenkette* an irgendeiner Position vorkommt.

Bezugnahme auf ein Wort einer früheren Kommandozeile

Auf die Kommandozeilenreferenz kann eine Wortreferenz folgen. Diese Wortreferenz wird durch einen Doppelpunkt ‘:’ von der Zeilenreferenz getrennt. Wenn die Wortreferenz mit einem ‘^’, ‘\$’, ‘*’ oder ‘%’ beginnt kann der Doppelpunkt auch weggelassen werden. Die Worte einer Kommandozeile sind vom Zeilenanfang an mit Null beginnend nummeriert.

n das *n*te Wort. Das Nullte Wort ist in der Regel der Kommandoname

^ (Caret) das erste Argument (das ist Wort Nummer 1)

\$ das letzte Argument

% das bei *?Zeichenkette?* gefundene Wort

n-m der Bereich vom *n*ten bis zum *m*ten Wort

***** alle Worte außer das Nullte

Modifikation der bezogenen Kommandozeilen

Nach (oder anstelle) der Wortreferenz kann noch ein oder mehrere Zeichen zur Modifikation des bezogenen Wortes folgen. Diese Zeichen werden wieder durch Doppelpunkte getrennt.

h wie **dirname**, → Seite

r schneidet jede Endung der Form ‘.xxx’ ab

e läßt nur die Endung übrig

t wie **basename**, → Seite 10

p gibt die entstandene Kommandozeile aus, ohne sie auszuführen. Dieser Befehl wird sofort ausgeführt, sollte also der letzte sein.

1.4.20 Rechnen in der Shell

Die Bash erlaubt die Berechnung von Ausdrücken im Zusammenhang mit der arithmetischen Erweiterung und dem **let** Shellkommando. Die Berechnungen werden mit ‘langen Ganzzahlwerten’ wie sie in der Programmiersprache C verwendet werden ausgeführt. Eine Überlaufkontrolle findet nicht statt. Die Division durch Null führt zu einem Fehler und kann mit der **trap** Shellfunktion abgefangen werden.

Die folgenden Operatoren sind erlaubt (die Gruppierung entspricht der Prioritätshierarchie):

– das minus Vorzeichen

! das logische NICHT

*** / %** Multiplikation, Division und Modulo (Rest bei ganzzahliger Division)

+ – Addition und Subtraktion

<= >= < > die Vergleiche

== != gleich und ungleich

= die Zuweisung

Shellvariablen sind als Operanden erlaubt. Das Ganzzahlattribut für die Variable muß nicht gesetzt sein.

Die Operationen werden von links nach rechts (der Reihe nach) ausgeführt. Klammern werden erkannt und vorrangig behandelt.

1.4.21 Eingebaute Shellkommandos

alias

Syntax: **alias** [*Name* [= *Kommando*]. . .]

Mit dem **alias** Shellkommando können einfache Kommandos mit benutzerdefinierten Namen (Synonymen) belegt werden. Wenn ein Name und ein einfaches Kommando angegeben sind, wird der Name als Synonym für das Kommando gespeichert. Besteht das Kommando aus mehreren Worten, muß es in Hochkommata oder Anführungszeichen eingeschlossen werden.

Ohne Argument werden alle Synonyme aufgelistet. Wenn nur ein Name angegeben ist, wird das Synonym für diesen Namen angezeigt. Der Rückgabewert von **alias** ist Null (wahr), wenn der Name als Synonym für ein Kommando steht.

bind

Syntax: **bind** [-ldv] [-f *Dateiname*] [*Tastaturcode: Kommandobezeichnung*]

Mit dem **bind** Shellkommando können Tasten mit neuen oder anderen Editorfunktionen belegt werden. Die Syntax einer Tastaturbelegung entspricht der für die '~/.inputrc' Datei (→ Seite 26).

Die Optionen haben die folgende Bedeutung:

- l gibt eine Liste aller möglichen Kommandobezeichnungen aus
- v gibt eine Liste der belegten Tasten und ihrer Funktionen aus
- d gibt eine Liste aller Kommandobezeichnungen und ihrer Belegungen im Format zum wiedereinlesen auf die Standardausgabe
- f *Datei* liest die Tastaturbelegung aus der *Datei*
- q *Kommandobezeichnung* gibt die Taste zur *Kommandobezeichnung* aus

bg

Syntax: **bg** [*Jobspezifikation*]

Das Kommando **bg** startet einen (mit ^Z) angehaltenen Prozeß (Job) im Hintergrund. Wenn keine Jobspezifikation (→ Seite 24) angegeben wurde, wird der zuletzt angehaltene Job gestartet.

Ein im Hintergrund laufender Job wird automatisch angehalten, wenn er vom Terminal lesen will.

break

Syntax: **break** [*n*]

Das **break** Shellkommando bricht eine **for**, **while** oder **until** Schleife ab. Wenn eine Zahl *n* als Argument angegeben ist, werden *n* verschachtelte Schleifen abgebrochen. Ist die Anzahl größer als die Zahl der umgebenden Schleifen, werden alle umgebenden Schleifen verlassen

builtin

Syntax: **builtin** [*Shellkommando* [*Argument* . . .]]

Das Shellkommando **builtin** führt ein anderes eingebautes Shellkommando aus, auch wenn es durch ein Synonym oder eine gleichnamige Scriptfunktion verdeckt ist. Eine mit 'enable -n' abgeschaltete Shellfunktion kann auch mit dem **builtin** Kommando nicht aufgerufen werden.

bye

Siehe **exit**

cd

Syntax: **cd** [*Verzeichnis*]

Das Shellkommando **cd** setzt das aktuelle *Verzeichnis*. Wenn kein Verzeichnis angegeben ist, wird in das HOME Verzeichnis gewechselt. In der Shellvariablen CDPATH kann eine (durch Doppelpunkt getrennte) Liste von Verzeichnissen angegeben werden, in denen das angegebene Verzeichnis gesucht wird. Ein Verzeichnisname der mit einem '/' beginnt wird als absoluter Name behandelt und nur vom Wurzelverzeichnis aus gesucht.

Das aktuelle Verzeichnis selbst ist unter der Bezeichnung '.' ansprechbar; das nächsttiefere Verzeichnis heißt '..' und das letzte aktuelle Verzeichnis (OLDPWD) heißt '-'.

command

Syntax: **command** [*Kommando* [*Argument* ...]]

Das Shellkommando **command** führt das externe *Kommando* aus und ignoriert Shellfunktionen, Synonyme und Scriptfunktionen.

continue

Syntax: **continue** [*n*]

Mit der Shellfunktion **continue** wird der aktuelle Schleifendurchlauf einer **for**, **while** oder **until** Schleife sofort unterbrochen, und mit dem nächsten Schleifendurchlauf angefangen. Wenn als Argument eine Zahl (größer als Null) angegeben ist, wird diese Anzahl umgebender Schleifen abgebrochen. Wenn die Zahl größer als die Zahl der umgebenden Schleifen ist, werden alle umgebenden Schleifen unterbrochen und mit dem nächsten Durchlauf der äußersten Schleife fortgefahren.

declare

Syntax: **declare** [-frxi] [*Name* [= *Wert*]]

Das Shellkommando **declare** erzeugt eine Shellvariable und/oder setzt die Attribute der Variablen. Wenn kein Name angegeben ist, werden die Werte aller Variablen angezeigt.

Optionen:

-f (zeigt) nur Funktionsnamen

-r setzt die Variable(n) auf nur lesen Status

-x markiert die Variable für automatischen Export in alle Unterschellumgebungen

-i setzt den Typ der Variablen auf Ganzzahl; wenn dieser Variablen ein Wert zugewiesen wird, findet eine arithmetische Auswertung des zugewiesenen Ausdrucks statt

Wenn die Optionen mit '+' anstelle von '-' gesetzt werden, wird das entsprechende Merkmal abgeschaltet.

Wenn die Shellfunktion innerhalb einer Funktionsdefinition benutzt wird, ist die Variable lokal zu dieser Funktion, genauso als ob sie mit der Shellfunktion **local** definiert wäre.

Die **typeset** Shellfunktion ist identisch mit der **declare** Shellfunktion.

dirs

Syntax: **dirs** [-l]

Die **dirs** Shellfunktion gibt eine Liste der im Verzeichnisstapel gespeicherten Verzeichnisse aus. Die Bearbeitung dieses Verzeichnisstapels findet mit den Shellkommandos **pushd** und **popd** statt (→ Seite 37).

Mit der Option '-l' werden die Verzeichnisnamen nicht relativ zum Heimatverzeichnis sondern vom Wurzelverzeichnis aus angezeigt.

echo

Syntax: **echo** [-ne] [*Argument* ...]

Das **echo** Shellkommando gibt die Argumente (durch Leerzeichen getrennt) auf die Standardausgabe. Es gibt auch ein externes **echo** Kommando, das mit dem eingebauten Shellkommando der **bash** identisch ist.

Wenn die Option ‘-n’ gesetzt ist, wird die Ausgabe nicht durch ein Zeilenende Zeichen abgeschlossen.

Wenn die Option ‘-e’ gesetzt ist, werden die folgenden Sonderzeichen zur Formatierung der Ausgabe erkannt:

\a Alarm (Piep)

\b Schritt zurück

\c kein Zeilenende

\f Seitenvorschub

\n Zeilenende

\r Wagenrücklauf

\t (horizontaler) Tabulator

\v vertikaler Tabulator

\nnn das Zeichen mit dem (oktalen) Code *nnn*

enable

Syntax: **enable** [-n] [*Kommando* ...]

Das Shellkommando **enable** ermöglicht es, Shellfunktionen ab- und wieder anzuschalten. Auf diese Weise kann anstelle eines (internen) Shellkommandos das gleichnamige (externe) Kommando aus einem Binärverzeichnis ausgeführt werden.

Wenn die Option ‘-n’ gesetzt ist, wird das Shellkommando abgeschaltet. Sonst wird das Shellkommando eingeschaltet.

eval

Syntax: **eval** [*Argument* ...]

Das **eval** Shellkommando fügt die Argumente zu einer Kommandozeile zusammen, die ausgeführt wird, ohne die Shell zu verdrängen. Das Ergebnis (Status) dieses Kommandos oder Shellkommandos kann dann einer Variablen zugewiesen werden.

exec

Syntax: **exec** [[-] *Kommando* [*Argument* ...]]

Normalerweise startet die Shell ein Programm mit dem *fork* Systemaufruf und wartet im Hintergrund bis das Programm beendet wird. Danach übernimmt die Shell wieder die Kontrolle über das Terminal.

Das Shellkommando **exec** führt ein Kommando mit den angegebenen Argumenten aus, ohne einen Kindprozeß zu erzeugen. Das heißt, die aufrufende Shell wird verdrängt und damit beendet. Auch wenn das Kommando aus irgendwelchen Gründen nicht ausgeführt werden kann, wird die Shell beendet (Wenn die Shellvariable `no_exit_on_failed_exec` nicht gesetzt ist).

Die Argumente werden als Optionen und Positionsparameter an das Kommando weitergegeben. Wenn das erste Argument ein ‘-’ ist, wird dieses Argument als Nulltes (!) Argument der Kommandozeile an das Kommando weitergegeben. Das ist die Art, wie **login** ein Programm aufruft.

Wenn kein Kommandoname angegeben ist, werden die Ein/Ausgabe Umleitungen, die mit dem **exec** Shellkommando gegeben werden, auf die aufrufende Shell angewendet.

exit

Syntax: **exit** [*n*]

Das **exit** Shellkommando verläßt die Shell mit dem Status *n*. Wenn kein Status angegeben ist, wird der Status des zuletzt ausgeführten Kommandos (in der Shellvariablen ‘?’) zurückgegeben.

Die **exit** Shellfunktion erzeugt ein EXIT Signal (0), daß mit dem **trap** Shellkommando abgefangen und als letzte Aktion der Shell behandelt werden kann.

Die **bye** Shellfunktion ist identisch mit der **exit** Shellfunktion.

export

Syntax: **export** [**-nfp**] [*Name* [= *Wert*]]

Bei der Ausführung von Programmen durch die Shell werden in der Regel neue Prozesse erzeugt. Diese Prozesse erhalten eine Umgebung (Environment) in der verschiedene “globale” Variable und Funktionen enthalten sein können. Diese können vom Prozeß ausgewertet und benutzt werden.

Es werden aber nicht alle, sondern nur die besonders für den Export bestimmten Variablen und Funktionen aus der Shellumgebung in die Umgebung eines neuen Prozesses kopiert.

Das **export** Shellkommando markiert die angegebenen Variablen und Funktionen für den Export. Wenn die Option ‘**-n**’ gesetzt ist, wird die Exportbestimmung zurückgenommen. Um Funktionen zu exportieren, muß die ‘**-f**’ Option benutzt werden.

Wenn keine Namen angegeben sind oder die Option ‘**-p**’ gesetzt ist, werden alle für den Export bestimmten Namen mit ihren Werten angezeigt. Wenn zu einem Namen zusätzlich ein Wert angegeben ist, wird eine Variable mit diesem Wert belegt oder erzeugt.

fc

Syntax: **fc** [**-e** *Editor*] [**-n|***r*] [*Anfang*] [*Ende*]
fc **-e** **-** [*Muster=Ersatz*] [*Kommandozeile*]

Mit dem Shellkommando **fc** können einzelne Kommandozeilen aus dem Kommandozeilenspeicher, aber auch ganze Bereiche des Kommandozeilenspeichers editiert und danach ausgeführt werden. Als Editor wird der mit der ‘**-e**’ Option spezifizierte Editor benutzt, oder der in der Shellvariablen FCEDIT bestimmte, oder schließlich der Standardeditor vi, wenn kein anderer Editor bestimmt wird.

In der ersten Form werden die Kommandozeilen von *Anfang* bis *Ende* in den Editor geladen. Anfang und Ende können als Zeichenkette (in Übereinstimmung mit dem Anfang der gewünschten Kommandozeile) oder als Zahl (die absolute Position des Kommandos im Kommandozeilenspeicher) angegeben werden. Eine negative Zahl bestimmt Anfang und Ende relativ zum aktuellen Kommando.

Wenn kein Ende gesetzt ist, wird nur das Kommando am Anfang editiert. Wenn kein Anfang gesetzt ist, wird das letzte Kommando genommen.

Wenn die Option ‘**-|**’ gesetzt ist, wird der entsprechende Bereich von Kommandozeilen angezeigt anstatt ihn zu editieren. Wenn zusätzlich noch die Option ‘**-n**’ gesetzt ist, wird die Ausgabe der Zeilennummern vor den Kommandozeilen unterdrückt.

Wenn die Option ‘**-r**’ gesetzt ist, werden die Kommandozeilen in umgekehrte Reihenfolge in den Editor geladen.

Wenn das Shellkommando **fc** in der zweiten Form aufgerufen wird, ersetzt es das *Muster* in der *Kommandozeile* durch *Ersatz*.

fg

Syntax: **fg** [*Jobspezifikation*]

Das Shellkommando **fg** bringt einen (mit ^Z) angehaltenen Prozeß im Vordergrund zum laufen. Wenn keine Jobspezifikation (→ Seite 24) angegeben ist, wird der zuletzt angehaltene Job im Vordergrund gestartet.

hash

Syntax: **hash** [-r] [*Name*]

Die Shell unterhält eine Hashtabelle, in der alle seit dem Start der Shell aufgerufenen (externen) Kommandos mit komplettem Pfadnamen gespeichert werden. Das beschleunigt jeden weiteren Aufruf eines solchen Kommandos, weil nicht erst auf dem Pfad danach gesucht werden muß. Wenn das Shellkommando **hash** mit einem Kommandonamen aufgerufen wird, fügt es diesen Namen (mit Pfad) in die Hashtabelle ein.

Wenn die Option ‘-r’ gesetzt ist, wird die Hashtabelle gelöscht. Diese Option kann notwendig sein, wenn eine Binärdatei gelöscht oder verschoben worden ist. Wenn kein Argument angegeben ist, wird der Inhalt der Hashtabelle ausgegeben.

help

Syntax: **help** [*Shellkommando*]

Das **help** Shellkommando zeigt einen kurzen Hilfstext zu dem angegebenen Shellkommando, oder, wenn kein Kommando angegeben ist, eine Übersicht über alle Shellkommandos zu denen Hilftexte verfügbar sind.

history

Syntax: **history** [*n*]

history [-anrw] [*Datei*]

Wenn die **history** Shellfunktion in der ersten Form aufgerufen wird, zeigt sie die Einträge im Kommandozeilenspeicher an. Das Argument *n* schränkt die Ausgabe auf die letzten *n* Zeilen ein.

In der zweiten Form wird der Kommandozeilenspeicher mit der Option ‘-r’ aus der *Datei* gelesen, oder mit der Option ‘-w’ dorthin geschrieben. Mit der Option ‘-a’ werden die gespeicherten Zeilen an den Kommandozeilenspeicher angehängt und mit der Option ‘-n’ werden die noch nicht gelesenen Zeilen aus der Datei gelesen. Wenn kein Dateiname angegeben ist, wird der Dateiname aus der Shellvariablen HISTFILE genommen, die mit ~/.bash_history vorbelegt ist.

jobs

Syntax: **jobs** [-l_{np}] [*Jobspezifikation*]

jobs -x *Kommando* [*Argument* ...]

Das Shellkommando **jobs** gibt eine Liste der aktuellen Jobs aus. In dieser Liste steht neben der Jobnummer zu jedem Job der Kommandoname, der Status und eine Markierung ‘+’ für den ‘aktuellen Job’ und ‘-’ für den davor aktuellen Job (→ Seite 24).

Wenn die Option ‘-l’ gesetzt ist, wird zusätzlich die Prozeßnummer zu jedem Job ausgegeben. Mit der Option ‘-p’ wird nur die Prozeßnummer ausgegeben. Mit der Option ‘-n’ werden nur die Jobs angezeigt, die ihren Status seit der letzten Anzeige geändert haben. Wenn eine Jobspezifikation angegeben ist, werden nur die Daten zu diesem Job angezeigt.

Mit der Option ‘-x’ kann das Kommando ausgeführt werden, indem alle in den Argumenten auftauchenden Jobspezifikationen durch ihre Prozeßnummern ersetzt werden.

kill

Syntax: **kill** -*Signal* [*Prozeßnummer* | *Jobspezifikation*]

kill -l

Das Shellkommando **kill** sendet das *Signal* an den Prozeß *Prozeßnummer*. Standardwert ist SIGTERM (15) zum terminieren des Prozesses. Es können aber auch beliebige andere Signale gesendet werden. Das Signal kann als Name oder als Nummer angegeben werden. Die Jobspezifikation ist auf Seite 24 erklärt.

Mit der Option ‘-l’ werden alle möglichen Signalnamen aufgelistet.

Es gibt auch ein externes **kill** Kommando, mit dem die gleichen Signale gesendet werden können, das aber nicht mit einer Jobspezifikation in der Kommandozeile umgehen kann.

let

Syntax: **let** *Ausdruck* [*Ausdruck* ...]

Das Shellkommando **let** berechnet jedes Argument als arithmetischen Ausdruck. Der Rückgabewert von **let** ist 1, wenn der letzte Ausdruck null liefert; sonst ist der Status null.

Die Syntax der Ausdrücke ist auf Seite 31 erklärt.

local

Syntax: **local** [*Name*[= *Wert*]]

Das Shellkommando **local** erzeugt eine lokale Variable *Name*, und weist ihr den *Wert* zu. Wenn eine lokale Variable innerhalb einer Funktion erzeugt wird, so ist sie nur innerhalb dieser Funktion und allen Unterfunktionen zugänglich. Außerhalb von Funktionen hat die Shellfunktion **local** keine Bedeutung.

Wenn kein Name angegeben ist, werden alle lokalen Variablen angezeigt.

logout

Syntax: **logout**

Das Shellkommando **logout** beendet eine Loginshell. Dabei wird das Shellsript '~/.bash_logout' abgearbeitet, wenn die Shell als **bash** gestartet wurde.

popd

Syntax: **popd** [+|-*n*]

Das **popd** Shellkommando löscht einen Verzeichnisnamen vom Verzeichnisstapel. Ohne Argument wird das erste (oberste) Verzeichnis vom Stapel geholt, und mit **cd** ein Verzeichniswechsel dorthin ausgeführt.

Mit der Option '+|-*n*' kann ein bestimmtes Verzeichnis aus dem Stapel gelöscht werden. Dabei ist die Zahl *n* die Position im Stapel, beginnend mit Null, und das Vorzeichen gibt an, ob das Verzeichnis vom "Anfang" (links in der Liste von **dirs**, mit '+') oder vom "Ende" ('-') aus gezählt werden soll.

Wenn die Shellvariable **pushd_silent** gesetzt ist, wird die Auflistung des aktualisierten Verzeichnisstapels (mit **dirs**) unterdrückt.

pushd

Syntax: **pushd** *Verzeichnis*

pushd[+|-*n*]

Das Shellkommando **pushd** legt das *Verzeichnis* als oberstes auf dem Verzeichnisstapel ab, oder rotiert den Verzeichnisstapel um die angegebenen Positionen.

Die Bedeutung der Shellvariablen **pushd_silent** ist die gleiche wie beim **popd** Shellkommando.

pwd

Syntax: **pwd**

Das Shellkommando **pwd** gibt den Pfadnamen des aktuellen Verzeichnisses aus.

read

Syntax: **read** [-*r*] [*Name* ...]

Die Shellfunktion **read** liest eine Zeile von der Standardeingabe, und weist die (durch die IFS getrennten) Worte den benannten Shellvariablen zu. Wenn mehr Worte in der Zeile sind als Namen angegeben, werden die verbleibenden Worte alle in der letzten benannten Variablen gespeichert.

Wenn die Option '-*r*' gesetzt ist, wird ein durch ein '\' eingeleitetes Zeilenende als Teil der Eingabe in einer Variablen abgespeichert.

Wenn kein Name für die Variable angegeben ist, unter dem die Eingabe gespeichert werden soll, so wird automatisch die Shellvariable **REPLY** (Antwort) benutzt.

readonly

Syntax: **readonly** [**-pf**] [*Name*]

Die Shellfunktion **radonly** gibt Variablen (oder Scriptfunktionen mit der Option ‘-f’) den “nur-lesen” Status. Solche Variable können nicht gelöscht oder verändert werden.

Wenn kein Name angegeben ist oder die Option ‘-p’ gesetzt ist, werden alle Variablen mit “nur-lesen” Status angezeigt.

return

Syntax: **return** [*n*]

Die **return** Shellfunktion hat nur innerhalb einer Scriptfunktion eine Bedeutung, und veranlaßt dort die Funktion mit dem angegebenen Rückgabewert zu verlassen.

Wenn kein Rückgabewert angegeben ist, wird der Status des zuletzt ausgeführten Kommandos weitergereicht.

set

Syntax: **set** [**-abefhknotuvxldHC**] [*Argument*]

- a** markiert alle neu erzeugten oder veränderten Variablen automatisch für den Export
- b** zeigt die Beendigung eines Jobs sofort an, ohne auf die nächste Eingabeaufforderung zu warten
- e** beendet die Shell sofort, wenn ein Kommando nicht den Rückgabewert Null liefert
- f** unterdrückt die Pfadnamenerweiterung
- h** speichert Funktionen sofort bei ihrer Definition in der Umgebung
- k** schreibt die komplette Kommandozeile in die Umgebung einer Funktion, nicht bloß die Argumente nach dem Funktionsnamen
- m** ermöglicht die Benutzung der Job-Kontrollfunktionen
- n** liest Kommandos ohne sie auszuführen. Diese Option funktioniert nicht in interaktiven Shells und dient zum Testen von Shellscripts
- o** *Option* setzt die *Option*. Dabei sind folgende Optionen erlaubt:
 - allexport** das Gleiche wie die Option **-a**
 - braceexpand** in geschweiften Klammern eingeschlossene, durch Komma getrennte Listen von Wortteilen in der Kommandozeile werden durch mehrere Worte mit je einem eingefügten Wortteil ersetzt (wie in der C-Shell)
 - emacs** schaltet den Kommandozeileneditor in den **emacs**-Stil
 - errexit** das Gleiche wie die Option **-e**
 - histexpand** das Gleiche wie die Option **-H**
 - ignoreeof** unterdrückt das Verlassen der Shell beim lesen von EOF
 - monitor** das Gleiche wie die Option **-m**
 - noclobber** verbietet das Überschreiben existierender Dateien durch Ausgabeumleitung, wie **-C**
 - noexec** das Gleiche wie die Option **-n**
 - noglob** das Gleiche wie die Option **-f**
 - nohash** das Gleiche wie die Option **-d**
 - notify** das Gleiche wie die Option **-b**

nounset das Gleiche wie die Option `-u`
verbose das Gleiche wie die Option `-v`
vi schaltet den Kommandozeileneditor in den vi-Stil
xtrace das Gleiche wie die Option `-x`

- `-t` beendet die Shell sofort nach der Ausführung eines einzigen Kommandos
- `-u` erzeugt eine Fehlermeldung für jede leere (ungesetzte) Variable, die erweitert werden soll
- `-v` gibt jede Kommandozeile aus, so wie sie gelesen wurde
- `-x` gibt nach der Erweiterung jedes einfachen Kommandos den Inhalt der Shellvariablen `PS4` gefolgt von dem erweiterten Kommando mit allen Argumenten aus
- `-l` speichert und restauriert die Belegung der Zählvariablen vor und nach einer `for` Schleife
- `-d` unterdrückt die Benutzung einer Hashtabelle für die Pfadnamen der Kommandos
- `-H` ermöglicht den Bezug auf Zeilen im Kommandozeilenspeicher mit dem `!` wie in der `csch`
- `-C` verbietet das Überschreiben existierender Dateien durch Ausgabeumlenkung (wie `noclobber`)
- `--` setzt die Positionsparameter auf die der Option folgenden Werte, auch wenn einer der Werte mit einem `'-'` beginnt. Wenn keine Werte folgen, werden die Positionsparameter gelöscht
- `-` setzt die Positionsparameter auf die dem Option folgenden Werte. Wenn keine Werte folgen, bleiben die Positionsparameter unverändert

Wenn anstelle des `'-'` bei den Optionen ein `'+'` gesetzt ist, wird die entsprechende Option abgeschaltet. Alle hier aufgeführten Optionen können auch beim Aufruf der Shell in der Kommandozeile gesetzt werden. Die aktuell gesetzten Optionen können mit der Shellvariablen `'-'` angezeigt werden (`echo $-`)

Wenn keine Optionen angegeben sind, werden alle Shellvariablen mit ihren Werten angezeigt.

shift

Syntax: **shift** [*n*]

Die `shift` Shellfunktion verschiebt (shiftet) die Positionsparameter um *n* Stellen nach links. Die Herausgeschobenen Parameter sind verloren. Wenn keine Anzahl angegeben ist, wird um eine Stelle geshiftet.

source

Syntax: **source** *Datei*

Die `source` Shellfunktion läßt die Shell das Shellsript *Datei* abarbeiten. Diese Funktion kann auch durch einen einzelnen Punkt am Anfang der Kommandozeile ausgelöst werden.

suspend

Syntax: **suspend** [`-f`]

Die `suspend` Shellfunktion veranlaßt die Shell auf das Signal `SIGCONT` zu warten. Wenn die Option `'-f'` gesetzt ist, kann auch eine Loginshell mit dieser Funktion angehalten werden.

test

Syntax: **test** *Ausdruck*

[Ausdruck]

Die Shellfunktion **test** bewertet den Ausdruck und liefert Null wenn der Ausdruck wahr (!) ist, und Eins wenn er falsch ist. Dieser Unterschied zu der gängigen Definition der Wahrheitswerte in C ist für die Shellprogrammierung normal! Ein Ausdruck kann eine unäre oder eine binäre Operation enthalten. Unäre Operationen dienen oft zum Ermitteln des Zustandes einer Datei.

Folgende Operationen können (wie Optionen) angegeben werden:

- b** *Datei* ist wahr, wenn die *Datei* ein Blockdevice ist
- c** *Datei* ist wahr, wenn die *Datei* ein Zeichendevise ist
- d** *Datei* ist wahr, wenn die *Datei* ein Verzeichnis ist
- e** *Datei* ist wahr, wenn die *Datei* existiert
- f** *Datei* ist wahr, wenn die *Datei* eine einfache Datei ist
- g** *Datei* ist wahr, wenn bei die *Datei* das SGID Bit gesetzt ist
- k** *Datei* ist wahr, wenn bei der *Datei* das “sticky” Bit gesetzt ist
- L** *Datei* ist wahr, wenn die *Datei* ein symbolischer Link ist
- p** *Datei* ist wahr, wenn die *Datei* eine benannte Pipeline (Named Pipe) ist
- r** *Datei* ist wahr, wenn die *Datei* existiert und lesbar ist
- s** *Datei* ist wahr, wenn die *Datei* existiert und größer als Null Bytes ist
- S** *Datei* ist wahr, wenn die *Datei* ein “Socket” ist
- t** *Dateinummer* ist wahr, wenn die Datei mit der *Dateinummer* für ein Terminal geöffnet ist. Wenn keine Nummer angegeben ist, wird Nummer 1 (Standardausgabe) angenommen
- u** *Datei* ist wahr, wenn die *Datei* existiert und das SUID Bit gesetzt hat
- w** *Datei* ist wahr, wenn die *Datei* existiert und beschreibbar ist
- x** *Datei* ist wahr, wenn die *Datei* existiert und ausführbar ist
- O** *Datei* ist wahr, wenn die *Datei* existiert und im Eigentum des Anwenders ist unter dessen UID das **test** Shellkommando läuft
- G** *Datei* ist wahr, wenn die *Datei* existiert und im Eigentum des Benutzers ist unter dessen GID das **test** Shellkommando läuft
- Datei1* –nt *Datei2*** ist wahr, wenn die *Datei1* neuer ist als die *Datei2*
- Datei1* –ot *Datei2*** ist wahr, wenn die *Datei1* älter ist als die *Datei2*
- Datei1* –ef *Datei2*** ist wahr, wenn *Datei1* und *Datei2* die gleiche Inode auf dem gleichen Device belegen
- z** *Zeichenkette* ist wahr, wenn die Länge der *Zeichenkette* Null ist
- n** *Zeichenkette* ist wahr, wenn die Länge der *Zeichenkette* nicht Null ist
- Zeichenkette* ist auch wahr, wenn die Länge der *Zeichenkette* nicht Null ist
- Zeichenkette1* = *Zeichenkette2*** ist wahr, wenn die Zeichenketten gleich sind

Zeichenkette1 != Zeichenkette2 ist wahr, wenn die Zeichenketten nicht gleich sind

! Ausdruck ist wahr, wenn der Ausdruck falsch ist

Ausdruck1 -a Ausdruck2 ist wahr, wenn *Ausdruck1* UND *Ausdruck2* wahr sind

Ausdruck1 -o Ausdruck2 ist wahr, wenn *Ausdruck1* ODER *Ausdruck2* wahr ist

Argument1 OP Argument2 *OP* steht hier für einen der arithmetischen Vergleich *-eq*, *-ne*, *-lt*, *-le*, *-gt* und *-ge* (gleich, ungleich, kleiner, kleinergleich, größer, größergleich) Der Ausdruck ist wahr, wenn die Relation von *Ausdruck1* und *Ausdruck2* stimmt.

times

Syntax: **times**

Das Shellkommando **times** gibt die verbrauchte Benutzer- und Systemzeit jeweils für die Shell und für die von der Shell aus gestarteten Prozesse aus

trap

Syntax: **trap** [*Kommando*] [*Signal*]

trap -|

Die Shellfunktion **trap** fängt das angegebene *Signal* ab, und führt das *Kommando* aus. Wenn kein *Signal* benannt ist, werden alle Signale zurückgesetzt. Wenn als *Kommando* der Nullstring angegeben ist, wird das damit angegebene *Signal* von der Shell und von allen *Kommandos*, die von dieser Shell ausgeführt werden, ignoriert.

Das *Signal* kann entweder als *Zahl* oder mit seinem Namen angegeben werden. Eine Liste aller möglichen Signale kann vom Shellkommando **trap** mit der Option *'-|'* ausgegeben werden.

Wenn das *Signal* **EXIT** (0) angegeben ist, wird das *Kommando* als letztes vor der Beendigung der Shell ausgeführt.

Wenn keine Argumente angegeben sind, wird eine Liste aller *Kommandos* und der damit verbundenen Signale ausgegeben.

type

Syntax: **type** [*-all*] [*-type* | *-path*] [*Name*]

Die Shellfunktion **type** gibt an, wie der angegebene *Name* von der Shell interpretiert würde, wenn er als ein *Kommando* auftauchen würde (alias, Scriptfunktion, Shellfunktion (builtin), Datei). Wenn der *Name* nicht gefunden wird, gibt **type** nichts aus.

Mit der Option *-type* wird nur ein Wort für den Kommandotyp entsprechend den oben genannten Möglichkeiten ausgegeben.

Wenn die Option *-path* benutzt wird, gibt die Shellfunktion den kompletten Pfadnamen des benannten *Kommandos* aus. Wenn es kein externes *Kommando* mit dem Namen gibt, wird nichts ausgegeben.

Die Option *-all* veranlaßt die Shellfunktion, nicht nur die erste Fundstelle eines passenden *Kommandos* anzuzeigen, sondern alle möglichen. Mit der Option *-path* kann dabei zusätzlich die Ausgabe auf externe *Kommandos* eingeschränkt werden.

typeset

Siehe **declare**

ulimit

Syntax: **ulimit** [**-SHacdfmnpst** [*Limit*]]

Die Shellfunktion **ulimit** erlaubt die Kontrolle über die von der Shell und den daraus gestarteten Programmen benutzten Systemressourcen.

- S** setzt "weiche" Grenzen ??
- H** setzt "harte" Grenzen ??
- a** zeigt alle eingestellten Grenzwerte an
- c** schränkt die Größe des Speicherabzugs (core) bei einem Programmabsturz ein
- d** schränkt die maximale Größe des Datensegments für Prozesse ein
- f** erlaubt dem Anwender nur Dateien bis zu einer bestimmten Größe zu erzeugen
- m** schränkt die Größe des residenten (nicht auszulagernden) Teiles der Prozesse ein
- n** beschränkt die maximale Anzahl offener Dateien
- p** bestimmt die Größe des Pipeline-Puffers (in 512 Byte Blöcken)
- s** schränkt den Stapelspeicher (Stack) auf eine bestimmte Größe ein
- t** beschränkt die verfügbare CPU Zeit auf eine bestimmte Zeit (in Sekunden)

Die Grenzen werden in Kilobytes angegeben, wenn oben keine andere Einheit genannt ist. Wenn beim Aufruf keine Grenze bestimmt wird, gibt **ulimit** die aktuelle Grenze aus.

umask

Syntax: **umask** [**-S**] [*Modus*]

Die Shellfunktion **umask** setzt die Maske für die Zugriffsrechte einer Datei bei ihrer Erzeugung. Wenn der Modus mit einer Ziffer beginnt wird er als Oktalzahl interpretiert. Sonst wird eine Angabe in der beim Kommando **chmod** auf Seite 45 angegebenen Form erwartet, auf den (oktalen) Wert *nnn*. Wenn kein Wert angegeben ist, wird die aktuelle Maske angezeigt.

Die in der Maske gesetzten Bits werden bei den Zugriffsrechten auf die Datei (oder das Verzeichnis) **nicht** gesetzt (sie werden maskiert).

Die Maske '022' verbietet beispielsweise generell allen Benutzern außer dem Eigentümer selbst das Schreiben in eine neuangelegte Datei oder ein Verzeichnis.

Wenn die Option '**-S**' gesetzt ist, wird die aktuelle Maske in Symbolischer Form ausgegeben.

unalias

Syntax: **unalias** [**-a**] [*Name* ...]

Die Shellfunktion **unalias** hebt ein durch das **alias** Shellkommando gesetztes Synonym für ein Kommando wieder auf.

Mit der Option '**-a**' werden alle Synonyme gelöscht.

unset

Syntax: **unset** [-fv] [*Name* ...]

Mit der Shellfunktion **unset** werden Shellvariable aus der Umgebung entfernt. Mit der Option ‘-f’ wird die bezeichnete Funktion aus der Umgebung entfernt, mit der Option ‘-v’ die entsprechende Variable. Wenn keine der Optionen angegeben ist, wird zuerst versucht, eine Variable mit passendem Namen zu entfernen, und nur wenn dieser Versuch fehlschlägt wird die entsprechende Funktion aus der Shellumgebung entfernt.

Eine Variable bzw. eine Funktion kann nur mit dem **unset** Kommando aus der Umgebung entfernt werden. Wenn eine Variable mit der leeren Zeichenkette (“”) belegt ist, gilt sie weiterhin als gesetzt.

Die Shellvariablen PATH, IFS, PPID, PS1, PS2, UID und EUID können nicht aus der Umgebung entfernt werden.

wait

Syntax: **wait** [*Jobspezifikation* | *Prozeßnummer*]

Die Shellfunktion **wait** wartet auf die Beendigung des durch die *Jobspezifikation* (Jobnummer oder Kommandoname) oder die Prozeßnummer angegebenen Hintergrundprozesses und gibt dessen Status aus.

Wenn kein Job spezifiziert wurde, wartet die Shellfunktion auf alle aktiven Hintergrundprozesse

1.4.22 Login- und andere Shells

Wenn die Bash mit einem ‘-’ als erstes Zeichen des nullten Arguments aufgerufen wird (wie es das **login** Kommando macht), oder wenn die Shell mit der **-login** Option aufgerufen wird, arbeitet sie als Loginshell.

Bei einer interaktiven Shell ist die Standardeingabe und die Standardausgabe mit einem Terminal (bzw. der Konsole) verbunden. Wenn die Shell mit der Option ‘-i’ gestartet wird, arbeitet sie auch interaktiv.

Eine Loginshell arbeitet vor der ersten Kommandozeile eine Reihe von Shellscripts zur Initialisierung ab:

Wenn ein Shellscrip /etc/profile existiert (und lesbar ist), werden die darin beschriebenen Einstellungen und Kommandos ausgeführt.

Wenn im Heimatverzeichnis des Benutzers die Datei ~/.bash_profile existiert, werden anschließend die darin enthaltenen Einstellungen und Kommandos zusätzlich ausgeführt. Wenn diese Datei nicht existiert, wird nacheinander noch nach den Dateien ~/.bash_login und ~/.profile gesucht, und ebenso behandelt.

Beim Verlassen der Loginshell (logout) wird die Datei ~/.bash_logout abgearbeitet.

Eine interaktive Shell, die keine Loginshell ist, arbeitet die Datei ~/.bashrc zur Initialisierung ab.

Eine nicht-interaktive Shell schließlich benutzt zur Initialisierung die in der Shellvariablen ENV angegebene Datei.

1.4.23 Dateien

/bin/bash das ausführbare Programm

/etc/profile die Standardinitialisierungsdatei für alle Loginshells

~/.bashrc die Initialisierungsdatei für Bash

~/.bash_profile Initialisierungsdatei für Bash als Loginshell

~/.bash_logout Shellscrip zum Aufräumen vor dem Logout

~/.inputrc Tastaturkommandobelegungen für den Kommandozeileneditor

1.4.24 Autoren

Brian Fox, Free Software Foundation und Chet Ramey, Case Western Reserve University

1.5 cat

Funktion:

cat (*concatenate*) verkettet Dateien und schreibt sie in die Standardausgabe

Syntax:

```
cat [-benstuvAET] [--number] [--number-nonblank] [--squeeze-blank] [--show-nonprinting]
[--show-ends] [--show-tabs] [--show-all] [Datei ...]
```

Beschreibung:

cat liest beliebige Dateien und schreibt sie ohne Veränderung in die Standardausgabe. Durch Umlenkung der Ausgabe auf eine Datei können so Dateien verkettet werden. Außerdem wird **cat** häufig benutzt, um Dateien an Programme zu übergeben, die nur von der Standardeingabe lesen. (Solche Programme werden im allgemeinen als Filter bezeichnet.) Für die Übergabe steht entweder die Ausgabeumlenkung der Shell mit '>' und '>>' zur Verfügung, oder die Ausgabe wird durch eine Pipeline '|' an den Filter weitergeleitet.

Optionen:

- b** alle nichtleeren Zeilen erhalten eine Zeilennummer
- e** das gleiche wie -vE
- n** sämtliche Zeilen werden nummeriert
- s** mehrere leere Zeilen in Folge werden zu einer einzigen leeren Zeile zusammengefaßt
- t** das gleiche wie -vT
- u** ohne Funktion
- v** alle Kontrollzeichen außer TAB und NEWLINE werden angezeigt
- A** das gleiche wie -vET
- E** gibt ein '\$' Zeichen am Ende jeder Zeile aus
- T** die TAB werden als ^I angezeigt

Siehe auch:

tac auf Seite 142

Autor:

Torbjorn Granlund und Richard Stallman

cd

cd ist ein eingebautes Shellkommando. Siehe im Abschnitt über die **bash** auf Seite 33.

1.6 chgrp

Funktion:

chgrp (*change group*) ändert die Gruppenzugehörigkeit einer Datei oder eines Verzeichnisses

Syntax:

chgrp [-Rcfv] [--recursive] [--show-changes] [--silent] [--quiet] [--verbose] *Gruppe Datei ...*

Beschreibung:

Der Befehl **chgrp** ändert die Gruppenzugehörigkeit einer Datei oder eines Verzeichnisses. Die Benutzung von **chgrp** ist nur dem Eigentümer und dem Superuser (root) erlaubt. Der Eigentümer kann eine Datei nur den Gruppen zuordnen, denen er selbst auch angehört.

Die *Gruppe* kann als ganze Zahl oder als Name angegeben werden. Die möglichen Gruppen und ihre Kennzahlen sind in der Datei */etc/group* festgelegt, und können mit *id* angezeigt werden.

Optionen:

-c (changes) diese Option zeigt die Dateien an, deren Gruppe geändert wird

-f (force) es werden keine Fehlermeldungen ausgegeben

-v (verbose) alle Aktionen werden angezeigt

-R (recursive) die Gruppenzugehörigkeit der Dateien in den Unterverzeichnissen werden ebenfalls geändert

Siehe auch:

chmod auf Seite 45 und **chown** auf Seite 151

Autor:

David MacKenzie

1.7 chmod

Funktion:

chmod (change mode) ändert die Zugriffsrechte auf Dateien und Verzeichnisse

Syntax:

chmod [-Rcfv] *Modus Datei ...*

Beschreibung:

chmod setzt oder ändert die Zugriffsrechte auf Dateien oder Verzeichnisse. Die Benutzung von **chmod** ist nur dem Eigentümer oder dem Superuser (root) erlaubt.

Die Zugriffsrechte werden als Modus bezeichnet. Der *Modus* kann entweder als (drei- oder vierstellige) Oktalzahl oder durch Buchstabenkennungen angegeben werden. Bei Angabe als Oktalzahl legen die letzten drei Ziffern jeweils die Rechte für den Besitzer, die Gruppe und die Anderen fest. Die einzelnen Bits der Oktalziffer stehen dabei für Lesen (4), Schreiben (2) und Ausführen (1).

Wenn eine vierte Ziffer angegeben wird, so setzt die erste Ziffer spezielle Ausführungsmodi:

Wenn das erste Bit (4) dieser Zahl gesetzt ist, wird ein Programm mit der effektiven Benutzerkennung (EUID für Effective User-ID) des Besitzers dieser Datei ausgeführt.

Wenn das zweite Bit (2) dieser Zahl gesetzt ist, wird ein Programm mit der Gruppenkennung dieser Datei anstelle der realen Gruppenkennung des aufrufenden Benutzers ausgeführt.

Wenn schließlich das dritte “sticky” Bit (1) dieser Zahl gesetzt ist, wird das Programm permanent im Speicher gehalten.

Die Buchstabenkennung setzt sich aus den folgenden Teilen zusammen:

[ugoa ...][+-=][rwxstugo ...][, ...]

Dabei steht **u** (user) für Besitzer, **g** (group) für Gruppe, **o** (other) für Andere und **a** (all) für Alle. Die Arithmetischen Symbole **+-=** geben an ob eine Berechtigung hinzugefügt (+), gelöscht (−) oder gesetzt (=) werden soll. Die Berechtigungen sind **r** (read) für Lesen, **w** (write) für Schreiben, **x** (execute) für Ausführen. Die Option **s** (set user/group ID on execution) setzt die Identitätskennung bei der Programmausführung. Die Option **t** (text) schützt den Programmtext vor Überschreiben im Swapbereich. Die nachgestellten **u g** und **o** schützen die entsprechenden Rechte für Besitzer, Gruppe und andere vor Veränderung (zur Benutzung im Zusammenhang mit **-a**).

Die Rechte von symbolischen Links werden von **chmod** nicht geändert. Es gelten hier immer die Rechte der Datei auf die der Link zeigt.

Beispiele:

Die oktale Zahl ‘644’ entspricht der binären Zahl ‘110.100.100’ ($6 = 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$; $4 = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$). Für den Eigentümer sind die Bits zum Lesen und zum Schreiben gesetzt, für die Gruppe und die anderen nur das Bit zum Lesen. Die Buchstabenkennung ‘**u=w,a=r**’ setzt die gleichen Zugriffsrechte.

Die oktale Zahl ‘4750’ entspricht der Bitkombination ‘100.111.101.000’. Hier erhält der Eigentümer die Rechte zum Lesen, Schreiben und Ausführen, die Gruppe erhält die Rechte zum Lesen und zum Ausführen und die anderen haben keinerlei Zugriffsrecht auf die Datei. Außerdem ist hier das SUID Bit gesetzt; das bedeutet, daß alle Gruppenmitglieder diese Datei mit der effektiven Benutzerkennung des Eigentümers ausführen können. Die Buchstabenkennung ‘**u=rws,g=wx,o=**’ setzt die gleichen Zugriffsrechte.

Optionen:

- c** (changes) es werden nur die Dateien angezeigt, deren Zugriffsrechte tatsächlich verändert werden
- f** (force) Fehlermeldungen wegen fehlgeschlagener Änderungsversuche werden unterdrückt
- v** (verbose) alle Aktionen werden angezeigt
- R** (recursive) die Zugriffsrechte aller Dateien in den Unterverzeichnissen werden ebenfalls geändert

Siehe auch:

chgrp auf Seite 44 und **chown** auf Seite 151

Autor:

David MacKenzie

1.8 chsh

Funktion:

chsh ändert den Loginshell Eintrag in der Paßwortdatei.

Syntax:

chsh [*Benutzer*] [*Shell*]

Beschreibung:

chsh ermöglicht es jedem eingetragenen Benutzer, seine Loginshell selbst, das heißt ohne Hilfe des Superusers (root) zu verändern. Die Loginshell wird in der Paßwortdatei `/etc/passwd` festgelegt. Diese Datei kann nur mit Rootprivilegien verändert werden. Um auch den anderen Anwendern das Verändern des Eintrages zu erlauben, läuft das **chsh** Programm SUID root. Das heißt, bei seiner Ausführung wird die effektive Benutzerkennung des Superusers gesetzt. Um die Systemsicherheit trotzdem zu gewährleisten, können nur Programme, die in der Datei `/etc/shells` eingetragen sind als Loginshell benutzt werden.

Normalerweise kann ein Anwender nur seine eigene Loginshell ändern. Der Superuser selbst kann das Programm aber auch für andere Benutzer anwenden, indem er den Benutzernamen in der Kommandozeile angibt.

Autor:

Peter Orbaek

1.9 cksum

Funktion:

cksum errechnet die CRC Prüfsumme und die Anzahl Bytes für eine Datei

Syntax:

cksum [*Datei* ...]

Beschreibung:

cksum errechnet die Prüfsumme für eine oder mehrere Dateien nach dem im POSIX.2 Standard festgelegten “cyclic redundancy check” (CRC) Verfahren. Wenn keine Datei oder anstelle einer Datei ‘-’ angegeben ist, liest **cksum** die Standardeingabe und gibt die Prüfsumme aus nachdem die Eingabe mit CONTROL-D beendet wurde.

Die Ausgabe von **cksum** besteht aus der Prüfsumme, der Dateilänge und dem Dateinamen. In dieser Form können diese Prüfsummen beispielsweise beim Verschicken übers Internet den Dateien mitgegeben werden, um dem Empfänger die Möglichkeit zu geben, den vollständigen und korrekten Empfang der Dateien durch den Vergleich der Prüfsummen zu bestätigen.

Es gibt noch andere Verfahren zur Prüfsummenberechnung, von denen zwei durch das **sum** Kommando angeboten werden. Das **cksum** Programm ist nicht kompatibel zu **sum**.

Autor:

Q. Frank Xia

Siehe auch:

sum auf Seite 141

1.10 cmp

Funktion:

cmp (compare) vergleicht zwei Dateien Byteweise

Syntax:

cmp [-cls] [--show-chars] [--verbose] [--silent] [--quiet] *Datei1* [*Datei2*]

Beschreibung:

cmp vergleicht zwei (binäre) Dateien und liefert die dezimale Position des ersten Bytes in dem sich die Dateien unterscheiden sowie die Zeilennummer zurück. Wird statt eines Dateinamens ‘—’ angegeben, so wird von der Standardeingabe gelesen; wird nur eine Datei benannt, so wird anstelle der Zweiten ebenfalls von der Standardeingabe gelesen.

Optionen:

-c (character) gibt die abweichenden Zeichen aus

-l (list) gibt die Position und den oktalen Wert aller differierenden Zeichen in einer Liste aus.

-s (silent) gibt nichts auf die Standardausgabe. Der Status ist 0 (wahr) wenn die Dateien übereinstimmen und 1 (falsch) wenn sie sich unterscheiden

Siehe auch:

diff(info) und comm auf Seite 48

Autor:

Torbjorn Granlund und David MacKenzie

1.11 comm

Funktion:

comm vergleicht zwei sortierte Dateien

Syntax:

comm [-{1,2,3}] *Datei1* *Datei2*

Beschreibung:

comm vergleicht zwei sortierte Dateien, und gibt die gemeinsamen und die verschiedenen Zeilen jeweils in Spalten aus. Die erste Spalte enthält die Zeilen, die nur in Datei1 enthalten sind. Die zweite Spalte enthält die Zeilen, die in beiden Dateien enthalten sind. Die dritte Spalte enthält schließlich die Zeilen, die nur in der zweiten Datei enthalten sind.

Ein ‘—’ anstelle eines Dateinamens steht für die Standardeingabe.

Optionen:

-{1,2,3} unterdrückt die erste, zweite bzw. dritte Spalte

Siehe auch:

diff(info) und cmp auf Seite 47

Autor:

Richard Stallman und David MacKenzie

1.12 compress

Funktion:

compress komprimiert Dateien

Syntax:

compress [-cdfvV] [-b *Maxbits*] [*Datei* ...]

Beschreibung:

compress komprimiert Dateien mit LZW (einem veränderten Lempel-Ziv) Algorithmus. **compress** überprüft erst, ob die *Datei* nach der Kompression wirklich kleiner ist als vorher. Nur in diesem Fall wird die *Datei* durch die komprimierte *Datei.Z* ersetzt. Wenn keine Datei angegeben wird, liest **compress** von der Standardeingabe und schreibt in die Standardausgabe. Wenn die Ausgabedatei *Datei.Z* schon existiert, wird sie nicht überschrieben. **compress** erhält den Zeitstempel der Datei.

Die Programme **uncompress** und **zcat** sind Links auf **compress**, bei denen bestimmte Optionen vorgelegt sind.

uncompress arbeitet wie '**compress -d**', das heißt, es entkomprimiert die mit **compress** gepackten Dateien.

zcat arbeitet wie '**compress -dc**', das heißt, es schreibt die entkomprimierten Dateien auf die Standardausgabe. Diese Funktion wird in den Shellscripts **zdiff**, **zmore** oder **zless** benutzt, um den Inhalt komprimierter Dateien direkt an bestimmte Filter weiterzuleiten.

Das **compress** Programm und seine Links kann als Standardpacker für Unix bezeichnet werden. Es wird im Zusammenhang mit **tar** auch für gepackte Dateiarhive verwendet. Das GNU **tar** bietet eine direkte Unterstützung von **compress** (→ Seite 143).

Allerdings wird in letzter Zeit dazu übergegangen das neue **gzip** Programm zum Packen einzelner Dateien zu benutzen, weil es deutlich höhere Kompressionsraten aufweisen kann. Es ist damit zu rechnen, daß **compress** durch **gzip** verdrängt wird.

Optionen:

-c die (de-)komprimierte Datei wird in die Standardausgabe geschrieben

-d dekomprimiert die Datei; die komprimierte Datei wird ersetzt

-f überschreibt existierende Ausgabedateien und ersetzt *Datei*, selbst wenn sie durch die Kompression nicht kleiner wird

-v gibt den Dateinamen und das Größenverhältnis aus

-V gibt die Versionsnummer aus

-r komprimiert rekursiv alle Dateien in den Unterverzeichnissen

-b *Maxbits* setzt die Kompressionstiefe (Voreinstellung ist 16 Bit)

Autoren:

Spencer W. Thomas, Jim McKie, Steve Davies, Ken Turkowski, James A. Woods, Joe Orost, Dave Mack und Peter Jannesen

1.13 cp

Funktion:

cp (copy) kopiert eine oder mehrere Dateien

Syntax:

cp [*Optionen*] *Quelle Ziel*

cp [*Optionen*] *Quelle ... Verzeichnis*

Optionen:

- a** (archiv) das gleiche wie -dpR
- b** (backup) sichert Dateien im Zielverzeichnis vor dem Überschreiben
- d** (no-dereference) kopiert die Links und nicht die Dateien auf die der Link zeigt
- f** (force) Dateien im Zielverzeichnis werden überschrieben
- i** (interactive) erwartet Bestätigung vor dem Überschreiben bereits existierender Dateien
- l** (link) macht Links anstelle von Kopien (nur bei normalen Dateien)
- P** (path) die Quelldateien werden mit Pfad relativ zum Zielverzeichnis kopiert
- p** (preserve) erhält die Zutrittsrechte und Eigentümer des Originals (nicht die SUID und SGID Bits)
- r** kopiert die Dateien der Unterverzeichnisse mit
- s** (symbolic link) macht symbolische Links anstelle von Kopien (absolute Pfadnamen)
- u** (update) überschreibt Ziel- nur durch neuere Quelldateien
- v** (verbose)
- x** (one file-system) ignoriert Unterverzeichnisse die in anderen Dateisystemen angesiedelt sind
- R** (recursive)
- S** *Endung* (suffix) sichert die Dateien vor dem Überschreiben durch Umbenennung mit der *Endung*; Voreinstellung ist '~'
- V** {**numbered**, **existing**, **simple**} (version-control) erhält auch frühere Versionen einer Datei, indem jeweils neue Backups erzeugt werden. Die Art der Backups kann auch durch die Umgebungsvariable VERSION_CONTROL bestimmt werden. Die Option -V überschattet VERSION_CONTROL. Wenn weder die Option noch die Environmentvariable gesetzt sind, wird *existing* benutzt. Gültige Werte für VERSION_CONTROL sind:
 - numbered** Backups werden nummeriert
 - existing** Backups werden nur für Dateien nummeriert die bereits nummerierte Backups haben.
 - simple** es werden immer einfache Backups gemacht

Autor:

Torbjorn Granlund, David MacKenzie und Jim Meyering

1.14 csplit

Funktion:

csplit (context split) teilt eine Datei in mehrere Teile wobei die Trennstelle durch ein Suchmuster angegeben werden kann

Syntax:

csplit [-sk] [-f *Prefix*] [-n *Stellen*] [--prefix=*Prefix*] [--digits=*Stellen*] [--quiet] [--silent] [--keep-files] *Datei Muster* ...

Beschreibung:

csplit liest aus *Datei* oder aus der Standardeingabe wenn als Datei '-' angegeben wird und gibt den Inhalt in mehrere Dateien aus. Der Inhalt der Ausgabedateien ist abhängig vom *Muster*. Die erste Zeile, in der das *Muster* vorkommt, wird zur ersten Zeile der nächsten Datei. Als *Muster* kommen folgende in Frage:

/Ausdruck/[Offset] {Anzahl} erzeugt eine Ausgabedatei die alle Zeilen der Eingabe bis (ausschließlich) der Zeile mit dem *Ausdruck* enthält. Wird zusätzlich eine Zahl *{Anzahl}* in geschweiften Klammern angegeben, wird der Vorgang mit dem gleichen *Ausdruck* und dem verbleibenden Rest *Anzahl* mal wiederholt. Wird zusätzlich eine ganze Zahl mit einem führenden '+' oder '-' als *Offset* angegeben, so wird der Beginn der nächsten Datei um diese Anzahl Zeilen verschoben.

%Ausdruck%[Offset] {Anzahl} arbeitet im Prinzip wie die vorher beschriebene Option, mit der Abweichung, daß keine Ausgabedatei erzeugt wird, dieser Teil der Eingabe also ignoriert wird.

Nummer (eine einfache ganze Zahl) als Muster erzeugt eine Ausgabedatei aus den *Nummer* folgenden Zeilen.

Die Namen der Ausgabedateien bestehen aus dem *Prefix* und einer normalerweise zweistelligen Zahl. Der Standard für *Prefix* ist xx. Tritt Während der Ausführung von **csplit** ein Fehler auf, so werden die bis dahin angelegten Dateien gelöscht.

Optionen:

- s die Ausgabe der Ausgabedateigröße wird unterdrückt
- k bei einem Abbruch von **csplit** werden die bereits angelegten Dateien nicht gelöscht
- f *Prefix* die Ausgabedateien erhalten den *Prefix* als Namen
- n *Stellen* die Ausgabedateien erhalten Nummern mit der angegebenen Anzahl *Stellen*

Siehe auch:

split auf Seite 135

Autor:

Stuart Kemp und David MacKenzie

1.15 cut

Funktion:

cut schneidet bestimmte Teile aus den Zeilen einer Datei aus

Syntax:

```
cut -b Bereich [-n] [Datei ...]
cut -c Bereich [Datei ...]
cut -f Bereich [-d Trenner] [-s] [Datei ...]
```

Beschreibung:

cut liest aus den angegebenen Dateien oder von der Standardeingabe und gibt bestimmte Teile jeder Eingabezeile auf die Standardausgabe. Welcher Teil der Eingabezeile ausgegeben wird, hängt von der gewählten Option und der Wahl eines Bereiches ab. Ein *Bereich* ist eine Liste von einzelnen Zahlen oder Zahlenbereichen. Ein Zahlenbereich ist ein Ausdruck der Form ' $m-n$ '. Wird eine der Zahlen m oder n weggelassen, so wird der Zeilenanfang bzw. das Zeilenende angenommen.

Optionen:

- b *Bereich* gibt nur die Bytes (Zeichen) im *Bereich* aus. TAB und BACKSPACE werden als ein Zeichen behandelt
- c *Bereich* gibt nur die Zeichen im *Bereich* aus. Diese Option ist identisch mit der Option '-b'. TAB und BACKSPACE werden als ein Zeichen behandelt
- f *Bereich* gibt die Felder im *Bereich* aus. Die einzelnen Felder sind durch TAB getrennt.
- d *Trenner* benutzt den *Trenner* anstelle eines TAB bei der Option '-f'
- n ohne Funktion. Vorgesehen für spätere Unterstützung internationaler Zeichensätze mit mehreren Bytes pro Zeichen.
- s unterdrückt die Ausgabe von Zeilen, die den *Trenner* nicht enthalten

Autor:

David M. Ihnat und David MacKenzie

1.16 date

Funktion:

date schreibt oder setzt die Systemzeit

Syntax:

```
date [-u] [-s Datum] [+FORMAT] [MMDDhhmm[[CC]YY][.ss]]
```

Beschreibung:

Ohne ein Argument gibt **date** die aktuelle Systemzeit aus. Wenn **date** mit einem Argument aufgerufen wird, dessen erstes Zeichen ein '+' ist, wird das Datum diesem Argument entsprechend formatiert. Dabei können alle Parameter benutzt werden, die von der `strftime` C-Bibliotheksfunktion verstanden werden. Alle Zeichen, die keine Parameter sind, werden unverändert ausgegeben.

Die folgenden Parameter werden unterstützt:

%% ein einfaches %

%n das Zeilenende

%t ein Tabulator

Die Zeitfelder:

%H Stunden 00 bis 23

%I Stunden 00 bis 12

%M Minuten 00 bis 59

%p AM oder PM

%r die Zeit, 12 Stunden (hh:mm:ss AM/PM)

%S Sekunden 00 bis 59

%T die Zeit, 24 Stunden (hh:mm:ss)

%X die Zeit, 24 Stunden (%H:%M:%S)

%Z die Zeitzone; oder nichts, wenn keine Zeitzone feststellbar ist

Die Datenfelder:

%a der abgekürzte Wochentag

%A der ausgeschriebene Wochentag

%b der abgekürzte Monat

%B der ausgeschriebene Monat

%c das Datum und die Zeit (Standardausgabeformat)

%d der Tag im Monat 00 bis 31

%D das Datum (mm/dd/yy)

%h das Gleiche wie %b

%j der Tag im Jahr

%m der Monat 00 bis 12

%U die Nummer der Woche, Sonntag erster Tag

%w der Tag in der Woche 0 bis 6

%W die Nummer der Woche, Montag erster Tag

%x das Datum (mm/dd/yy)

%y die letzten beiden Stellen der Jahreszahl

%Y die Ganze Jahreszahl

Wenn dem **date** Kommando ein Argument übergeben wird, daß nicht mit einem ‘+’ Zeichen beginnt, wird die Systemzeit diesem Argument entsprechend gesetzt. Das Argument muß vollständig aus Ziffern bestehen. Die einzelnen Stellen der Zahl haben folgende Bedeutung:

MM Monat

DD Tag im Monat

hh Stunde

mm Minute

CC die ersten beiden Stellen der Jahreszahl (optional)

YY die letzten beiden Stellen der Jahreszahl (optional)

ss die Sekunden (optional)

Nur der Superuser (root) darf die Systemzeit verändern.

Optionen:

-s *Datum* (set) setzt die Zeit auf das *Datum*; das *Datum* kann Monatsnamen, Zeitzonen und ähnliches enthalten

-u zeigt (oder setzt) die Zeit als Greenwich Mean Time

Autor:

David MacKenzie

1.17 dd

Funktion:

dd (disk dump) konvertiert Dateien für verschiedene Speichermedien

Syntax:

dd [*if=Datei*] [*of=Datei*] [*ibs=Bytes*] [*obs=Bytes*] [*bs=Bytes*] [*cbs=Bytes*] [*skip=Blöcke*] [*seek=Blöcke*] [*count=Blöcke*] [*conv={ascii, ebcdic, ibm, block, unblock, lcase, ucase, swab, noerror, notrunc, sync}*]

Beschreibung:

dd liest eine Datei und schreibt den Inhalt mit wählbarer Blockgröße und verschiedenen Konvertierungen. Mit Hilfe dieses Kommandos können reguläre Dateien ebenso wie ganze Disketten oder Festplattenpartitionen kopiert werden.

Beispielsweise wird das Kommando **dd bs=8192 if=Image of=/dev/fd0** verwendet um aus der fertig übersetzten Kerneldatei (Image) eine Bootdiskette zu erzeugen.

Optionen:

if=Datei (input file) der Name der Eingabedatei (Voreingestellt ist die Standardeingabe)

of=Datei (output file) der Name der Ausgabedatei (Voreingestellt ist die Standardausgabe)

ibs=Schritt (input block size) Blockgröße der Eingabedatei

obs=Schritt (output block size) Blockgröße der Ausgabedatei

bs=Schritt (block size) Blockgröße für Ein- und Ausgabedatei

cbs=*Schritt* (conversion block size) Blockgröße für Konvertierung

skip=*Blocks* ignoriert am Anfang die angegebene Anzahl *Blocks* von der Eingabe

seek=*Blocks* unterdrückt am Anfang die Ausgabe der angegebenen Anzahl *Blocks*

count=*Blocks* kopiert die angegebene Anzahl *Blocks*

conv=*Konvertierung* ... Bestimmt die Art der Konvertierung. *Konvertierung* ist dabei eine von:

ascii konvertiert EBCDIC nach ASCII

ebcdic konvertiert ASCII nach EBCDIC

ibm konvertiert ASCII nach big blue special EBCDIC

block schreibt Zeilen in Felder der Größe *cbs* und ersetzt das Zeilenende durch Leerzeichen. Der Rest des Feldes wird ebenfalls mit Leerzeichen aufgefüllt

unblock ersetzt abschließende Leerzeichen eines Blocks der Größe *cbs* durch ein Zeilenende.

lcase wandelt Großbuchstaben in Kleinbuchstaben

ucase wandelt Kleinbuchstaben in Großbuchstaben

swab vertauscht je zwei Bytes der Eingabe. Wenn die Anzahl der gelesenen Bytes ungerade ist, wird das letzte Byte einfach kopiert.

noerror ignoriert Lesefehler

sync füllt Eingabeblocke bis zur Größe von *ibs* mit Nullen

Autor:

Paul Rubin, David MacKenzie und Stuart Kemp

1.18 df

Funktion:

df (disk free) zeigt den freien Festplattenplatz

Syntax:

df [-aikPv] [-t *Fstyp*] [--all] [--inodes] [--type *fstype*] [--kilobytes] [--portability] [*Pfad* ...]

Beschreibung:

df zeigt den freien Plattenplatz für das Dateisystem in in dem das Verzeichnis *Pfad* angesiedelt ist. Wenn kein Verzeichnis angegeben ist, wird der freie Platz für alle aufgesetzten Dateisysteme angezeigt. Die Angabe erfolgt in Kilobyte, wenn nicht die Umgebungsvariable POSIXLY_CORRECT gesetzt ist. In diesem Fall wird der freie Platz in 512-Byte Sektoren angezeigt.

Wird als *Pfad* der absolute Name der Gerätedatei eines aufgesetzten Dateisystems angegeben, so wird der freie Platz auf diesem Gerät (Partition einer Festplatte) angegeben und nicht der des Dateisystem in dem sich die Gerätedatei befindet. Der freie Platz auf einem abgesetzten Dateisystem ist auf diese Weise nicht zu ermitteln.

Optionen:

- a** (all) zeigt alle Dateisysteme an, einschließlich der mit 0 Bytes Kapazität und der vom Typ `ignore` oder `auto`
- i** (inodes) zeigt die Auslastung der Inodes anstelle der Blockauslastung
- k** (kilobytes) zeigt den freien Platz in Kilobyteblöcken auch wenn die Umgebungsvariable `POSIXLY_CORRECT` gesetzt ist
- P** (portability) erzwingt die Ausgabe in einer Zeile pro Dateisystem
- t** *Fstyp* (type) schränkt die Ausgabe auf die Dateisysteme vom Typ *Fstyp* ein
- v** ohne Funktion

Autor:

David MacKenzie

dir

dir arbeitet wie `'ls -C'`. Die mehrspaltige Ausgabe ist unabhängig vom Typ des Ausgabegerätes.

`dir` ist nicht mehr ein Link auf `ls`, sondern ein eigenes Kommando. Die Eigenschaft dieses Kommandos ändert sich durch Umbenennung nicht.

Die Verwendung und die Optionen entsprechen denen von `ls`. Eine Beschreibung ist auf Seite 109 nachzulesen.

1.19 dirname

Funktion:

dirname gibt den Pfadanteil eines vollständigen Dateinamens aus

Syntax:

dirname *Datei*

Autor:

David MacKenzie und Jim Meyering

1.20 doshell

Funktion:

doshell startet eine Shell auf einem (virtuellen) Port ohne `login`

Syntax:

doshell *Port Shell*

Beschreibung:

doshell ermöglicht es, auch auf den (virtuellen) Terminals auf denen von init kein getty-Prozeß gestartet wurde, und auf denen deshalb kein login möglich ist, eine Shell zu starten.

Die Umschaltung zwischen den virtuellen Terminals erfolgt mit den Tastenkombinationen ALT-F1 bis ALT-F8.

Autor:

Jim Wiegand

1.21 du

Funktion:

du (disk usage) zeigt die Verteilung des belegten Plattenplatzes auf die Verzeichnisse

Syntax:

du [-abcklsxDLS] [--all] [--total] [--count-links] [--summarize] [--bytes] [--kilobytes] [--one-file-system] [--separate-dirs] [--dereference] [--dereference-args][*Verzeichnis* ...]

Beschreibung:

du zeigt den belegten Plattenplatz für das *Verzeichnis* und für alle Unterverzeichnisse. Die Menge wird in Kilobyte angegeben, wenn nicht die Umgebungsvariable POSIXLY_CORRECT gesetzt ist. In diesem Fall wird die Menge in 512 Byte Blöcken angegeben.

Optionen:

- a** (all) zeigt auch den Platzbedarf aller Dateien
- b** (bytes) zeigt den Platzbedarf in Bytes
- c** zeigt den (summierten) Platzbedarf der in der Kommandozeile übergebenen Dateien
- k** (kilobytes) gibt den Platzbedarf in Kilobytes auch wenn die Umgebungsvariable POSIXLY_CORRECT gesetzt ist
- l** zählt die Größe der (harten) Links mit, auch wenn sie dadurch doppelt vorkommen
- s** gibt nur die Summe für jedes Verzeichnis in der Kommandozeile
- x** ignoriert Verzeichnisse, die in anderen Dateisystemen liegen
- D** folgt dem Verweis auf ein anderes Verzeichnis bei einem symbolischen Link, wenn dieser als Kommandozeilenargument übergeben wird. Andere symbolische Links werden nicht dereferenziert.
- L** alle symbolischen Links werden dereferenziert, das heißt es wird der Platzbedarf des referenzierten Verzeichnisses anstelle des Linkfiles gezeigt
- S** zeigt den Platzbedarf jedes Verzeichnisses einzeln, ohne die Unterverzeichnisse

Autor:

Torbjorn Granlund und David MacKenzie

echo

echo schreibt eine Zeichenkette in die Standardausgabe. Das **echo** Kommando aus dem GNU Shellutility Paket ist identisch mit dem **echo** Shellkommando, das auf der Seite 34 des Handbuchs beschrieben ist.

ex

ex ist ein Link auf den vi-Clone **elvis**. Wenn **elvis** unter diesem Namen aufgerufen wird, startet er im ‘colon mode’, wie der alte Unix Editor **ex**. Die Bedienung des Editors ist ab der Seite 60 im Handbuch erklärt.

1.22 egrep**Funktion:**

egrep gibt alle Zeilen aus in denen ein bestimmter Ausdruck gefunden wird

Syntax:

egrep [*−CVbchilnsvwX*] [*−Anzahl*] [*−AB Anzahl*] [*[−e] Ausdruck | −f Datei*][*Datei ...*]

Beschreibung:

egrep durchsucht die angegebenen *Dateien* (oder die Standardeingabe) nach einem *Ausdruck* und gibt die entsprechenden Zeilen aus. Der Status von **egrep** ist 0 wenn der *Ausdruck* gefunden wurde, und sonst 1.

egrep unterscheidet sich nur in der Syntax einiger Ausdrücke vom **grep** Kommando. Als *Ausdruck* akzeptiert **egrep** reguläre Ausdrücke mit den folgenden Steuerzeichen:

c ein einzelner Buchstabe paßt auf sich selbst

. ein Punkt paßt auf jeden Buchstaben außer auf das Zeilenende

? das dem Fragezeichen vorangehende Zeichen bzw. Muster kann null oder einmal vorkommen

***** das dem Asterisk ‘*’ vorangehende Zeichen oder Muster kann 0 mal oder öfter vorkommen

+ das dem Pluszeichen ‘+’ vorangehende Zeichen bzw. Muster kann 1 mal oder öfter vorkommen

| die durch den Operator ‘|’ verbundenen Argumente werden **oder** verknüpft

^ (Caret) paßt auf den Zeilenanfang

\$ paßt auf das Zeilenende

\< paßt auf den Wortanfang

\> paßt auf das Wortende

[Buchstaben] paßt auf alle *Buchstaben*; dabei können einzelne Buchstaben, aber auch Bereiche in der Form ‘*von–bis*’ angegeben werden; wenn der erste Buchstabe nach ‘[’ ein ‘^’ ist, paßt der Ausdruck auf alle Buchstaben, außer den Aufgeführten

() die Klammern fassen Ausdrücke und Zeichenketten zusammen; außerdem wird der auf den in Klammern eingeschlossene Teil des Musters passende Text markiert und mit einem folgenden '\N' Ausdruck referenziert (Tag)

\N referenziert die auf das in der N-ten runden Klammern eingeschlossene Muster passende Zeichenkette. Beispielsweise kann mit dem Muster '(.....).*\1' nach Zeilen gesucht werden, in denen eine Zeichenkette aus sieben Zeichen doppelt vorkommt.

\ jedes der Sonderzeichen kann durch ein '\' (Backslash) eingeleitet sich selbst suchen

\b paßt auf kein Zeichen sondern auf den Anfang oder das Ende eines Wortes

\B steht für den Raum innerhalb eines Wortes

\w paßt auf alle alphanumerischen Zeichen [A-Za-z0-9]

\W paßt auf alle nichtalphanumerischen Zeichen [^A-Za-z0-9]

Die Rangfolge der Operatoren ist (von der höchsten zur niedrigsten):

('(', ')', '?', '*', '+', '|')

Die anderen Operatoren sind mit den anderen Buchstaben gleichrangig.

Optionen:

-A *Anzahl* gibt *Anzahl* Zeilen Kontext **nach** jeder gefundenen Zeile aus

-B *Anzahl* gibt *Anzahl* Zeilen Kontext **vor** jeder gefundenen Zeile aus

-C gibt 2 Zeilen Kontext **vor und nach** jeder gefundenen Zeile aus

-*Anzahl* gibt *Anzahl* Zeilen Kontext **vor und nach** jeder gefundenen Zeile aus

-V gibt die Versionsnummer auf die Standardfehlerausgabe

-b gibt die Position jeder gefundenen Stelle mit aus

-c gibt nur die Gesamtzahl der gefundenen Stellen aus

-e *Ausdruck* sucht nach *Ausdruck*

-f *Datei* *Datei* enthält die Ausdrücke nach denen gesucht werden soll.

-h unterdrückt die Dateinamen vor jeder Fundstelle

-i ignoriert Groß und Kleinschreibung

-l gibt nur die Dateinamen mit Fundstellen aus

-n gibt die Zeilennummer zu jeder Fundstelle aus

-s (silent) keine Ausgabe außer Fehlermeldungen.

-v gibt nur Zeilen aus, die den *Ausdruck* nicht enthalten

-w gibt nur Zeilen aus, in denen der *Ausdruck* als komplettes Wort vorkommt

-x gibt nur Zeilen aus, die den *Ausdruck* als ganze Zeile enthalten

Siehe auch:

grep, Seite 91 und fgrep, Seite 83

Autor:

Mike Haertel, James A. Woods und David Olson

1.23 elvis

Funktion:

elvis ist eine Weiterentwicklung, des standard Unix Editors ex/vi.

Beschreibung:

elvis bietet sowohl im ‘visual mode’, als auch im ‘colon mode’ nahezu alle Möglichkeiten des ‘Vorbilds’ ex/vi. Wie ex/vi hält auch elvis den Hauptteil des Textes in einer temporären Datei und nicht im RAM. Dadurch können auch Dateien editiert werden, die wegen ihrer Größe nicht im Speicher gehalten werden können. Im Falle eines Systemabsturzes besteht so die Möglichkeit, den Inhalt des Arbeitspuffers wiederherzustellen. elvis ist ein public-Domain Programm und unterliegt keinen Beschränkungen in der Verwendung.

1.23.1 Überblick

Die Benutzerschnittstelle von elvis ist etwas gewöhnungsbedürftig. Es gibt zwei wichtige Kommandomodi und einige Eingabemodi. Jeder Kommandomodus hat einen Befehl, um in den jeweils anderen umzuschalten. Der meistverwendete Modus ist vermutlich der ‘visual command mode’. In diesem Modus befindet sich elvis normalerweise nach dem Starten.

Im ‘visual command mode’ dient der gesamte Bildschirm der Textdarstellung. Jeder Tastendruck wird als Teil eines Befehls interpretiert. Eingegebener Text wird **nicht** in die Datei eingefügt. Um Text einzufügen, muß erst ein ‘Text einfügen’ Kommando gegeben werden.

Der ‘colon mode’ unterscheidet sich vom ‘visual mode’ dadurch, daß in der letzten Zeile ein ‘:’ angezeigt wird. elvis erwartet dann die Eingabe eines Befehls gefolgt von einem RETURN. Im ‘colon mode’ erwartet elvis andere Kommandos als im ‘visual command mode’.

1.23.2 Der ‘visual mode’

‘Visual mode’ Befehle

Die meisten ‘visual command mode’ Befehle sind nur einen Tastendruck lang. Nachfolgende Tabelle listet die möglichen Befehle und ihre Optionen auf.

Zusätzlich zu den hier aufgeführten Kommandos interpretiert elvis die Cursortasten als entsprechende Positionierungsbefehle, sofern der entsprechende `termcap`-Eintrag korrekt ist. Hier treten gelegentlich Probleme auf. Gleiches gilt für die BILDAUF- und BILDAB-Tasten. Im ‘colon mode’ stellt elvis noch einen Befehl (:map) zur Verfügung, der es ermöglicht noch andere Tasten, wie z.B. die Funktionstasten, mit Kommandos zu belegen.

Noch ein Tip: der ‘visual command mode’ ist dem ‘text input mode’ sehr ähnlich. Um herauszufinden in welchem Modus man sich gerade befindet genügt es, einmal ESC zu drücken. Befindet man sich im ‘visual command mode’ piepst elvis einmal. Piepst elvis nicht, so hat man sich im ‘input mode’ befunden und elvis ist in den ‘visual command mode’ zurückgekehrt, d.h. das heißt nach einem ESC befindet elvis sich immer im ‘visual command mode’.

Anzahl Vielen Kommandos kann ein numerischer Parameter vorangestellt werden. Er besteht aus einer Folge von Ziffern, die als Dezimalzahl interpretiert werden. Dieser Parameter ist immer optional. Sein Standardwert ist meistent 1.

Taste Einige Kommandos erfordern zwei Tastendrücke. Der erste ist immer der Befehl, der zweite ein Parameter zum Befehl. So bedeutet *ma* beispielsweise, daß an die aktuelle Cursorposition die Marke *a* gesetzt wird.

Bereich Kommandos, die auf einen Textbereich wirken, benötigen zusätzlich zur aktuellen Cursorposition, die eine Grenze eines Bereichs angibt, eine zweite. *elvis* kennt drei Möglichkeiten, diese zweite Grenze anzugeben. Die erste ist, dem Befehl ein Bewegungskommando nachzustellen. So löscht *dw* ein Wort, und sowohl *3dw* als auch *d3w* löschen jeweils drei Worte. Als zweite Möglichkeit kann man den Befehl zweimal angeben. Er wirkt dann auf die ganze Zeile. Die dritte ist, den Cursor auf ein Ende des Bereichs zu setzen, *v* oder *V* einzugeben, den Cursor an das andere Ende des Bereichs zu bewegen und daraufhin den gewünschten Befehl einzugeben.

eing Bei einigen Kommandos ist es möglich, interaktiv Text einzugeben. Es erscheint entweder ein Prompt, oder *elvis* schaltet in den sog. 'input mode' (s. u.).

Text Bei Kommandos, die einen Parameter *Text* erwarten, kann der gewünschte Text in der letzten Bildschirmzeile interaktiv eingegeben werden. Die Texteingabe wird mit RETURN abgeschlossen.

BEW Diese Kommandos bewegen den Cursor, und können einem Befehl übergeben werden, der einen *Bereich*-Parameter erwartet.

EDIT Diese Kommandos modifizieren Text und können mit '.' wiederholt werden.

EXT Diese Befehle sind Erweiterungen von *elvis*, die bei 'ex/vi' nicht zur Verfügung stehen.

	BEFEHL	Beschreibung
	^A	sucht nach dem nächsten Vorkommen des Wortes, auf dem der Cursor steht (BEW) (ERW)
	^B	bewegt den Cursor seitenweise zum Dateianfang
	^C	— (sendet normalerweise ein SIGINT zum Abbrechen eines Kommandos)
<i>Anzahl</i>	^D	bewegt den Cursor Richtung Dateiode um <i>Anzahl</i> Zeilen (Standard: 1/2 Bildschirmseite)
<i>Anzahl</i>	^E	bewegt den Cursor Richtung Dateianfang <i>Anzahl</i> Zeilen
	^F	bewegt den Cursor Richtung Dateiode um 1 Bildschirmseite
	^G	zeigt den Dateistatus und die momentane Zeilennummer
<i>Anzahl</i>	^H	Cursor rechts (wie h) <i>Anzahl</i> Spalten (BEW)
	^I	—
<i>Anzahl</i>	^J	Cursor abwärts (wie j) <i>Anzahl</i> Zeilen (BEW)
	^K	—
	^L	erneuert den Bildschirm (wie ^R)
<i>Anzahl</i>	^M	bewegt den Cursor abwärts zum Zeilenanfang <i>Anzahl</i> Zeilen (BEW)
<i>Anzahl</i>	^N	Cursor abwärts <i>Anzahl</i> Zeilen (BEW)
	^O	—
<i>Anzahl</i>	^P	Cursor aufwärts <i>Anzahl</i> Zeilen (BEW)
	^Q	— (normalerweise XON, zum Fortsetzen des Bildschirmaufbaus)
	^R	erneuert den Bildschirm (wie ^L)

	^S		— (normalerweise XOFF, zum Anhalten des Bildschirmaufbaus)
	^T		—
<i>Anzahl</i>	^U		bewegt den Cursor Richtung Dateianfang um <i>Anzahl</i> Zeilen (Standard: 1/2 Bildschirmseite)
	^V		—
	^W		—
<i>Anzahl</i>	^X		Cursor zur Spalte <i>Anzahl</i> (BEW) (ERW)
<i>Anzahl</i>	^Y		Cursor abwärts <i>Anzahl</i> Zeilen
	^Z		— (SIGSUSP, zum suspendieren des momentanen Prozesses)
	ESC		—
	^\		— (normalerweise SIGQUIT, wird ignoriert)
	^]		wenn sich der Cursor auf einem tag-Namen befindet, gehe dorthin
	^^		zur vorherigen Datei
	^_		—
<i>Anzahl</i>	SPC		Cursor rechts <i>Anzahl</i> Spalten (wie l) (BEW)
	!	<i>Bereich</i>	führt die selektierten Zeilen einem externen Filter zu, der nachdem Kommando angegeben werden muß
	"	<i>Taste</i>	legt fest, welcher Zwischenpuffer als nächstes verwendet wird
<i>Anzahl</i>	#	+	erhöht eine Zahl (EDIT) (ERW)
	\$		bewegt den Cursor zum Zeilenende (BEW)
<i>Anzahl</i>	%		plaziert den Cursor auf die zugehörige () { } [] oder plaziert den Cursor auf <i>Anzahl</i> Prozent der Datei (BEW) (ERW)
<i>Anzahl</i>	&		wiederholt den letzten :s// Befehl an der momentanen Position (EDIT)
	'	<i>Taste</i>	bewegt den Cursor zu einer markierten Zeile (BEW)
<i>Anzahl</i>	(bewegt den Cursor <i>Anzahl</i> Sätze zurück (BEW)
<i>Anzahl</i>)		bewegt den Cursor <i>Anzahl</i> Sätze vorwärts (BEW)
	*		bewegt den Cursor zum nächsten Fehler in der Fehlerliste (ERW)
<i>Anzahl</i>	+		bewegt den Cursor zum Anfang der nächste Zeile (BEW)
<i>Anzahl</i>	-		bewegt den Cursor zum Anfang der vorhergehenden Zeile (BEW)
<i>Anzahl</i>	,		wiederholt das vorhergehende f, F, t, T Kommando, in umgekehrter Richtung (BEW)
<i>Anzahl</i>	.		wiederholt das vorhergehende EDIT Kommando
	/	<i>Text</i>	sucht vorwärts nach einem regulären Ausdruck <i>Text</i> (BEW)
	0		wenn nicht ein Teil von <i>Anzahl</i> , bewegt es den Cursor zum Zeilenanfang
	:	<i>Text</i>	führt ein einzelnes 'ex' Kommando aus
<i>Anzahl</i>	;		wiederholt das vorhergehende f, F, t, T Kommando (BEW)

	<	<i>Bereich</i>	schiebt den Text nach links (EDIT)
	=	<i>Bereich</i>	formatiert <i>Bereich</i> neu
	>	<i>Bereich</i>	schiebt den Text nach rechts (EDIT)
	?	<i>Text</i>	sucht rückwärts nach dem regulären Ausdruck <i>Text</i> (BEW)
	@	<i>Taste</i>	führt den Inhalt eines Zwischenspeichers als ‘vi’ Befehle aus
<i>Anzahl</i>	A	<i>eing</i>	hängt <i>eing</i> an das Zeilenende an (EDIT)
<i>Anzahl</i>	B		bewegt den Cursor ein Wort zurück (BEW)
	C	<i>eing</i>	ändert den Text bis Zeilenende in <i>eing</i> (EDIT)
	D		löscht den Text bis Zeilenende (EDIT)
<i>Anzahl</i>	E		bewegt den Cursor zum Wortende (BEW)
<i>Anzahl</i>	F	<i>Taste</i>	bewegt den Cursor nach links auf das Zeichen <i>Taste</i> (BEW)
<i>Anzahl</i>	G		bewegt den Cursor zur Zeile <i>Anzahl</i> (Standard: zur letzten Zeile) (BEW)
<i>Anzahl</i>	H		bewegt den Cursor zur Zeile <i>Anzahl</i> unter der ersten Bildschirmzeile (BEW)
<i>Anzahl</i>	I	<i>eing</i>	fügt den Text <i>eing</i> am Zeilenanfang ein (EDIT)
<i>Anzahl</i>	J		hängt die nachfolgenden <i>Anzahl</i> Zeilen an die momentane an (EDIT)
	K		dient in Verbindung mit dem Programm ctags dazu Schlüsselworte nachzuschlagen
<i>Anzahl</i>	L		positioniert den Cursor auf den Anfang der letzten Bildschirmzeile (BEW)
	M		positioniert den Cursor auf den Anfang der mittleren Bildschirmzeile (BEW)
	N		wiederholt das letzte Suchen in umgekehrter Richtung (BEW)
<i>Anzahl</i>	O	<i>eing</i>	eröffnet über der aktuellen Zeile <i>Anzahl</i> neue und fügt <i>eing</i> ein (EDIT)
	P		fügt ausgeschnittenen Text vor dem Cursor ein (EDIT)
	Q		schaltet in den ‘ex’ Modus
	R	<i>eing</i>	überschreibt vorhandenen Text mit <i>eing</i> (EDIT)
<i>Anzahl</i>	S	<i>eing</i>	ändert <i>Anzahl</i> Zeilen (wie <i>Anzahl</i> cc) (EDIT)
<i>Anzahl</i>	T	<i>Taste</i>	bewegt den Cursor nach links vor das Zeichen <i>Taste</i> (BEW)
	U		nimmt alle letzten Änderungen der momentanen Zeile zurück
	V		setzt den Anfang einer Zeilenmarkierung für c, d, y, <, >, !, \ (ERW)
<i>Anzahl</i>	W		bewegt den Cursor um <i>Anzahl</i> Worte vorwärts (BEW)
<i>Anzahl</i>	X		löscht <i>Anzahl</i> Zeichen links vom Cursor (EDIT)
<i>Anzahl</i>	Y		kopiert die aktuelle und <i>Anzahl</i> weitere Zeilen in einen Puffer
	Z	Z	speichert die Datei und beendet elvis
	\	<i>Bereich</i>	öffnet ein Pop-Up Menü zum ändern von Text (ERW)

	^		bewegt den Cursor zum Anfang der aktuellen Zeile (BEW)
<i>Anzahl</i>	_		bewegt den Cursor zum Anfang der aktuellen Zeile (BEW)
	'	<i>Taste</i>	bewegt den Cursor zu dem mit <i>Taste</i> markierten Zeichen (BEW)
<i>Anzahl</i>	a	<i>eing</i>	fügt Text hinter dem Cursor ein (EDIT)
<i>Anzahl</i>	b		bewegt den Cursor um <i>Anzahl</i> Worte zurück (BEW)
	c	<i>Bereich</i>	ändert den Text im Bereich <i>Bereich</i> (EDIT)
	d	<i>Bereich</i>	löscht den mit <i>Bereich</i> angegebenen Text (EDIT)
<i>Anzahl</i>	e		bewegt den Cursor zum Ende des Wortes (BEW)
<i>Anzahl</i>	f	<i>Taste</i>	bewegt den Cursor nach rechts auf das Zeichen <i>Taste</i> (BEW)
	g		—
<i>Anzahl</i>	h		bewegt den Cursor um <i>Anzahl</i> Zeichen nach rechts (BEW)
<i>Anzahl</i>	i	<i>eing</i>	fügt den Text <i>eing</i> nach dem Cursor ein (EDIT)
<i>Anzahl</i>	j		bewegt den Cursor um <i>Anzahl</i> Zeilen abwärts (BEW)
<i>Anzahl</i>	k		bewegt den Cursor um <i>Anzahl</i> Zeilen aufwärts (BEW)
<i>Anzahl</i>	l		bewegt den Cursor um <i>Anzahl</i> Zeichen nach links (BEW)
	m	<i>Taste</i>	markiert eine Zeile oder ein Zeichen
	n		wiederholt das letzte Suchen (BEW)
<i>Anzahl</i>	o	<i>eing</i>	eröffnet unter der aktuellen Zeile eine neue und fügt <i>eing</i> ein (EDIT)
	p		fügt ausgeschnittenen Text hinter dem Cursor ein (EDIT)
	q		—
<i>Anzahl</i>	r	<i>Taste</i>	ersetzt <i>Anzahl</i> Zeichen durch <i>Taste</i> (EDIT)
<i>Anzahl</i>	s	<i>eing</i>	ersetzt <i>Anzahl</i> Zeichen durch <i>eing</i> (EDIT)
<i>Anzahl</i>	t	<i>Taste</i>	bewegt den Cursor nach rechts vor das Zeichen <i>Taste</i> (BEW)
	u		nimmt das letzte EDIT Kommando zurück
	v		setzt den Anfang einer Markierung für c, d, y, <, >, !, \ (ERW)
<i>Anzahl</i>	w		bewegt den Cursor um <i>Anzahl</i> Worte vorwärts (BEW)
<i>Anzahl</i>	x		löscht das Zeichen auf dem sich der Cursor befindet (EDIT)
	y	<i>Bereich</i>	kopiert den mit <i>Bereich</i> angegebenen Text in einen Puffer
	z	<i>Taste</i>	bewegt die aktuelle Zeile zum Anfang(+) Ende(−) oder zur Mitte(.) des Bildschirms (BEW)
<i>Anzahl</i>	{		bewegt den Cursor <i>Anzahl</i> Absätze zurück (BEW)
<i>Anzahl</i>	 		bewegt den Cursor zur Spalte <i>Anzahl</i> (BEW)
<i>Anzahl</i>	}		bewegt den Cursor <i>Anzahl</i> Absätze vorwärts (BEW)
	DEL		— (normalerweise zu 'shift-X' übersetzt, das ein Zeichen löscht)

Eingabemodi

Im ‘visual mode’ läßt sich Text **nicht** eingeben, sondern es muß erst in einen sog. ‘input mode’ geschaltet werden. Dies geschieht mit den Befehlen A, C, I, O, R, S, a, i, o, s. Die S, s, C, c Kommandos setzen zeitweilig ein ‘\$’ an das Ende des Textbereiches, auf den sie wirken. Im ‘input mode’ werden alle Zeichen mit Ausnahme der folgenden in den Text eingefügt.

Befehl	Funktion
^A	fügt eine Kopie der letzten Texteingabe ein
^D	löscht ein Einrückungszeichen
^H	löscht das Zeichen vor dem Cursor
^L	baut den Bildschirm neu auf
^M	(Wagenrücklauf) fügt eine neue Zeile ein (^J, Zeilenvorschub)
^O	führt den nächsten Tastendruck als ‘visual mode’ Befehl aus (eingeschränkt)
^P	fügt den Inhalt des Zwischenpuffers ein
^R	wie ^L
^T	fügt ein Einrückungszeichen ein
^U	löscht alle eingegebenen Zeichen bis zum Zeilenanfang
^V	fügt den folgenden Tastendruck ein, auch wenn er eine Sonderbedeutung hat
^W	löscht alle eingegebenen Zeichen bis zum Wortanfang
^Z^Z	speichert die Datei und beendet <i>elvis</i>
^[(ESCAPE) schaltet in den ‘visual command mode’ zurück

Auf einigen Systemen kann mit ^S und ^Q die Bildschirmausgabe angehalten und wieder gestartet, oder mit ^C *elvis* abgebrochen werden. ^@ (das NULL Zeichen) kann nicht in den Text eingefügt werden.

Der ‘visual mode’ Befehl R schalten in den sog. ‘replace mode’. In diesem Modus werden vorhandene Zeichen durch die Eingegebenen ersetzt. Ist das Zeilenende erreicht, werden die folgenden Eingaben an die Zeile angefügt.

Die Cursortasten in den Eingabemodi

Im Gegensatz zu *ex/vi* ist es bei *elvis* möglich, in den Eingabemodi die Cursortasten zu verwenden. Desweiteren funktionieren auch die BILDAUF, BILDAB, POS1 und ENDE Tasten. Die ENTF Taste löscht ein einzelnes Zeichen im ‘input mode’ und die EINFG Taste schaltet zwischen ‘input mode’ und ‘replace mode’ um. Der Hauptvorteil dabei ist, daß sich *elvis*, solange man sich im ‘input mode’ befindet, fast wie jeder normale Editor verhält. Bei fast allen anderen Editoren befindet sich der Benutzer immer im ‘input’ oder ‘replace mode’, und kann die Cursortasten jederzeit verwenden. Zusammen mit der ^Z ^Z Befehlsfolge braucht man für einfache Änderungen nie in den ‘visual command mode’ zu schalten. Leider scheint *elvis* bezüglich der Funktion dieser Sondertasten, hohe Ansprüche an die Eintragungen in der Datei */etc/termcap* zu stellen. Deshalb bedarf es auf vielen Systemen einer Anpassung dieser Datei.

Digraphs

Ein Digraph ist ein Zeichen, das aus zwei anderen zusammengesetzt ist. *elvis* unterstützt Digraphs als Möglichkeit nicht-ASCII einzugeben. So sollte z.B. ein Apostroph und ein ‘e’ als ein é angezeigt und gespeichert werden.

Bedauerlicherweise existiert kein Standard für erweiterte ASCII-Zeichen. *elvis* kann so übersetzt werden, daß er entweder den PC-, den ISO-LATIN1-Zeichensatz, der vom X-Windows System verwendet wird, oder keinen von beiden unterstützt. Die Digraph-Tabelle kann mit dem ‘colon mode’ Befehl *:digraph* angezeigt und verändert werden. Bevor nicht das Kommando *:set digraph* eingegeben wurde, erkennt *elvis* Digraphs nicht. Um ein Digraph einzugeben, muß zuerst das eine Zeichen, dann ^H (BACKSPACE), dann das andere Zeichen eingegeben werden. *elvis* ersetzt dann das letzte Zeichen durch das Zusammengesetzte.

Abkürzungen

`elvis` kann Abkürzungen erweitern. Mit dem ‘colon mode’ Kommando `:abbr` kann eine Abkürzung definiert werden. Wird dann im ‘input mode’ die Abkürzung eingegeben, erweitert sie `elvis` sofort auf den vollen Ausdruck. `elvis` führt die Erweiterung durch, sobald das erste nicht alphanumerische Zeichen eingegeben wird. Um die Ersetzung zu verhindern, kann vor dem ersten nicht alphanumerischen Zeichen `^V` eingegeben werden.

Automatisches Einrücken

Wird mit der Option `:set autoindent` das automatische einrücken eingeschaltet, fügt `elvis` vor jedem Zeilenanfang automatisch soviel Leerraum ein, wie in der vorangegangenen Zeile verwendet wurde. Der Leerraum am Zeilenanfang läßt sich mit `^T` vergrößern und mit `^D` verkleinern. Wird die Option `:set noautotab` verwendet, fügt `elvis` statt Tabulatorzeichen Leerzeichen ein. Der ‘auto indent’ Modus von `elvis` ist mit dem von `ex/vi` nicht 100%ig identisch. `0^D` und `^^D` funktionieren nicht, `^U` löscht den gesamten Leerraum, und in einigen Situationen fügt `elvis` weniger oder mehr Leerraum ein, als `ex/vi`.

1.23.3 Der ‘colon mode’

‘Colon mode’ Befehle

Zeilen	Befehl	Argumente
	<code>ab[br]</code>	<i>[Kurzform] [erweiterteForm]</i>
<i>[Zeile]</i>	<code>a[ppend][!]</code>	
	<code>ar[gs]</code>	<i>[Dateien]</i>
	<code>cc</code>	<i>[Dateien]</i>
	<code>cd[!]</code>	<i>[Verzeichnis]</i>
<i>[Zeile][,Zeile]</i>	<code>c[hange]</code>	
	<code>chd[ir][!]</code>	<i>[Verzeichnis]</i>
<i>[Zeile][,Zeile]</i>	<code>co[py]</code>	<i>Zeile</i>
	<code>col[or]</code>	<i>[textart] [[bright] Farbe] [on Farbe]</i>
<i>[Zeile][,Zeile]</i>	<code>d[ele]te</code>	<i>[x]</i>
	<code>dig[raph][!]</code>	<i>[XX [Y]]</i>
	<code>e[dit][!]</code>	<i>[Datei]</i>
	<code>er[rlist][!]</code>	<i>[Fehlerdatei]</i>
	<code>f[ile]</code>	<i>[Datei]</i>
<i>[Zeile][,Zeile]</i>	<code>g[lobal]</code>	<i>/regAusdruck/Befehl</i>
<i>[Zeile]</i>	<code>i[n]sert</code>	
<i>[Zeile][,Zeile]</i>	<code>j[oin][!]</code>	
<i>[Zeile][,Zeile]</i>	<code>l[ist]</code>	
	<code>mak[e]</code>	<i>[Ziel]</i>
	<code>map[!]</code>	<i>Taste soll_übersetzt_werden_zu</i>
<i>[Zeile]</i>	<code>ma[rk]</code>	<i>x</i>
	<code>mk[exrc]</code>	
<i>[Zeile][,Zeile]</i>	<code>m[ove]</code>	<i>Zeile</i>
	<code>n[ext][!]</code>	<i>[Dateien]</i>
	<code>N[ext][!]</code>	
<i>[Zeile][,Zeile]</i>	<code>nu[mber]</code>	
<i>[Zeile][,Zeile]</i>	<code>p[rint]</code>	
<i>[Zeile]</i>	<code>pu[t]</code>	<i>x</i>

	q[uit][!]	
[Zeile]	r[ead]	<i>Datei</i>
	rew[ind][!]	
	se[t]	[<i>Option</i>]
	so[urce]	[<i>Datei</i>]
[Zeile][,Zeile]	s[ubstitute]	<i>/regAusdruck/Ersatz/[p][g][c]</i>
	ta[g][!]	<i>Tagname</i>
	una[bbr]	[<i>Abkürzung</i>]
	u[ndo]	
	unm[ap][!]	<i>Taste</i>
	ve[rsion]	
[Zeile][,Zeile]	v[globa]	<i>/regAusdruck/Befehl</i>
	vi[sual]	<i>Dateiname</i>
	wq	
[Zeile][,Zeile]	w[rite][!]	<i>[[>>]Datei]</i>
	x[i>t][!]	
[Zeile][,Zeile]	y[ank]	<i>x</i>
[Zeile][,Zeile]	!	<i>externerBefehl</i>
[Zeile][,Zeile]	<	
[Zeile][,Zeile]	=	
[Zeile][,Zeile]	>	
[Zeile][,Zeile]	&	
	@	<i>x</i>

Um ‘colon mode’ Kommandos zu verwenden, muß vom ‘visual mode’ in den ‘colon mode’ umgeschaltet werden. Dies geschieht mit ‘:’ für ein Kommando und mit Q bis zum nächsten :vi Kommando.

Zeilenangaben

Zeilenangaben sind immer optional. Die erste Zeilenangabe wird normalerweise als die momentane Zeile angenommen, die zweite wird gleich der ersten gesetzt. Ausnahmen hiervon sind die Befehle :write; :global und :vglobal, die sich, wenn nicht anders angegeben auf den gesamten Text beziehen und !, der standardmäßig auf keine Zeile wirkt. Zeilenangaben bestehen auf einem absoluten und/oder einem relativen Teil.

Der absolute Teil einer Zeilenangabe kann eine Zeilennummer, eine Marke, ein ‘.’, um die aktuelle Zeile zu bezeichnen, ein \$ um die letzte Zeile des Textes zu bezeichnen oder ein vorwärts oder rückwärts Suchen sein. Eine Zeilennummer ist eine Ziffernfolge, die als Dezimalzahl interpretiert wird. Eine Marke wird als ein ‘’ gefolgt von einem Buchstaben angegeben. Marken müssen gesetzt werden, bevor sie verwendet werden können. Im ‘visual mode’ kann eine Marke mit dem Befehl m gefolgt von einem Buchstaben gesetzt werden. Im ‘colon mode’ wird eine Marke mit dem :mark Befehl gesetzt. Ein vorwärts Suchen wird als ein von / eingeschlossener regulärer Ausdruck angegeben. Das Suchen beginnt in der aktuellen Zeile. Ein rückwärts Suchen wird als ein von ? eingeschlossener regulärer Ausdruck angegeben. Das Suchen beginnt in der vorhergehenden Zeile.

Der relative Teil einer Zeilenangabe wird als + oder – gefolgt von einer Dezimalzahl angegeben. Die Zahl wird von dem absoluten Teil abgezogen oder hinzuaddiert.

Ein Sonderfall ist das % Zeichen. Es wird dazu verwendet, alle Zeilen des Textes zu bezeichnen (es ist mit 1,\$ identisch).

Beispiele:

<code>:p</code>	gibt die aktuelle Zeile aus
<code>:37p</code>	gibt Zeile 37 aus
<code>:'gp</code>	gibt die Zeile, in der die Marke <i>g</i> gesetzt ist aus
<code>:/foo/p</code>	gibt die nächste Zeile, die <i>foo</i> enthält aus
<code>:\$p</code>	gibt die letzte Zeile des Textes aus
<code>:20,30p</code>	gibt die Zeilen 20—30 aus
<code>:!,\$p</code>	gibt den gesamten Text aus
<code>:%p</code>	gibt ebenfalls den gesamten Text aus
<code>:/foo/-2,+4p</code>	gibt 5 Zeilen um das nächste Vorkommen von <i>foo</i> aus

Texteingabe Kommandos

`[Zeile] append`
`[Zeile][,Zeile] change`
`[Zeile] insert`

Das Kommando **append** fügt Text hinter der angegebenen Zeile ein.

Das Kommando **insert** fügt Text vor der angegebenen Zeile ein.

Das Kommando **change** kopiert die angegebenen Zeilen in einen Zwischenpuffer, löscht sie und fügt an ihrer Stelle neuen Text ein.

Bei diesen Kommandos wird das Eingeben durch `^D` oder durch eine Zeile, die nur einen Punkt enthält beendet.

Ausschneiden und Einfügen

`[Zeile][,Zeile] delete [x]`
`[Zeile][,Zeile] yank [x]`
`[Zeile] put [x]`
`[Zeile][,Zeile] copy Zeile`
`[Zeile][,Zeile] to Zeile`
`[Zeile][,Zeile] move Zeile`

Das **delete** Kommando kopiert die angegebenen Zeilen in einen Zwischenpuffer, und löscht sie dann.

Das **yank** Kommando kopiert die angegebenen Zeilen in einen Zwischenpuffer, löscht sie aber nicht.

Das **put** Kommando fügt den Text eines Zwischenpuffers hinter der angegebenen Zeile ein. Bei diesen Kommandos ist *x* die optionale Angabe eines Zwischenpuffers, mit dem das Kommando arbeiten soll.

Die **copy** und **to** Kommandos kopieren den Text direkt hinter eine andere Zeile.

Das **move** Kommando löscht die angegebenen Zeilen und fügt sie sofort hinter einer anderen ein. Liegt die Zielzeile hinter den gelöschten Zeilen, so werden die Zeilennummern automatisch korrigiert.

Kommandos zum Anzeigen von Text

`[Zeile][,Zeile] print`
`[Zeile][,Zeile] list`
`[Zeile][,Zeile] number`

Das **print** Kommando stellt die angegebenen Zeilen auf dem Bildschirm dar.

Das **list** stellt die Zeilen ebenfalls dar, zeigt jedoch auch Control-Zeichen an.

Das **number** Kommando zeigt die Zeilen mit Zeilennummern an.

Kommandos, die auf den gesamten Text wirken

`[Zeile][,Zeile] global /regAusdruck/Befehl`
`[Zeile][,Zeile] vglobal /regAusdruck/Befehl`

Das **global** Kommando durchsucht die angegebenen Zeilen (oder die ganze Datei, wenn keine Zeilennummern angegeben wurden) nach dem regulären Ausdruck, positioniert den Cursor auf den entsprechenden Zeilen und führt *Befehl* darauf aus.

Das **vglobal** Kommando arbeitet im Prinzip genauso, nur führt es *Befehl* in allen Zeilen aus, die den regulären Ausdruck **nicht** enthalten.

Komandos zum Editieren einzelner Zeilen

```
[Zeile][,Zeile] join[!]  
[Zeile][,Zeile] ! programm  
[Zeile][,Zeile] <  
[Zeile][,Zeile] >  
[Zeile][,Zeile] substitute/regAusdruck/Ersatz/[p][g][c]  
[Zeile][,Zeile] &
```

Das **join** Kommando hängt alle angegebenen Zeilen aneinander. Wird nur eine Zeile angegeben, wird die darauffolgende Zeile angehängt. Der **join** Befehl fügt ein oder zwei Leerzeichen zwischen die Zeilen ein. Wird die Variante **join!** verwendet, unterbleibt dies.

Das **!** Kommando führt die entsprechenden Zeilen einem externen Filterkommando zu und ersetzt sie durch die Ausgabe des Filters. So würde beispielsweise das Kommando **!a,z!** **sort** die Zeilen zwischen den Marken *a* und *z* alphabetisch sortieren.

Die **<** und **>** Kommandos rücken die Zeilen um die Größe eines Tabulatorzeichens ein oder aus. Die Größe des Tabulators wird durch die Option **shiftwidth** festgelegt.

Der Befehl **substitute** sucht nach dem regulären Ausdruck und ersetzt ihn durch *Ersatz*. Die **p** Option gibt die geänderten Zeilen aus, die **c** Option fragt vor jedem Ersetzten, ob das Ersetzen durchgeführt werden soll. Wenn die **g** Option nicht angegeben wird, bearbeitet **elvis** nur das erste Auftreten des Ausdrucks in jeder Zeile.

Das **&** Kommando wiederholt das letzte Suchen und Ersetzen. Es sucht nach dem zuletzt eingegebenen Suchmuster (auch wenn es bei einem anderen Kommando angegeben wurde) und ersetzt es durch *Ersatz* des letzten **substitute** Befehls.

Der undo Befehl

Das **undo** Kommando nimmt die Änderungen des letzten Editierkommandos zurück.

Konfiguration und Status

```
map[!] [Taste soll_übersetzt_werden_zu]  
unmap[!] Taste  
abbr [Kurzform] [erweiterteForm]  
unabbr [Kurzform]  
digraph[!] [XX] [Y]  
set [Optionen]  
mkexrc  
[Zeile] mark x  
visual  
version  
[Zeile][,Zeile] =  
file [Datei]  
source Datei  
@ x  
color [textart] [[bright] Farbe] [[on] Farbe]
```

Mit dem **map** Kommando kann **elvis** so konfiguriert werden, daß er Funktions- und Sondertasten erkennt und ihnen eine benutzerdefinierte Funktion zuweist. Normalerweise wird die Übersetzung nur im **visual**

mode’ durchgeführt. Wird statt `:map` aber `:map!` angegeben, so führt `elvis` diese Funktion auch im ‘input’ oder ‘replace mode’ aus. Wird `:map` kein Argument übergeben, gibt es eine Liste der momentan aktiven Übersetzungstabelle aus. Wird der Befehl mit zwei Parametern aufgerufen, so ist der erste die Zeichenfolge, die die Taste tatsächlich sendet, und der zweite die Tastenfolge, die von `elvis` ausgeführt werden soll. Ist das erste Argument eine Zahl, so übersetzt `elvis` sie zu der entsprechenden Funktionstaste. So würde `map 5 dd` der Taste F5 das Kommando `dd` zuweisen, d.h. eine Zeile löschen.

Das `unmap` Kommando nimmt die mit `map` festgelegten Tastaturdefinitionen für die entsprechende Taste wieder zurück.

Das `abbr` Kommando dient dazu, eine Abkürzungstabelle anzuzeigen bzw. zu erstellen. Die Tabelle besteht aus den gewünschten Abkürzungen und den dazugehörigen ausgeschriebenen Worten. Im ‘input mode’ ersetzt `elvis` *Kurzform* durch *erweiterteForm* sobald nach *Kurzform* ein nicht alphanummerisches Zeichen eingegeben wird. Soll die Ersetzung verhindert werden, so muß als erstes Zeichen nach *Kurzform* ein `^V` eingegeben werden. Ohne Parameter gibt das Kommando die Tabelle aus. Mit zwei Parametern wird der erste als Abkürzung und der Rest der Zeile als ausgeschriebene Form betrachtet.

Das `unabbr` Kommando löscht Einträge aus der Abkürzungstabelle.

Das `digraph` Kommando erlaubt es dem Benutzer die von `elvis` verstandenen Digraphs anzuzeigen, zu erweitern oder einzelne Digraphs aus der Tabelle zu entfernen. Ohne Parameter wird die Tabelle angezeigt. Um ein Digraph zu setzen, müssen dem Befehl zwei Parameter übergeben werden. Der erste sind die beiden Zeichen, aus denen das Digraph zusammengesetzt ist, der zweite ist das nicht-ASCII Zeichen, das durch die beiden ersten dargestellt werden soll. Das höchstwertigste Bit des nicht-ASCII Zeichens wird von dem `digraph` Kommando automatisch gesetzt, wenn kein `!` an den Kommandonamen angehängt wird. Wird dem Kommando nur der erste Parameter übergeben, so wird das entsprechende Zeichen aus der Tabelle gelöscht.

Das `set` Kommando dient zum Setzen und Anzeigen der `elvis`-Optionen. Ohne Argumente zeigt es die geänderten Optionen an. Mit dem Argument `all` zeigt es den Zustand aller Optionen an. Ansonsten wird das Argument als eine Option, die gesetzt werden soll behandelt.

Der Befehl `mkexrc` speichert die momentane Konfiguration in einer Datei namens ‘.exrc’ im aktuellen Verzeichnis ab.

Das `mark` Kommando setzt an die angegebene Stelle eine Marke. Sie kann später dazu verwendet werden um in einem Kommando eine Zeile zu referenzieren.

Das `visual` Kommando schaltet aus dem ‘colon mode’ in den ‘visual mode’ zurück.

Der `version` Befehl gibt die Versionsnummer von `elvis` aus.

Das `=` Kommando gibt aus, welche Zeile angegeben wurde, oder, wenn ein Bereich angegeben wurde, den Anfangs- sowie den Endpunkt und die Anzahl der Zeilen, die dazwischen liegen.

Das Kommando `file` gibt den Dateinamen, die Anzahl der Zeilen und den Veränderungsstatus aus. Es kann auch dazu verwendet werden, den Dateinamen zu ändern.

Das `source` Kommando liest eine Folge ‘colon mode’ Befehle aus einer Datei ein und führt die Befehle aus.

Das `@` Kommando führt den Inhalt eines Zwischenspeichers als Befehle aus.

Der `color` Befehl funktioniert nur unter MS-DOS oder auf einem ANSI-Farbterminal. Er ermöglicht verschiedene Vorder- und Hintergrundfarben für verschiedene Texttypen (normal, fett, kursiv, das PopUp-Menü und die sichtbaren Makierungen) festzulegen. Standardmäßig ändert es die ‘normal’ Farben. Um die Farben für andere Textdarstellungen zu ändern, muß als erstes Argument der Anfangsbuchstabe des Texttyps angegeben werden (z.B. ‘`color bright yellow on blue`’ ändert die Standardtextdarstellung in hellgelben Text auf blauem Hintergrund; ‘`color b bright white`’ ändert die Textdarstellung von Fettdruck in hellweiße Schrift auf blauem Hintergrund). Die Hintergrundfarbe entspricht, wenn nicht angegeben, immer der momentanen Hintergrundfarbe. Nur in dem ersten `:color` Befehl muß sowohl die Vorder- als auch die Hintergrundfarbe für die normale Textdarstellung angegeben werden.

Kommandos zum Arbeiten mit mehreren Dateien

```
args [Dateien]
next[!] [Dateien]
Next[!]
```


previous[!]**rewind[!]**

Wenn **elvis** von der Kommandozeile der shell gestartet wird, werden alle Dateinamen, die übergeben werden in der Kommandozeilenliste gespeichert. Das **:args** Kommando zeigt die Kommandozeilenliste an oder definiert eine neue.

Das **:next** Kommando wechselt von einer Datei in der Kommandozeilenliste zur nächsten. Auch hier kann eine neue Kommandozeilenliste angegeben werden.

Das **:Next** und das **:previous** Kommando (sie sind völlig gleichbedeutend) schaltet zur vorhergehenden Datei.

Das **:rewind** Kommando schaltet zur ersten Datei in der Kommandozeilenliste.

Zwischen Dateien wechseln

edit[!] *Datei***tag[!]** *Tagname*

Das **:edit** Kommando dient zum Wechseln der Datei. Es hat nichts mit der Kommandozeilenliste zu tun.

Der **:tag** Befehl sucht den angegebenen Referenzpunkt *Tagname* in einer Datei namens tags. Diese Datei enthält eine Liste der verfügbaren Referenzpunkte und der Dateien, in denen sie sich befinden. **elvis** wechselt dann zu der Datei und platziert den Cursor auf dem Referenzpunkt. Eine solche 'tags'-Datei wird beispielsweise von dem Programm 'ctags' erstellt.

Arbeiten mit einem Compiler

cc [*Dateien*]**make** [*Ziel*]**errlist[!]** [*Fehlerliste*]

Das **:cc** und das **:make** Kommando starten den Compiler oder das **make**-Dienstprogramm und leiten deren Ausgabe in eine Fehlerdatei namens *Fehlerliste* um. Werden keine Dateien angegeben, wird dem Compiler die aktuelle Datei übergeben (sie muß vorher gespeichert werden). Der Inhalt der Fehlerdatei wird dann nach Fehlermeldungen durchsucht. Wird eine Fehlermeldung gefunden, wechselt **elvis** zu der Datei, in der der Fehler aufgetreten ist und platziert den Cursor in der entsprechende Zeile. In der Statuszeile von **elvis** wird die Fehlerbeschreibung angezeigt. Wurde ein Fehler behoben, wird der Cursor auf die nächste Zeile, in der ein Fehler aufgetreten ist, gesetzt. Im 'visual mode' geschieht das auch durch das '*' Kommando.

Es ist auch möglich außerhalb von **elvis** eine Fehlerdatei zu erstellen und **elvis** mit der **-m** Option zu starten. Der Cursor wird dann ebenfalls auf die Zeile gesetzt, in der der erste Fehler aufgetreten ist. Da in der Fehlerdatei gespeichert ist, in welcher Datei der Fehler aufgetreten ist, muß kein Dateinamen angegeben werden.

Wird das **:errlist** Kommando wiederholt ausgeführt, so versucht **elvis** die Änderungen der Zeilennummern durch das Einfügen oder Löschen von Zeilen zu berücksichtigen. Diese Korrekturen werden in der Annahme durchgeführt, daß die Datei vom Anfang zum Ende durchgearbeitet wird.

elvis beenden

quit[!]**wq****xit**

Das **:quit** Kommando beendet **elvis** ohne die Datei zu speichern.

Das **:wq** Kommando speichert die Datei und beendet danach **elvis**.

Das **:xit** Kommando arbeitet wie das **wq** Kommando, außer daß es die Datei nur speichert, wenn sie geändert wurde.

Datei Ein- und Ausgabekommandos

[Zeile] **read** *Datei*

[Zeile][,Zeile] **write**[!] [[>>] *Datei*]

Das **:read** Kommando liest Text aus einer anderen Datei ein und fügt ihn hinter der angegebenen Zeile an. Es kann ebenfalls die Ausgabe eines anderen Kommandos einlesen, indem dem Programmnamen ein **!** vorangestellt und anstelle von *Datei* verwendet wird.

Das **write** Kommando schreibt die gesamte Datei oder einen Teil in eine andere. Wird **!** angegeben, so wird die Datei geschrieben, selbst wenn das **readonly**-Flag gesetzt ist. Wenn dem Dateinamen '>>' vorangestellt wird, werden die Zeilen an die Datei angehängt. Die Ausgabe des **write** Befehls kann der Standardeingabe eines Programms zugeführt werden, wenn statt des Dateinamens der Programmname mit vorangestelltem **!** angegeben wird. Vorsicht: zwischen dem Ausrufezeichen und dem Programmnamen muß mindestens ein Leerzeichen stehen (**w!Dateiname**, aber **w!Programmname**).

Verzeichnis Kommandos

cd [*Verzeichnis*]

chdir [*Verzeichnis*]

shell

Die Kommandos **:cd** und **:chdir** wechseln das aktuelle Arbeitsverzeichnis (synonym).

Das Kommando **:shell** startet eine interaktive Shell.

Debugging Kommandos

[Zeile][,Zeile] **debug**[!]

validate[!]

Diese Kommandos stehen nur zur Verfügung, wenn **elvis** mit der **-DDEBUG** Option übersetzt wurde.

Das **:debug** Kommando gibt die Daten des Blockes aus, der die angegebenen Zeilen enthält. Wird **!** mit angegeben, so wird zusätzlich noch der Inhalt des Blockes angezeigt.

Das **:validate** Kommando überprüft bestimmte interne Variablen auf ihre Konsistenz. Normalerweise produziert das Kommando keine Ausgabe, wenn es keine Fehler entdeckt. Wird **!** angegeben, gibt es immer 'etwas' aus.

1.23.4 Reguläre Ausdrücke

elvis kann zum Suchen und Ersetzen reguläre Ausdrücke verwenden. Ein regulärer Ausdruck ist eine Zeichenfolge, in der einige Zeichen eine Sonderbedeutung haben. Durch die Verwendung regulärer Ausdrücke bietet sich die Möglichkeit, sehr komplizierten Suchaufgaben gerechtzuwerden.

Funktion

elvis' Funktion zum Auswerten regulärer Ausdrücke belegt die folgenden Zeichen (Metazeichen genannt) mit einer Sonderbedeutung.

\(unterAusdruck) Die Metazeichen **\(** und **\)** werden dazu verwendet Unterausdrücke in regulären Ausdrücken zu begrenzen. Trifft der reguläre Ausdruck auf einen Textbereich zu, so behält **elvis** auf welchen Teilbereich *unterAusdruck* zutrifft. Das Kommando **s/regAusdruck/neuerText/** benutzt diese Funktion.

^ Das **^** Metazeichen bezeichnet einen Zeilenanfang. Soll z. B. *foo* am Zeilenanfang gefunden werden, so ist der nötige reguläre Ausdruck **/^foo/**. **^** ist nur ein Metazeichen, wenn es am Anfang eines regulären Ausdrucks steht.

\$ Durch **\$** wird in einem regulären Ausdruck ein Zeilenende bezeichnet. **\$** ist nur ein Metazeichen, wenn es am Ende eines regulären Ausdrucks steht. **/\$\$/** würde demnach auf ein **\$** Zeichen am Zeilenende zutreffen.

- `\<` Das `\<` Metazeichen findet eine Zeichenkette der Länge null am Anfang eines Wortes. Eine Zeichenkette, die aus mindestens einem Buchstaben oder einer Zahl besteht, wird als Wort betrachtet. Ein Wort kann nach jedem nicht alphanumerischen Zeichen beginnen.
- `\>` Das `\>` Metazeichen findet eine Zeichenkette der Länge null am Ende eines Wortes. Ein Wort endet am Zeilenende oder vor einem nicht alphanumerischen Zeichen. Der reguläre Ausdruck `/\<ende\>/` würde jedes Vorkommen des Wortes *ende* im Text finden, ohne jedoch auch das Vorkommen von *ende* innerhalb eines anderen Wortes (z. B. *Kalender*) anzuzeigen.
- `.` Das Metazeichen `'.'` steht für jedes beliebige Zeichen.
- `[zeichenliste]` Dieser Ausdruck trifft auf jedes Zeichen zu, das in *zeichenliste* enthalten ist. In *zeichenliste* kann ein Zeichenbereich angegeben werden indem zwischen zwei Zeichen ein `-` gesetzt wird. `[a-zA-Z]` würde auf jeden Buchstaben zutreffen. Wird der *zeichenliste* ein `^` vorangestellt, so trifft sie jedes Zeichen, das nicht in ihr enthalten ist. `[^]` würde alle Zeichen außer dem Leerzeichen bezeichnen.
- `\{n\}` Dies ist ein Näherungsoperator, d. h. er kann nur nach einem Ausdruck angegeben werden, der ein einzelnes Zeichen liefert. Er gibt an, wie oft der Ausdruck, der das Zeichen zurückliefert, wiederholt werden soll. *n* ist hierbei die Anzahl der Wiederholungen. `/^-\{80\}\$/` bezeichnet Eine Zeile aus 80 Bindestrichen. `/\<[a-zA-Z]\{4\}\>/` bezeichnet jedes aus vier Buchstaben bestehende Wort.
- `\{n,m\}` Dies ist ein Näherungsoperator, d. h. er kann nur nach einem Ausdruck angegeben werden, der ein einzelnes Zeichen liefert. Er gibt an, wie oft der Ausdruck, der das Zeichen zurückliefert, wiederholt werden soll. *n,m* ist hierbei der Bereich, indem die Anzahl der Wiederholungen liegen muß. Wird *m* weggelassen (das Komma muß bleiben), so wird für *m* 'unendlich' eingesetzt. `/\<[a-zA-Z]\{4,6\}\>/` bezeichnet jedes aus vier, fünf oder sechs Buchstaben bestehende Wort.
- `*` Dies ist ein Näherungsoperator, d. h. er kann nur nach einem Ausdruck angegeben werden, der ein einzelnes Zeichen liefert und bezeichnet eine beliebige Anzahl Wiederholungen. Er ist gleichbedeutend mit `\{0,\}`.
- `\+` Dies ist ein Näherungsoperator, d. h. er kann nur nach einem Ausdruck angegeben werden, der ein einzelnes Zeichen liefert und bezeichnet eine beliebige Anzahl Wiederholungen, mindestens jedoch eine. Er ist gleichbedeutend mit `\{1,\}`.
- `\?` Dies ist ein Näherungsoperator, d. h. er kann nur nach einem Ausdruck angegeben werden, der ein einzelnes Zeichen liefert und drückt aus, daß der vorhergehende Ausdruck optional ist. Er ist gleichbedeutend mit `\{0,1\}`.

Alle anderen Zeichen werden nicht interpretiert und müssen dem Text exakt entsprechen. Um die Sonderbedeutung der Metazeichen aufzuheben, muß ihnen ein `\` vorangestellt werden.

Ersetzungen

Das `:s` Kommando benötigt mindestens zwei Argumente: einen regulären Ausdruck, und eine Zeichenkette, durch die der Text, auf den der reguläre Ausdruck zutrifft, ersetzt werden soll. Auch in der Ersetzungszeichenkette haben einige Zeichen eine Sonderbedeutung.

`&` fügt eine Kopie des Originaltextes ein

`~` (Tilde) fügt eine Kopie des vorhergehenden Ersetzungstextes ein

`\x` fügt eine Kopie des Originaltextes ein, auf den der *x*-te Unterausdruck `\(\)` zutrifft

`\U` konvertiert den Text der nächsten `&` und `\x` Kommandos in Großbuchstaben

`\L` konvertiert den Text der nächsten `&` und `\x` Kommandos in Kleinbuchstaben

`\E` Hebt die Wirkung des letzten `\U` oder `\L` Kommandos wieder auf

`\u` konvertiert das erste Zeichen des nächsten `&` oder `\x` in einen Großbuchstaben

`\l` konvertiert das erste Zeichen des nächsten `&` oder `\x` in einen Kleinbuchstaben

Um die Sonderbedeutung der Metazeichen aufzuheben, muß ihnen ein `\` vorangestellt werden. Ist die Option ‘nomagic’ gesetzt, so haben `&` und `~` keine Sonderbedeutung, außer ihnen wird ein `\` vorangestellt.

Optionen

`elvis` besitzt zwei Optionen, mit denen die Art, in der reguläre Ausdrücke verwendet werden, gesteuert werden kann. Die erste `[no]magic` ist eine boolean Option, die standardmäßig wahr ist. Solang diese Option eingeschaltet ist, haben alle Metazeichen die oben beschriebenen Bedeutungen. Wird die Option mit `:set nomagic` auf falsch gesetzt, so behalten nur `~` und `$` ihre Sonderbedeutung. Die andere Option ist ebenfalls eine boolean Variable, und heißt `[no]ignorecase`. Sie ist standardmäßig auf falsch gesetzt. Wird diese Option gesetzt, so unterscheidet `elvis` bei Suchkommandos nicht zwischen Klein- und Großbuchstaben.

Beispiele

Dieses Beispiel ändert jedes Vorkommen von ‘Egon’ in ‘Fritz’:

```
:%s/Egon/Fritz/g
```

Dieses Beispiel löscht den Leerraum am Zeilenende jeder Zeile (die eckigen Klammern enthalten ein Leerzeichen und ein Tabulatorzeichen):

```
:%s/[ ]\+$//
```

Dieses Beispiel ändert alle Buchstaben in der aktuellen Zeile in Großbuchstaben:

```
:s/*/\U&/
```

Dieses Beispiel unterstreicht alle Buchstaben der aktuellen Zeile, indem es sie in mit Hilfe eines Rückschritts unterstrichene Buchstaben umwandelt:

```
:s/[A-Za-z]/_<H&/g
```

Dieses Beispiel sucht den letzten Doppelpunkt einer Zeile und vertauscht den Text vor dem Doppelpunkt mit dem hinter dem Doppelpunkt. Das erste `\(\)` Paar dient dazu den Text vor dem Doppelpunkt einzulesen, das zweite liest den Text hinter dem Doppelpunkt. Die Metazeichen `\1` und `\2` werden in umgekehrter Reihenfolge eingegeben, um die Zeile am Doppelpunkt gespiegelt wieder auszugeben:

```
:s/(.*)\:(.*)/\2:\1/
```

1.23.5 Die Optionen von `elvis`

Die Optionen werden mit dem ‘colon mode’ Kommando `:set` gesetzt. Der Wert der Optionen beeinflusst das Verhalten der nachfolgenden Befehle. Der Bequemlichkeit zuliebe haben die Optionen einen langen Namen, der ihre Bedeutung beschreibt, und einen kurzen Namen, der schnell einzugeben ist. `elvis` versteht beide Namen. Es gibt drei verschiedene Arten Optionen: boolean, Zeichenketten und numerische. Boolean Optionen werden auf wahr gesetzt, indem ihr Name dem `:set` Kommando übergeben wird. Wird ihrem Namen ein `no` vorangestellt (z.B. `nomagic`) so werden sie auf falsch gesetzt. Zusätzlich gibt es die Möglichkeit boolean Optionen ein `neg` voranzustellen, wodurch ihr aktueller Status umgekehrt wird. Dies ist eine Erweiterung von `elvis`. Der originale `vi` kennt sie nicht. Um den Inhalt einer numerischen oder einer Zeichenkettenoption zu ändern, gibt man im `:set` Kommando den Namen gefolgt von einem ‘=’ und dem neuen Wert an (z.B. `:set tabstop=8`). Bei Zeichenkettenoptionen kann der Wert in Anführungszeichen eingeschlossen werden.

autoindent (ai) Wird die `autoindent` Option gesetzt, rückt `elvis` jede neue Zeile soweit wie die vorherige ein. Ohne diese Option, beginnt jede Zeile in der ersten Spalte.

autoprint (ap) Diese Option betrifft nur den ‘ex mode’. Ist diese Option gesetzt, gibt `elvis`, wenn der Cursor in eine neue Zeile bewegt wird oder das letzte Kommando die Datei verändert hat, die aktuelle Zeile aus.

autotab (at) Die Option bestimmt das Verhalten von `elvis` beim Einfügen von Leerraum an Zeilenanfang (`autoindent`). Ist die `autotab` Option gesetzt, so verwendet `elvis` eine Mischung aus Tabulatorzeichen und Leerzeichen um die richtige Menge Lerraum einzufügen. Ist die Option ausgeschaltet, so verwendet `elvis` nur Leerzeichen. Die `autotab` Option betrifft nur den den automatisch eingefügten Leerraum.

autowrite (aw) Soll von einer geänderten Datei, z.B. mit dem `:tag` oder dem `:next` Kommando, zu einer anderen geschaltet werden, so gibt `elvis` eine Fehlermeldung aus und wechselt die Datei nicht. Ist die `autowrite` Option gesetzt, so speichert `elvis` die Datei und wechselt zur nächsten.

beautify (bf) Diese Option löscht, wenn sie gesetzt ist, beim Laden der Datei alle Kontrollzeichen aus dem Text. Wird sie gesetzt, wenn schon eine Datei editiert wird, so ist sie bezüglich der aktuellen Datei wirkungslos.

cc (cc) Sie enthält den Namen des C-Compilers, meistens `cc -c` oder `gcc -c`.

charattr (ca) Viele Textverarbeitungsprogramme erlauben Text unterstrichen, fett oder kursiv darzustellen, in dem in den Text Formatkommandos wie `'\fu'`, `'\fb'` oder `'\fi'` eingefügt werden. Normalerweise behandelt `elvis` diese Kommandos wie normalen Text. Ist die Option `charattr` gesetzt, so interpretiert `elvis` diese Befehle und zeigt den Text in der entsprechenden Darstellungsweise an, wenn das Terminal dies unterstützt und in der Datei `/etc/termcap` die richtigen Einträge existieren.

columns (co) Diese Option enthält die Anzahl der Spalten auf dem Bildschirm.

digraph (dig) Diese Option steuert, ob Digraphs erkannt werden. Der Standartwert ist `nodigraph`, was bedeutet, daß `elvis` keine Sonderzeichen darstellt.

directory (dir) `elvis` speichert den zu bearbeitenden Text in Temporärdateien. Diese Option gibt an in welchem Verzeichnis sie angelegt werden sollen. Diese Option kann nur in der Datei `.exrc` gesetzt werden, da `elvis` nach dem abarbeiten dieser Datei schon Temporärdateien anlegt, d.h. das Ändern der Option käme zu spät.

edcompatible (ed) Diese Option beeinflusst das Verhalten des `substitute` Kommandos. Normalerweise ist diese Option ausgeschaltet, was dazu führt, daß alle Optionen des `substitute` Kommandos als nicht gesetzt betrachtet werden, wenn sie nicht explizit angegeben sind. Ist diese Option eingeschaltet, so verwendet `elvis` die Optionen des vorherigen `substitute` Kommandos bis sie explizit geändert werden.

equalprg (ep) Diese Option enthält den Namen und die Kommandozeilenoptionen des Formatierers, der für das `=` Kommando verwendet werden soll. Die Standardeinstellung ist `fmt`, so daß mit dem `=` Kommando der Text auf 80 Zeichen pro Zeile formatiert wird.

errorbells (eb) Normalerweise piepst `elvis`, wenn etwas Falsches eingegeben wird. Mit dieser Option wird der Warnton abgeschaltet.

exrc Diese Option gibt an, ob eine `.exrc` Datei im momentanen Verzeichnis beim Starten von `elvis` ausgeführt werden soll. Wird diese Option in der Datei `.exrc` im Heimatverzeichnis eingeschaltet, so versucht `elvis` die Datei `.exrc` im aktuellen Verzeichnis auszuführen. Diese Option ist hauptsächlich als Schutz gedacht. Sollte z. B. ein böser Mensch auf den Gedanken kommen mit `echo >/tmp/.exrc 'lrm -rf $HOME'` im Verzeichnis `/tmp` eine Datei `.exrc` zu erstellen, so wird ein Benutzer, der in `/tmp` eine Datei editieren möchte alle Dateien in seinem Heimatverzeichnis verlieren.

exrefresh (er) Im `'ex mode'` gibt `elvis` alle Zeilen einzeln auf dem Bildschirm aus. Wird diese Option gesetzt, so schreibt `elvis` alle Zeilen auf einmal. Ist z.B. der `write()` Systemaufruf sehr Prozessorzeit intensiv, oder wird eine Fensterumgebung verwendet, so kann es sich anbieten diese Option auf falsch zu setzen, da einige Fensterumgebungen den Text wesentlich schneller ausgeben, wenn mehrere Zeilen auf einmal geschrieben werden. Diese Option hat keinen Einfluß auf den `'visual command mode'` oder den `'input mode'`.

flash (vbell) Unterstützt der Termcapeintrag eine optische Alternative, zum Warnton, so wird diese Option gesetzt. Ist keine Unterstützung vorhanden, so wird sie von `elvis` auf falsch gesetzt und läßt sich auch nicht einschalten.

flipcase (fc) Diese Option steuert die Umwandlung von Groß- in Kleinbuchstaben und umgekehrt bei nicht ASCII Zeichen. Die angegebene Zeichenkette wird als Zeichenpaare interpretiert. Wird der `~` Befehl auf einem nicht ASCII Zeichen ausgeführt, so vergleicht `elvis` das Zeichen mit der Liste und ersetzt es durch das andere des Paares.

hideformat (hf) Viele Textformatierer erwarten im Text Formatkommandos, die mit einem `'` beginnen. Normalerweise zeigt `elvis` diese Zeilen wie normalen Text an. Ist die `hideformat` Option gesetzt, so werden diese Zeilen als Leerzeilen angezeigt.

ignorecase (ic) Normalerweise unterscheidet `elvis` bei Textsuchen zwischen Groß- und Kleinbuchstaben. Mit dieser Option läßt sich diese Unterscheidung abschalten.

inputmode (im) Diese Option sollte in `.exrc` gesetzt werden und bringt `elvis` dazu im `'input mode'` zu starten. Das Verlassen des `'input modes'` mit ESC ist nachwievor möglich.

keytime (kt) Auf den meisten Terminals liefern die Cursortasten zusammengesetzte Tastaturcodes. Diese Übertragung benötigt eine bestimmte Zeit, bis die komplette Sequenz eingegangen ist. Die `keytime` Option erlaubt die maximale Übertragungszeit für die komplette Sequenz anzugeben. Auf den meisten Systemen gibt erfolgt (wie bei Linux) die Angabe in zehntel Sekunden zwischen den einzelnen Zeichen. Die `keytime` Option auf 1 zu setzen sollte vermieden werden, da viele Systeme nur die Anzahl der CPU-Takte zählen, und so, wenn eine Taste kurz vor Beginn eines neuen Takts gedrückt wird, kaum Zeit zum Einlesen der Sequenz zur Verfügung steht. Hat das System relativ lange Antwortzeiten, oder läuft `elvis` unter einer grafischen Benutzeroberfläche, so sollte bei `keytime` mindestens eine Sekunde angegeben werden. Als Sonderfall kann `keytime` auf 0 gesetzt werden. Dadurch wird das Timeout abgeschaltet, was dazuführen kann, daß `elvis`, wenn die Cursortasten Escape-Sequenzen senden, ewig wartet und nicht in den `'command mode'` zurückkehrt. Diese Option ist eine Erweiterung der Timeout-Option bei `ex/vi`.

keywordprg (kp) `elvis` hat eine besondere Schlüsselwortfunktion. Befindet sich der Cursor auf einem Wort, so kann der Benutzer SHIFT-K eingeben, und `elvis` benutzt ein anderes Programm, um das Schlüsselwort nachzuschlagen und zusätzliche Informationen anzuzeigen. Diese Option gibt an, welches Programm verwendet werden soll. Der Standardwert ist `"ref"`. Dieses Programm schlägt Definitionen von C-Funktionen in einer Datei namens `refs` nach, die von dem Programm `ctags` generiert wurde. Dieser Programmaufruf kann durch einen anderen ersetzt werden z. B. durch eine Rechtschreibhilfe oder durch ein Online-Manual. `elvis` startet das Programm mit dem Schlüsselwort als einzigem Kommandozeilenparameter. Das Programm sollte seine Ausgabe auf die Standardausgabe schreiben und als Status 0 zurückliefern.

lines (ln) Diese Option enthält die Anzahl der Bildschirmzeilen.

list (li) Im `nolist` Modus zeigt `elvis` den Text `'normal'` auf dem Bildschirm an, d. B. Tabulatorzeichen werden zu einer bestimmten Anzahl Leerzeichen umgewandelt. Wird die `list` Option gesetzt, so zeigt `elvis` für jedes Tabulatozeichen `^I` und am Zeilenende ein `$` an.

magic (ma) `elvis'` Suchmechanismus kann reguläre Ausdrücke auswerten. Reguläre Ausdrücke sind Zeichenketten, in denen einige Zeichen eine Sonderbedeutung haben. Normalerweise ist die `magic` Option gesetzt d. h. `elvis` weist einigen Zeichen Sonderbedeutungen zu. Wird diese Option ausgeschaltet, so verlieren alle Zeichen außer `^` und `$` ihre Sonderbedeutung.

make (mak) Das `:make` Kommando startet das `"make"` Dienstprogramm. Die `make` Option enthält den Namen und die Kommandozeilenoptionen des Dienstprogramms.

mesg Diese Option wird von `elvis` ignoriert.

modelines (ml) elvis unterstützt Moduszeilen. Moduszeilen sind Zeilen am Anfang oder Ende des Textes, die typischerweise Einträge wie "ex:set ts=5 ca kp=spell wm=15" enthalten. Anderer Text darf ebenfalls in Moduszeilen vorkommen, so daß sie z. B. als Kommentar geschrieben werden können /* "ex:set ts=5 ca kp=spell wm=15" */. Normalerweise werden die Moduszeilen aus Sicherheitsgründen ignoriert. Diese Option muß in .exrc gesetzt werden.

nearscroll (ns) Die Zeile, auf der der Cursor steht, ist immer auf dem Bildschirm. Wird der Cursor zu einer Zeile bewegt, die sich nicht auf dem Bildschirm befindet so scrollt elvis, wenn die Zeile nicht weit entfernt ist, oder er baut den Bildschirm neu auf. Die nearscroll Option definiert, was von elvis als 'weiter entfernt' verstanden werden soll. Wird die Option z. B. auf 1 gesetzt, so scrollt elvis maximal eine Zeile. Setzt man die Option auf 0, so wird das Scrollen abgeschaltet und elvis baut den Bildschirm immer neu auf.

novice (nov) Das Kommando :set novice ist synonym zu :set nomagic report=1 showmode.

number (nu) Über die Option number wird gesteuert, ob elvis vor jeder Zeile eine Zeilennummer ausgibt. Die Zeilennummern sind nicht Teil des Textes. Wird die Datei gespeichert, so wird sie ohne Zeilennummern auf die Festplatte geschrieben.

paragraphs (pa) Die { und } Kommandos bewegen den Cursor jeweils um einen Absatz vorwärts oder zurück. Absätze können durch Leerzeilen oder 'Punktcommandos' eines Textformatierers voneinander getrennt werden. Die Option paragraphs ermöglicht elvis so zu konfigurieren, daß er mit verschiedenen Textformatierern korrekt zusammenarbeitet. elvis geht davon aus, daß ein Formatkommando aus einem '.' mit einem oder zwei anschließenden Zeichen besteht. Die paragraphs Option besteht aus einer Zeichenkette, in der jedes Zeichenpaar eine Befehlskombination des Absatzkommandos des Formatierers darstellt.

more Wenn elvis im 'visual mode' mehrere Meldungen in der letzten Bildschirmzeile ausgeben muß, so wartet er mit dem Anzeigen der nächsten bis eine Taste gedrückt wurde. Wird die Option auf falsch gesetzt, so wartet elvis nicht. Das bedeutet, daß nur die letzte Meldung gelesen werden kann. Sie ist aber meistens auch die Wichtigste.

prompt (pr) Wird die Option auf falsch gesetzt, so gibt elvis kein ':' mehr aus, wenn er die Eingabe eines ex-Kommandos erwartet. Diese Option ist nur hilfreich, wenn elvis auf einer extrem langsamen Maschine verwendet wird.

readonly (ro) Normalerweise erlaubt elvis das Schreiben jeglicher Dateien, auf die der Benutzer Schreibberechtigung besitzt. Besitzt der Benutzer keine Schreibberechtigung auf die Datei, so kann die geänderte Datei nur in eine andere geschrieben werden. Wird die readonly Option gesetzt, so geht elvis davon aus, daß auf keine Datei geschrieben werden darf. Diese Option ist insbesondere hilfreich, wenn elvis nur zum Anzeigen von Dateien verwendet werden soll. So wird verhindert, daß eine Datei versehentlich geändert wird. Diese Option ist normalerweise ausgeschaltet, außer elvis wird als view gestartet.

remap Über diese Option wird gesteuert, wie elvis sich verhält, wenn in einer :map Angabe Textabschnitte vorkommen, die schon als Mapeintrag existieren. Ist die Option eingeschaltet, so werden Mapeinträge innerhalb eines :map Kommandos wie Tastendrücke behandelt. So würde z. B. :map A B und :map B C dazu führen, daß A zu C übersetzt wird, falls die Option eingeschaltet ist.

report (re) Beim Editieren können sich Kommandos auf mehrere Zeilen beziehen. Ändert ein Kommando viele Zeilen, so gibt elvis eine Meldung aus, wieviele Zeilen geändert wurden. Diese Option steuert, ab wievielen geänderten Zeilen elvis eine Meldung ausgibt.

ruler (ru) Wird diese Option eingeschaltet, so gibt elvis in der letzten Zeile ständig die momentane Cursorposition in Form von Zeile,Spalte aus.

scroll (sc) Die Kommandos ^U und ^D scrollen normalerweise im Text einen halben Bildschirm vorwärts oder zurück. Dies läßt sich mit dieser Option ändern. Sie enthält die Anzahl Zeilen, um die gescrollt werden soll. Wird das Kommando ^U oder ^D mit einem Anzahl Parameter aufgerufen, so wird diese Option gleich Anzahl gesetzt.

sections (se) Die [und] Kommandos bewegen den Cursor jeweils um ein Kapitel vorwärts oder zurück. Kapitel können durch Leerzeilen oder ‘Punktkommandos’ eines Textformatierers voneinander getrennt werden. Die Option **paragraphs** ermöglicht **elvis** so zu konfigurieren, daß er mit verschiedenen Textformatierern korrekt zusammenarbeitet. **elvis** geht davon aus, daß ein Formatkommando aus einem ‘.’ mit einem oder zwei anschließenden Zeichen besteht. Die **section** Option besteht aus einer Zeichenkette, in der jedes Zeichenpaar eine Befehlskombination des Kapitelkommandos des Formatierers darstellt.

shell (sh) Startet **elvis** ein Shell beispielsweise durch ein ! oder :shell Kommando, so wird das Kommando dieser Option ausgeführt. Der Standardwert ist “/bin/sh”, außer die SHELL Variable ist gesetzt. In diesem Fall wird das in dieser Variable genannte Kommando als Standardwert angenommen.

shiftwidth (sw) Mit den Kommandos < und > können Zeilen um eine bestimmte Anzahl Spalten ein oder ausgerückt werden. Diese Option enthält die Anzahl der Spalten, um die ein- oder ausgerückt werden soll.

showmatch (sm) Wird die Option showmatch gesetzt, so bewegt **elvis** im ‘input mode’, wenn eine Klammer { } [] () eingegeben wird, den Cursor kurzzeitig zur jeweils zugehörigen anderen Klammer des Paares.

showmode (smd) Im ‘visual mode’ ist es leicht möglich zu vergessen, in welchem Modus sich **elvis** gerade befindet. Wird diese Option eingeschaltet, so zeigt **elvis** in der rechten unteren Ecke des Bildschirms ständig eine Nachricht an, die den momentanen Modus angibt.

sidescroll (ss) Bei langen Zeilen scrollt **elvis** den Bildschirm seitwärts. (Hierin unterscheidet er sich von ex/vi, der lange Zeilen über mehrere Zeilen verteilt darstellt.) Um die Anzahl der Scrolloperationen zu verkleinern, scrollt **elvis** den Bildschirm immer um mehrere Spalten. Diese Option enthält die Anzahl der Spalten, um die **elvis** den Bildschirm bewegt. Diese Option kann umso kleiner eingestellt werden, je schneller der Rechner den Bildschirm scrollen kann.

sync (sy) Falls das System abstürzt, kann der größte Teil einer geänderten Datei mit Hilfe der Temporärdatei, die **elvis** verwendet um Änderungen zu speichern, wiederhergestellt werden. Trotzdem werden von dem Betriebssystem nicht alle Änderungen sofort auf die Festplatte geschrieben. Über die sync Option läßt sich festlegen, ob **elvis** die Speicherung der Änderungen dem Betriebssystem überläßt oder ob nach jeder Änderung ein sync() aufgerufen wird. Letzteres ist wesentlich langsamer und auf Mehrbenutzersystemen geradezu unfreundlich, da es die gesamte Systemleistung reduziert.

tabstop (ts) Normalerweise ist ein Tabulatorzeichen 8 Spalten breit. Über diese Option läßt sich ein anderer Wert einstellen.

taglength (tl) Diese Option bestimmt die Anzahl der signifikanten Zeichen beim Nachschlagen eines Schlüsselwortes. Als Sonderfall kann die Option auf 0 gesetzt werden. Dann müssen alle Zeichen auf das Schlüsselwort zutreffen.

term (te) Diese Option kann nur gelesen werden und gibt an, welchen Terminaleintrag aus /etc/termcap **elvis** verwendet.

terse (tr) Ex/vi verwendet diese Option um festzulegen ob lange oder kurze Meldungen ausgegeben werden sollen. **elvis** hat nur einen Satz Meldungen, d.h. diese Option ist wirkungslos.

timeout (to) Das Kommando :set notimeout ist gleichbedeutend mit :set keytime=0. Das Kommando :set timeout ist gleichbedeutend mit :set keytime=1. Dadurch wird das Verhalten der ESC-Taste festgelegt.

warn (wa) Wurde eine Datei geändert ohne gespeichert zu werden, so gibt **elvis** vor einem !Befehl Kommando eine Warnung aus. Ebenso gibt **elvis** eine Meldung nach einem erfolgreichen Suchen, das das Dateiende überschritten hat, aus. Die nowarn Option unterbindet dieses Verhalten.

window (wi) Diese Option bestimmt, wieviele Zeilen des Bildschirms nach einem langen Bewegungskommando neu aufgebaut werden. Auf schnellen Terminals wird diese Option auf die Anzahl der Bildschirmzeilen minus eins gesetzt. Damit wird der gesamte Bildschirm um den Cursor herum zur Textausgabe genutzt. Auf einem langsamen Terminal bietet es sich an diese Zahl zu reduzieren, damit beispielsweise ein n Kommando nach einem Suchen schneller ausgeführt werden kann.

warmargin (wm) Normalerweise können sehr lange Zeilen eingegeben werden. Wird die Option auf einen anderen Wert als 0 gesetzt, so fügt *elvis* automatisch einen Zeilenumbruch an einem Wortanfang ein, wenn die Zeile zu nah an den rechten Bildschirmrand herankommt. So würde auf einem 80-Zeichen Bildschirm das Kommando `:set wm=10` dazu führen, daß alle Zeilen auf höchstens 70 Zeichen pro Zeile umbrochen werden.

warp scan (ws) Wird nach einer Textstelle gesucht, so findet sie *elvis* unabhängig von der Position innerhalb der Datei. Erreicht *elvis* beim Suchen das Dateende, so zeigt er in der linken unteren Bildschirmecke die Meldung “warped” und setzt das Suchen in der ersten Zeile der Datei fort. Wird die **warp scan** Option ausgeschaltet, so beendet *elvis* das Suchen am Dateende.

write any (wr) Das Ausschalten dieser Option schützt existierende Dateien vor versehentlichem Überschreiben. So würde das Kommando `:w foo` nicht ausgeführt werden, wenn die Datei *foo* schon existiert. Das Kommando `:w! foo` wird unabhängig von der Einstellung der Option immer ausgeführt.

1.23.6 Zwischenspeicher (Puffer)

Wenn *elvis* Text löscht, wird er in einen Puffer kopiert. Dies geschieht sowohl im ‘visual mode’, als auch im ‘ex mode’. Es gibt keine Grenze, wieviel Text in einem Zwischenspeicher gespeichert werden kann. *elvis* besitzt 36 Zwischenpuffer, 26 mit Namen **a—z**, 9 unbenannte **1—9** und einen besonderen ‘.’. Im ‘ex mode’ verwenden die Kommandos `:move` und `:copy` einen Puffer um den Text zwischenzuspeichern.

Text zwischenspeichern

Im ‘visual mode’ benutzen die Kommandos **d**, **y**, **c**, **C**, **s** und **x** Kommandos die Zwischenspeicher **1—9**. Standardmäßig wird der Text in Puffer 1 gespeichert. Der Inhalt des ersten Puffers kommt in den zweiten. Der Inhalt des zweiten in den dritten. Der Inhalt des neunten Puffers geht verloren. Auf diese Weise bleibt der Text, der in den letzten neun Kommandos gelöscht wurde erhalten.

Ebenso kann Text in einen benannten Zwischenspeicher kopiert werden. Damit der Text in den gewünschten Puffer kopiert wird, muß der Name des Puffers mit einem doppelten Anführungszeichen vor dem Befehl angegeben werden. In diesem Fall werden die Puffer **1—9** nicht verändert.

Um Text an einen Puffer anzuhängen wird sein Name als Großbuchstabe vor dem Kommando angegeben. Der Puffer ‘.’ hat eine Sonderbedeutung. Er enthält die Eingaben, des letzten ‘input modes’. Er wird dazu verwendet das ‘.’ und **^A** Kommando zu implementieren. Um im ‘ex mode’ zusammen mit den Kommandos `:delete`, `:change` und `:yank` Text in einen benannten Puffer zu kopieren, muß der Puffer nach dem Befehl angegeben werden (z.B. `:20,30y a`). Auf die Puffer **2—9** und ‘.’ kann nicht direkt, schreibend zugegriffen werden.

Einfügen aus einem Puffer

Es gibt zwei Arten Text einzufügen: den Zeilenmodus und den Zeichenmodus. Enthält ein Puffer ganze Zeilen (z.B. von einem **dd** Kommando), so wird der Zeilenmodus verwendet. Enthält ein Puffer nur Teile von Zeilen (z.B. von einem **d3w** Kommando), wird der Zeichenmodus verwendet. Die ‘ex’-Kommandos schneiden immer ganze Zeilen aus. Im Zeichenmodus wird der Text in die Zeile eingefügt, in der der Cursor steht. Im Zeilenmodus wird der Text in eine neue Zeile oberhalb oder unterhalb der aktuellen Cursorposition eingefügt. Im ‘visual mode’ fügen die Kommandos **p** und **P** Text aus einem Puffer ein. Standardmäßig wird der Puffer 1 verwendet. Soll ein anderer Puffer Verwendung finden, so muß sein Name vor dem Kommando angegeben werden (z.B. “**ap** fügt den Puffer *a* vor dem Cursor ein). Das **p** Kommando fügt den Pufferinhalt vor dem Cursor ein, das **P** Kommando dahinter. Im ‘ex mode’ wird der Text nach einer angegebenen Zeile eingefügt. Soll ein anderer als der Puffer 1 verwendet werden, muß sein Name nach dem Kommando stehen.

Makros

Der Inhalt eines benannten Puffers kann als Kommandofolge ausgeführt werden. Um die Kommandos in den Puffer zu bringen, werden sie zuerst in die Datei geschrieben und anschließend in einen benannten Puffer

gelöscht. Um den Inhalt eines Puffers als ‘ex’ Befehle auszuführen wird das ‘ex’ Kommando **@** verwendet (:@z führt den Inhalt des Puffers z aus). Soll der Pufferinhalt als ‘vi’ Kommandos ausgeführt werden, wird das ‘visual mode’ Kommando **@** verwendet. Die beiden **@** Kommandos unterscheiden sich dadurch, daß beim ‘ex’-Kommando der Puffer **nach** dem Befehl angegeben wird, und der Inhalt als ‘ex’ Befehle interpretiert wird. Im ‘vi’ Kommando wird der Puffer **vor** dem Kommando angegeben, und der Pufferinhalt wird als ‘vi’-Befehle interpretiert.

Bei der Verwendung des ‘vi’-Kommandos **@** ist zu beachten, daß der Pufferinhalt Zeichen für Zeichen interpretiert wird. Jedes Zeichen wird als Tastendruck betrachtet. Ist der Pufferinhalt z.B. mit “zdd ausgeschnitten worden, so wird das Zeilenvorschubzeichen am Ende der Zeile ebenfalls als Tastendruck ausgeführt, und der Cursor eine Zeile abwärts bewegt. Um dieses Verhalten zu verhindern, sollte der Puffer mit 0“zD ausgeschnitten werden.

Obwohl in Zwischenspeichern nahezu jede beliebige Textmenge gespeichert werden kann, kann **elvis** nur kleine Puffer als Makros ausführen. Ist ein Puffer zu groß um ausgeführt werden zu können, gibt **elvis** eine Fehlermeldung aus. Die maximale Makrogröße liegt bei ca. 1000 Zeichen. Desweiteren ist es nicht möglich :@ Befehle zu verschachteln, sie aus der Datei .exrc heraus auszuführen, oder aus einem :@ Kommando heraus einen :source Befehl aufzurufen.

Wechseln der Arbeitsdatei

Nach dem Starten von **elvis** sind alle Puffer leer. Wird (z.B. mit einem :n Befehl) die Datei gewechselt, so werden die 9 anonymen Puffer gelöscht. Der Inhalt der Puffer a—z bleibt jedoch erhalten.

Autor:

Steve Kirkendall

1.24 env

Funktion:

env (environment) führt einen Befehl in veränderter Umgebung aus

Syntax:

env [-] [-i] [-u *Name*] [--ignore-environment] [--unset=*Name*] [*Name*=*Wert*]. . . [*Befehl* [*Argumente*. . .]]

Beschreibung:

Mit **env** lassen sich einzelne Unix-Kommandos in veränderter Umgebung ausführen, ohne das Environment der Shell zu verändern. Mit *Name*=*Wert* lassen sich beliebige Umgebungsvariable ändern bzw. der Umgebung hinzufügen. Der Wert kann leer sein. Eine leere Umgebungsvariable ist verschieden von einer mit *unset* gelöschten.

Achtung! Die Umgebungsvariablen PATH, IFS, PPID, PS1, PS2, UID und EUID können nicht aus der Umgebung entfernt werden (siehe auch beim Shellkommando **unset** auf Seite 43). Wenn diese Variablen geleert werden, erscheinen in der neuen Prozessumgebung automatisch die voreingestellten Werte.

Optionen:

-u *Name* (unset) entfernt die Umgebungsvariable *Name* aus dem Environment

‘-’ oder -i (ignore) löscht alle Umgebungsvariablen

Autor:

Richard Mlynarik und David MacKenzie

eval

eval ist eine eingebaute Shellfunktion. Eine Beschreibung ist im Abschnitt über die **bash** auf Seite 34 zu finden.

exec

exec ist eine eingebaute Shellfunktion. Eine Beschreibung ist im Abschnitt über die **bash** auf Seite 34 zu finden.

exit

exit ist eine eingebaute Shellfunktion. Eine Beschreibung ist im Abschnitt über die **bash** auf Seite 35 zu finden.

1.25 expand

Funktion:

expand ersetzt Tabulatorzeichen durch Folgen von Leerzeichen

Syntax:

expand [- *Tab1* [, *Tab2* [, ...]]] [-t *Tab1* [, *Tab2* [, ...]]] [-i] [--tabs=*Tab1* [, *Tab2* [, ...]]] [--initial] [*Datei* ...]

Beschreibung:

expand liest eine Textdatei und ersetzt alle Tabulatoren durch entsprechende Folgen von Leerzeichen. In der Voreinstellung werden alle Tabulatoren ersetzt und eine Tabellenbreite von 8 Zeichen angenommen. Backspace Zeichen werden unverändert ausgegeben.

Wenn anstelle eines Dateinamens ein ‘-’, oder gar kein Dateiname angegeben ist, wird von der Standardeingabe gelesen.

Optionen:

- **Tab1** setzt die Tabellenbreite für alle Spalten auf *Tab1* anstelle von 8

- **Tab1**, **Tab2**, ... setzt die erste Tabellenspalte auf *Tab1*, die Zweite auf *Tab2* und so weiter; wenn im Text mehr Tabulatoren auftauchen als bei der Option angegeben, werden alle folgenden Tabulatoren durch einzelne Leerzeichen ersetzt

-t **Tab1** ... macht nichts anderes als die oben beschriebenen Optionen

-i konvertiert nur die führenden Tabulatoren jeder Zeile in Leerzeichen

Autor:

David MacKenzie

export

export ist eine eingebaute Shellfunktion. Eine Beschreibung ist im Abschnitt über die **bash** auf Seite 35 zu finden.

1.26 expr**Funktion:**

expr (expression) bearbeitet einen Ausdruck

Syntax:

expr *Ausdruck* ...

Beschreibung:

expr bewertet oder berechnet einen oder mehr *Ausdrücke* und gibt das Ergebnis auf die Standardausgabe. Ein Ausdruck besteht aus Zahlen oder Zeichenketten, die durch Operatoren verbunden sind. Eine Zeichenkette braucht nicht in Anführungszeichen eingeschlossen werden. Ob eine Ziffernfolge als Zeichenkette oder als Zahl bearbeitet wird, hängt vom Operator und der Position der Ziffernfolge im Ausdruck ab.

Folgende Operatoren werden erkannt:

| *Ausdruck1*|*Ausdruck2* liefert *Ausdruck1* wenn dieser nicht leer oder gleich 0 ist. Anderenfalls wird *Ausdruck2* ausgegeben.

& *Ausdruck1*&*Ausdruck2* ist gleich 0 wenn einer der beiden Ausdrücke leer oder 0 ist. Sonst wird *Ausdruck1* ausgegeben.

<, <=, =, !=, >=, > Vergleicht zwei Ausdrücke und liefert 1 wenn die Relation stimmt, anderenfalls 0. Es wird zuerst versucht beide Ausdrücke numerisch zu vergleichen. Wenn mindestens einer der Ausdrücke keinen numerischen Wert hat, werden die Ausdrücke lexikografisch verglichen.

+ − */% verknüpft die Ausdrücke arithmetisch. Wenn einer der Ausdrücke keinen numerischen Wert hat wird eine Fehlermeldung ausgegeben. Der % Operator liefert den Rest bei ganzzahliger Division (Modulo)

: *Ausdruck1*:*Ausdruck2* wendet den regulären *Ausdruck2* auf die Zeichenkette *Ausdruck1* an und liefert die Anzahl der passenden Zeichen oder den auf den von '(' und ')' eingeschlossenen Teil von *Ausdruck2* passenden Teil von *Ausdruck1* zurück. (alles klar?) Wenn der *Ausdruck2* auf *Ausdruck1* nicht paßt, liefert der Operator 0.

Der Status von **expr** ist

0 wenn der gesamte bewertete Ausdruck weder leer noch 0 ist

1 wenn der gesamte bewertete Ausdruck leer oder 0 ist

2 wenn ein Fehler aufgetreten ist

Autor:

Mike Parker

1.27 **fdformat**

Funktion:

fdformat formatiert eine Diskette (low-level)

Syntax:

fdformat [*-n*] *Gerätedatei*

Beschreibung:

fdformat formatiert eine Diskette. Der Diskettentyp, die Kapazität und das Laufwerk werden durch die entsprechende Gerätedatei ausgewählt. In der folgenden Liste sind die Gerätedateien mit den dazugehörigen Diskettenformaten aufgeführt. Die Zahlen in Klammern sind die Major/Minor Gerätenummern.

/dev/fd0D360 (2,20) Laufwerk A, 3.5 Zoll 369 KB
/dev/fd0H1440 (2,28) Laufwerk A, 3.5 Zoll 1440 KB
/dev/fd0Q720 (2,16) Laufwerk A, 3.5 Zoll 720 KB
/dev/fd0d360 (2,20) Laufwerk A, 5 1/4 Zoll 360 KB
/dev/fd0h1200 (2,8) Laufwerk A, 5 1/4 Zoll 1200 KB
/dev/fd0q720 (2,24) Laufwerk A, 5 1/4 Zoll 720 KB
/dev/fd1D360 (2,21) Laufwerk B, 3.5 Zoll 360 KB
/dev/fd1H1440 (2,29) Laufwerk B, 3.5 Zoll 1440 KB
/dev/fd1Q720 (2,17) Laufwerk B, 3.5 Zoll 720 KB
/dev/fd1d360 (2,21) Laufwerk B, 5 1/4 Zoll 360 KB
/dev/fd1h1200 (2,9) Laufwerk B, 5 1/4 Zoll 1200 KB
/dev/fd1q720 (2,25) Laufwerk B, 5 1/4 Zoll 720 KB

Auf der Diskette wird kein Dateisystem eingerichtet. Dazu stehen die Kommandos **mkfs** und **mkefs** für Linux Dateisysteme und **mformat** für MS-DOS Dateisysteme zur Verfügung. Die roh formatierte Diskette kann aber auch von **tar** direkt beschrieben werden.

Optionen:

-n unterdrückt die Verifizierung der formatierten Diskette

Autor:

Werner Almesberger

fgrep

fgrep sucht einen Ausdruck in einer (oder mehr) Datei(en). Die Funktionalität und die Optionen entsprechen **egrep**, allerdings ist diese Funktion für feste Ausdrücke ohne Sonderzeichen optimiert, und bringt in diesem Fall einen Geschwindigkeitsvorteil.

Siehe auch bei **egrep**, Seite 58 und **grep** Seite 91.

1.28 file

Funktion:

file bestimmt den Dateityp

Syntax:

file [-c] [-f *Namendatei*] [-m *Magiedatei*] *Datei* ...

Beschreibung:

file versucht die Art oder den Typ der angegebenen *Datei* zu bestimmen. Dazu werden drei Tests durchgeführt: Ein Dateisystemtest, ein Kennzahlentest und ein Sprachtest. Der erste erfolgreiche Test führt zur Ausgabe des erkannten Dateityps.

Der erkannte Typ enthält normalerweise eines der Schlüsselworte ‘text’ für Dateien die ohne Schwierigkeiten angezeigt werden können, ‘executable’ für Dateien, die ausführbare Programme enthalten und auf dem einen oder anderen Unix Rechner auch ausgeführt werden können, und ‘data’ für alle anderen Dateien, die normalerweise nicht angezeigt werden können. Nur allgemein bekannte Dateiformate wie ‘core’ Dateien oder tar Archive werden ohne diese Schlüsselworte benannt.

Der Dateisystemtest wird mit Hilfe des ‘stat’ Systemaufrufs durchgeführt. Hier werden leere Dateien ebenso erkannt, wie alle Gerätedateien, Sockets, symbolische Links und andere Spezialdateien.

Der Kennzahlentest kann einige Dateien anhand festgelegter Kennzahlen – sogenannter ‘magic numbers’ – erkennen, die sich in der Nähe des Dateianfangs an einer festgelegten Stelle befinden. Mit Hilfe solcher Kennzahlen entscheidet beispielsweise das Betriebssystem, ob eine Datei korrekt ausführbar ist, oder nicht. Diese Kennzahlen sind in der Datei /etc/magic abgespeichert.

Wenn eine Datei als Text erkannt ist, versucht **file** noch die (Programmier-)Sprache zu erkennen, indem es nach bestimmten Schlüsselworten sucht. Auf diese Weise kann beispielsweise C-Quelltext oder die Eingabe für den groff Textprozessor erkannt werden.

Optionen:

-m *Magiedatei* benutzt die benannte *Magiedatei* anstelle von /etc/magic für den Kennzahlentest

-c gibt den interpretierten Inhalt der Kennzahlendatei für Testzwecke aus

-f *Namendatei* veranlaßt **file** die Namen der zu untersuchenden Programme aus der *Namendatei* zu lesen; in der *Namendatei* werden die Dateinamen durch Zeilenende getrennt aufgeführt

Dateien:

/etc/magic

Autor:

Ian F. Darwin

1.29 find

Funktion:

find sucht nach bestimmten Dateien

Syntax:

find [*Verzeichnis*] [*−Option ...*] [*−Test ...*] [*−Aktion ...*]

Beschreibung:

find durchsucht eine oder mehrere Verzeichnishierarchien nach Dateien mit bestimmten Eigenschaften, und führt damit bestimmte Aktionen aus. Die Eigenschaften können durch Tests bestimmt werden.

Optionen, Tests und Aktionen können mit Operatoren zusammengefasst werden. **find** bewertet für jede Datei in den Verzeichnishierarchien die Optionen, Tests und Aktionen von links nach rechts, bis ein falscher Wahrheitswert auftaucht, oder die Kommandozeilenargumente zuende sind.

Das erste Argument, das mit einem ‘−’, einer Klammer ‘(, ‘)’, einem Komma ‘,’ oder einem Ausrufezeichen ‘!’ beginnt, wird als Anfang einer Option oder Test interpretiert. Alle Argumente davor werden als Verzeichnisnamen interpretiert.

Wenn kein Verzeichnis angegeben ist, wird das aktuelle Verzeichnis genommen. Wenn keine Aktion angegeben ist, wird die Aktion ‘−print’ ausgeführt.

Der Status von **find** ist Null, wenn alle Aktionen erfolgreich waren, im Fehlerfall ist der Status größer als Null.

Optionen:

Die Optionen bestimmen das allgemeine Verhalten des Kommandos, und beziehen sich nicht auf spezielle Dateien. Die Optionen sind immer wahr.

- −**daystart** mißt die Zeiten für die −amin, −atime, −cmin, −ctime, −mmin und −mtime Eigenschaften vom Beginn des aktuellen Tages anstelle der letzten 24 Stunden
- −**depth** bearbeitet den Inhalt jedes Verzeichnisses vor dem Verzeichnis selbst
- −**follow** folgt den symbolischen Links; diese Option schließt ‘−noleaf’ mit ein
- −**maxdepth Ebenen** steigt bis zu der gegebenen Zahl von Ebenen im Verzeichnisbaum auf (in der Hierarchie ab); bei 0 Ebenen werden die Tests nur auf die in der Kommandozeile übergebenen Dateien und Verzeichnisnamen angewendet
- −**mindepth Ebenen** steigt mindestens die gegebene Zahl von Ebenen im Verzeichnisbaum auf (in der Hierarchie ab); bei einer Ebene werden die in der Kommandozeile genannten Dateien und Verzeichnisnamen nicht bearbeitet
- −**noleaf** erzwingt die Bearbeitung aller Verzeichniseinträge; normalerweise kann davon ausgegangen werden, daß jedes Linuxverzeichnis wenigstens zwei (harte) Links enthält: das Verzeichnis ‘.’ ist ein Link auf das Verzeichnis selbst, und jedes Unterverzeichnis enthält den Eintrag ‘..’ als Link auf das Oberverzeichnis; wenn **find** bei der Untersuchung eines Verzeichnisses zwei Unterverzeichnisse weniger untersucht hat, als das Verzeichnis Links zählt, kann deshalb normalerweise die weitere Suche beendet werden
- −**version** gibt die Versionsnummer auf die Standardfehlerausgabe
- −**xdev** durchsucht keine Verzeichnisse in anderen Dateisystemem (auf anderen Partitionen)

Tests:

Alle numerischen Argumente können auf drei Arten angegeben werden:

- +**N** steht für alle Zahlen größer als *N*
- −**N** steht für alle Zahlen kleiner als *N*

N steht für genau *N*

Alle Tests werden auf die Dateien in den angegebenen Verzeichnissen einzeln angewendet. Die Tests liefern einen Wahrheitswert von 0 (Wahr) wenn der Test erfolgreich war.

- amin *N*** auf die Datei ist vor *N* Minuten zugegriffen worden
- anewer *Referenzdatei*** auf die Datei ist vor weniger Zeit zugegriffen worden, als seit der letzten Veränderung der *Referenzdatei* vergangen ist; im Zusammenhang mit ‘**-follow**’ tritt ‘**-anewer**’ nur in Effekt, wenn ‘**-follow**’ vor ‘**-anewer**’ in der Kommandozeile steht
- atime *N*** auf die Datei ist vor *N**24 Stunden zugegriffen worden
- cmin *N*** der Status der Datei wurde vor *N* Minuten geändert
- cnewer *Referenzdatei*** der Status der Datei wurde vor weniger Zeit verändert, als seit der letzten Veränderung der *Referenzdatei* vergangen ist; zusammen mit ‘**-follow**’ tritt ‘**-cnewer**’ nur in Effekt, wenn ‘**-follow**’ vor ‘**-cnewer**’ in der Kommandozeile steht
- ctime *N*** der Dateistatus wurde vor *N**24 Stunden geändert
- empty** die reguläre Datei oder das Verzeichnis ist leer
- false** ist immer falsch
- fstype *Typ*** die Datei ist in einem Dateisystem vom angegebenen *Typ*; unter anderem werden *minix*, *msdos*, *ext* und *proc* erkannt
- gid *N*** die Datei gehört der Gruppe mit der Kennzahl *N*
- group *Name*** die Datei gehört der Gruppe ‘*Name*’
- inum *N*** die Datei belegt die Inode mit der Nummer *N*
- links *N*** die Datei hat *N* (harte) Links
- lname *Muster*** die Datei ist ein symbolischer Link auf eine Datei oder ein Verzeichnis mit einem zum *Muster* passenden Namen
- mmin *N*** der Inhalt der Datei wurde vor *N* Minuten verändert
- mtime *N*** der Inhalt der Datei wurde vor *N**24 Stunden verändert
- name *Muster*** der Name der Datei paßt zu dem *Muster*
- newer *Referenzdatei*** die Datei ist später verändert worden als die *Referenzdatei*; zusammen mit ‘**-follow**’ tritt ‘**-newer**’ nur in Effekt, wenn ‘**-follow**’ vor ‘**-newer**’ in der Kommandozeile steht
- nouser** die Datei gehört keinem im System eingetragenen Benutzer
- nogroup** die Datei gehört keiner im System angemeldeten Gruppe
- path *Muster*** der Pfadname der Datei paßt zum *Muster*
- perm *Modus*** die Zugriffsrechte auf die Datei entsprechen exakt dem *Modus*; der *Modus* kann als Oktalzahl oder mit den bei **chmod** auf Seite 45 beschriebenen Kennungen beschrieben werden, die Kennungen werden auf *Modus* ‘000’ bezogen
- perm **-***Modus*** (mindestens) die Zugriffsrechte für den *Modus* sind gesetzt
- perm **+***Modus*** die Zugriffsrechte entsprechen höchstens dem *Modus* (oder sind weiter eingeschränkt)
- regex *Muster*** der Pfadname paßt zu dem regulären Ausdruck *Muster*

size *N* [{**c**,**k**}] die Datei belegt *N* Datenblöcke zu 512 Bytes, bzw. *N* Bytes und *N* Kilobytes mit nachgestelltem '**c**' oder '**k**'

-true ist immer wahr

-type *C* die Datei ist vom Typ *C*; folgende Typen werden unterschieden:

- b** gepufferte Gerätedatei für ein blockorientiertes Gerät
- c** ungepufferte Gerätedatei für ein zeichenorientiertes Gerät
- d** Verzeichnis
- p** benannte Pipeline (FiFo)
- f** normale Datei
- l** symbolischer Link
- s** Socket

-uid *N* die Kennziffer des Eigentümers ist *N*

-used *N* auf die Datei ist *N* Tage nach der letzten Änderung zugegriffen worden

-user *Name* die Datei gehört dem Anwender '*Name*'

-xtype *C* das Gleiche wie '**-type**' für alle Dateien, die keine symbolischen Links sind; wenn die Datei ein symbolischer Link ist und die Option '**-follow**' nicht gesetzt ist, wird die Datei auf die der Link zeigt auf den Typ *C* geprüft; wenn die Option '**-follow**' gesetzt ist, ist der Test wahr, wenn *C* = '**l**' ist (**xtype** untersucht bei symbolischen Links immer die Datei, die von **type** nicht untersucht wird)

Aktionen:

-exec *Kommando* ; führt das *Kommando* aus; die Aktion ist wahr, wenn das Kommando einen Status von Null liefert; alle auf den Kommandonamen folgenden Argumente bis zu einem Semikolon ';' werden als Kommandozeilenargumente für das Kommando interpretiert; die Konstruktion '{ }' wird durch den Pfadnamen der Datei ersetzt; die Klammern und das Semikolon müssen in der Kommandozeile für **find** quotiert werden, damit sie nicht von der Shell bearbeitet werden (siehe auch auf Seite 14)

-fprint *Ausgabedatei* schreibt den Pfadnamen der getesteten Datei in die *Ausgabedatei*; wenn die Ausgabedatei nicht existiert, wird sie erzeugt, sonst wird sie erweitert; die Standardausgabe und die Standardfehlerausgabe werden als '/dev/stdout' und '/dev/stderr' angesprochen

-fprint0 *Ausgabedatei* schreibt den Namen der getesteten Datei in die *Ausgabedatei* und schließt die Ausgabe mit einem Nullbyte ab, wie '**-print0**'

-fprintf *Ausgabedatei* *Format* schreibt den Namen der getesteten Datei in die *Ausgabedatei*, und benutzt dabei das *Format* mit Sonderzeichen wie bei '**-printf**'

-ok *Kommando* ; wie '**-exec**', vor der Ausführung des Kommandos wird aber noch eine Bestätigung erwartet; nur eine Eingabe, die mit einem 'Y' oder einem 'y' beginnt, führt zur Ausführung des Kommandos

-print gibt den vollständigen Pfadnamen der getesteten Datei auf die Standardausgabe

-print0 gibt den Pfadnamen der getesteten Datei auf die Standardausgabe, und schließt ihn mit einem Nullbyte ab; auf diese Weise können auch Pfadnamen korrekt weiterverarbeitet werden, die ein Zeilenende enthalten

-printf *Format* gibt für die getestete Datei die Zeichenkette *Format* auf der Standardausgabe aus; *Format* kann verschiedene Sonderzeichen und Platzhalter enthalten, die von **find** bearbeitet werden:

\a Alarmton

\b Rückschritt
\c Abbruch der Ausgabe
\f Seitenvorschub
\n Zeilenvorschub
\r Wagenrücklauf
\t horizontaler Tabulator
\v vertikaler Tabulator
**** der Backslash selbst

ein Backspace gefolgt von irgendeinem anderen Zeichen wird als normales Zeichen interpretiert, und einfach ausgegeben

%% das Prozentzeichen selbst

%a die Zeit des letzten Zugriffs auf die Datei, in dem Format der C `ctime` Funktion

%Ak die Zeit des letzten Zugriffs auf die Datei, in dem von *k* bestimmte Format; *k* hat dabei das gleiche Format, wie der entsprechende Parameter der `strftime` Funktion in C:

@ Sekunden seit dem 1.1.1970 0 Uhr GMT
H Stunde (00 bis 23)
I Stunde (01 bis 12)
k Stunde (0 bis 23)
l Stunde (1 bis 12)
M Minute (00 bis 59)
p PM oder AM
r Zeit, 12 Stunden (hh:mm:ss: AM/PM)
S Sekunden (00 bis 61)
T Zeit, 24 Stunden (hh:mm:ss)
X Zeit (H:M:S)
Z Zeitzone, oder nichts
a abgekürzter Wochentag
A ausgeschriebener Wochentag
b abgekürzter Monatsname
B ausgeschriebener Monatsname
c Datum und Zeit
d Tag im Monat
D Datum (mm/dd/yy)
h das Gleiche wie 'b'
j der Tag im Jahr
m die Zahl des Monats
U die Nummer der Woche, Sonntag als erster Wochentag
w die Zahl des Wochentags
W die Nummer der Woche, Montag als erster Wochentag
x Datum (mm/dd/yy)
y die letzten beiden Stellen der Jahreszahl
Y die Jahreszahl

%b die Dateigröße in 512 Byte Blöcken (aufgerundet)

%c das Datum der letzten Statusänderung im Format der C `ctime` Funktion

%Ck das Datum der letzten Statusänderung im Format der `strftime` Funktion; Parameter wie oben

%d die Höhe der Datei im Verzeichnisbaum; Null bedeutet, daß die Datei Kommandozeilenargument ist

%f der Name der getesteten Datei, ohne Verzeichnisse

%g der Gruppenname der getesteten Datei, oder die Kennzahl, wenn die Gruppe nicht eingetragen ist

%G die Gruppenkennzahl

%h die Verzeichnisnamen des Pfadnamen der getesteten Datei

%H das Kommandozeilenargument (Test) mit dem die Datei gefunden wurde

%i die Nummer der Inode der getesteten Datei

%k die aufgerundete Größe der getesteten Datei in Kilobytes

%l das Objekt auf die ein symbolischer Link zeigt; leer, wenn die getestete Datei kein symbolischer Link ist

%m die Zugriffsrechte als Oktalzahl

%n die Anzahl der harten Links auf die getestete Datei

%p der Pfadname der Datei

%P der Pfadname und das Kommandozeilenargument (Test) mit dem die Datei gefunden wurde

%s die Größe der getesteten Datei in Bytes

%t die Zeit der letzten Änderung im `ctime` Format

%Tk die Zeit der letzten Änderung, im `strftime` Format (siehe oben)

%u der Name des Eigentümers der getesteten Datei, oder die Kennzahl, wenn der Benutzer nicht eingetragen ist

%U die Benutzerkennzahl des Eigentümers der getesteten Datei

-prune wahr, wenn die Option `'-depth'` gesetzt ist ; sonst falsch

-ls zeigt das Verzeichnis in dem die getestete Datei gefunden wurde mit `'ls -dils'` an

Operatoren:

Die Optionen, Tests und Aktionen können mit Operatoren verknüpft werden. Die Bearbeitung erfolgt prinzipiell von links nach rechts.

(*Ausdruck*) die Klammern fassen den *Ausdruck* zu einer Operation zusammen

! *Ausdruck* ist wahr, wenn der *Ausdruck* falsch ist

-not *Ausdruck* ist ebenfalls wahr, wenn der *Ausdruck* falsch ist

***Ausdruck1* *Ausdruck2* UND** Verknüpfung; *Ausdruck2* wird bewertet (ausgefuehrt), wenn *Ausdruck1* wahr ist

Ausdruck1* -a *Ausdruck2 auch eine UND Verknüpfung

Ausdruck1* -and *Ausdruck2 auch eine UND Verknüpfung

***Ausdruck1* -o *Ausdruck2* ODER** Verknüpfung; *Ausdruck2* wird bewertet (ausgefuehrt), wenn *Ausdruck1* falsch ist

Ausdruck1* -or *Ausdruck2 auch eine ODER Verknüpfung

Ausdruck1* , *Ausdruck2 Liste; beide Ausdrücke werden immer bewertet (ausgefuehrt); der Wahrheitswert des gesamten Ausdrucks entspricht dem von *Ausdruck2*

Autor:

Eric Decker, David MacKenzie, Jay Plett und Tim Wood

1.30 fold**Funktion:**

fold bricht Zeilen ab einer bestimmten Länge um

Syntax:

fold [*-bs*] [*-w Länge*] [*--bytes*] [*--spaces*] [*--width=Länge*] [*Datei ...*]

Beschreibung:

fold liest die Datei(en) oder die Standardeingabe wenn keine Datei oder anstelle einer Datei ‘-’ angegeben ist, und schreibt den Inhalt auf die Standardausgabe. Dabei werden Zeilen ab einer bestimmten Länge umgebrochen. Voreingestellte Länge ist 80 Zeichen pro Zeile. Ohne weitere Optionen wird einfach nach dem achtzigsten Zeichen ein Zeilenvorschub eingefügt.

fold zählt nicht die Zeichen, sondern die Bildschirmspalten, sodaß Tabulatoren korrekt zählen. Ein Backspace erniedrigt die Zeichenzahl entsprechend und ein Wagenrücklauf (carriage return) setzt die Zahl auf Null zurück. ein Zeichen

Optionen:

- b** es werden nicht die Bildschirmspalten, sondern die Zeichen gezählt; die oben genannten Sonderzeichen erhöhen die Zeichenzahl jeweils um Eins, wie alle anderen Zeichen auch
- s** der Zeilenumbruch findet bei dem letzten Leerzeichen der Zeile statt, wenn in der Zeile ein Leerzeichen vorhanden ist
- w Länge** setzt die maximale Zeilenlänge auf den angegebenen Wert

Autor:

David MacKenzie

1.31 free**Funktion:**

free zeigt den freien Speicherplatz im Arbeitsspeicher und auf dem Swapdevice an

Syntax:

free [*-msibpc*] [*-d Blockdevice*] [*-S Intervall*] [*Swapdevice*]

Beschreibung:

Das **free** Kommando gehört zu der “**ps**-Suite”. Wie das **ps** Programm muß es bestimmte Daten direkt aus dem Kernelspeicher lesen. Dazu gibt es wieder die beiden Möglichkeiten, die bei **ps** auf Seite 127 beschrieben sind. Die **free** Version, die mit dem Prozessdateisystem arbeitet erkennt die hier aufgeführten Optionen nicht. Es zeigt nur den freien Speicherplatz im Arbeitsspeicher und auf dem Swapgerät, wie ‘**free -ms**’.

Optionen:

- m** zeigt nur den freien Speicherplatz im Arbeitsspeicher
- s** zeigt nur den freien Speicherplatz im Swapdevice
- i** zeigt die Auslastung der Inodes im Blockdepot
- b** zeigt die Auslastung der Datenblöcke im Blockdepot
- d** *Gerät* zeigt nur die Daten für die gepufferten Blöcke vom Gerät
- f** zeigt die Auslastung der Dateikennzeichner (Filedescriptoren)
- r** zeigt die Daten einer aktuellen Anforderung von Blöcken aus dem Blockdepot
- p** zeigt die Anzahl der Seiten anstelle von Kilobytes (bei den Optionen ‘-m’ und ‘-s’)
- S** *Sekunden* wiederholt die Anzeige nach der angegebenen Zahl Sekunden

Autor:

Branco Lancaster

1.32 grep

Funktion:

grep gibt alle Zeilen aus in denen ein bestimmter Ausdruck gefunden wird

Syntax:

grep [-CVbchilnsvw] [-Anzahl] [-AB Anzahl] [[-e] *Ausdruck* | -f *Datei*][*Datei* ...]

Beschreibung:

grep durchsucht die angegebenen *Dateien* (oder die Standardeingabe) nach einem *Ausdruck* und gibt die entsprechenden Zeilen aus. Der Status von **grep** ist 0 wenn der *Ausdruck* gefunden wurde und sonst 1.

Als *Ausdruck* akzeptiert **grep** reguläre Ausdrücke mit den folgenden Steuerzeichen:

- c** ein einzelner Buchstabe paßt auf sich selbst
- .** ein Punkt paßt auf jeden Buchstaben außer auf das Zeilenende
- \?** das dem Operator ‘\?’ vorangehende Muster kann null oder einmal vorkommen
- *** das dem Operator ‘*’ vorangehende Muster kann 0 mal oder öfter vorkommen
- \+** das dem Operator ‘\+’ vorangehende Muster kann 1 mal oder öfter vorkommen
- \|** die durch den Operator ‘\|’ verbindenden Argumente werden **oder** verknüpft
- ^** (Caret) paßt auf den Zeilenanfang
- \$** paßt auf das Zeilenende
- \<** paßt auf den Wortanfang
- \>** paßt auf das Wortende

[Buchstaben] paßt auf alle *Buchstaben*; dabei können einzelne Buchstaben, aber auch Bereiche in der Form ‘*von–bis*’ angegeben werden; wenn der erste Buchstabe nach ‘[’ ein ‘^’ ist, paßt der Ausdruck auf alle Buchstaben, außer den Aufgeführten

\(\) die Klammern fassen Ausdrücke zusammen; außerdem wird der auf den in Klammern eingeschlossene Teil des Musters passende Text markiert und mit einem folgenden ‘\N’ Ausdruck referenziert (Tag)

\N referenziert die auf das in der N-ten runden Klammern eingeschlossene Muster passende Zeichenkette. Beispielsweise kann mit dem Muster ‘\(_\.\._\)\)+\1’ nach Zeilen gesucht werden, in denen ein Wort mit vier Buchstaben doppelt, aber nicht unmittelbar nach dem ersten vorkommt.

\ jedes der Sonderzeichen kann durch ein ‘\’ (Backslash) eingeleitet sich selbst suchen

\b paßt auf kein Zeichen, sondern auf den Anfang oder das Ende eines Wortes

\B symbolisiert den Raum innerhalb eines Wortes

\w paßt auf alle alphanumerischen Zeichen [A-Za-z0-9]

\W paßt auf alle nichtalphanumerischen Zeichen [^A-Za-z0-9]

Die Rangfolge der Operatoren ist (von der höchsten zur niedrigsten): ‘(’, ‘)’, ‘?’, ‘*’, ‘+’ und ‘|’. Die anderen Operatoren sind mit den anderen Buchstaben gleichrangig.

Optionen:

- A** *Anzahl* gibt *Anzahl* Zeilen Kontext **nach** jeder gefundenen Zeile aus
- B** *Anzahl* gibt *Anzahl* Zeilen Kontext **vor** jeder gefundenen Zeile aus
- C** gibt 2 Zeilen Kontext **vor und nach** jeder gefundenen Zeile aus
- Anzahl* gibt *Anzahl* Zeilen Kontext **vor und nach** jeder gefundenen Zeile aus
- V** gibt die Versionsnummer auf die Standardfehlerausgabe
- b** gibt die Position jeder gefundenen Stelle mit aus
- c** gibt nur die Gesamtzahl der gefundenen Stellen aus
- e** *Ausdruck* sucht nach *Ausdruck*
- f** *Datei* *Datei* enthält die Ausdrücke, nach denen gesucht werden soll.
- h** unterdrückt die Dateinamen vor jeder Fundstelle
- i** ignoriert Groß und Kleinschreibung
- l** gibt nur die Dateinamen mit Fundstellen aus
- n** gibt die Zeilennummer zu jeder Fundstelle aus
- s** (silent) keine Ausgabe außer Fehlermeldungen.
- v** gibt nur Zeilen aus, die den *Ausdruck* nicht enthalten
- w** gibt nur Zeilen aus, in denen der *Ausdruck* als komplettes Wort vorkommt
- x** gibt nur Zeilen aus, die den *Ausdruck* als ganze Zeile enthalten

Siehe auch:

egrep, Seite 58 und fgrep, Seite 83

Autor:

Mike Haertel, James A. Woods und David Olson

1.33 groff

Funktion:

groff ist ein reiner Textfresser. Es frisst die weitverbreiteten ‘.1’, ‘.n’, ‘.an’ und ‘.ms’ Texte. Es lebt in symbiotischer Gemeinschaft mit **grog**, **gsoelim**, **grops**, **grotty** und einigen verwandten Programmen.

Syntax:

```
groff [ -tpeszaivhblCENRVZ ] [ -w Name ] [ -W Name ] [ -H Datei ] [ -m Makro ] [ -F Verzeichnis ] [ -T Format ]
[ -f Familie ] [ -M Verzeichnis ] [ -d cs ] [ -r cn ] [ -n Nummer ] [ -o Liste ] [ -P Argument ] [Datei ...]
```

Beschreibung:

groff ist der Kern eines Textformatiersystems. Normalerweise startet **groff** das **gtroff** Kommando und leitet die Ausgabe durch einen Postprozessor für ein bestimmtes Ausgabegerät. Folgende Ausgabeformate stehen zur Verfügung:

ps Postscript

dvi (DeVice Independent) das TeX Ausgabeformat

X75 für X-Windows Ausgabe mit 75dpi

X100 für X-Windows Ausgabe mit 100dpi

ascii für einfache Druckerausgabe

latin1 für Druckerausgabe mit ISO Latin-1 Zeichensatz

Das Standardformat ist **ascii**.

Im **groff** System stehen außerdem die Präprozessoren **gpic**, **geqn**, **gtbl**, **grefer** und **gsoelim** zur Verfügung.

Optionen:

-h gibt einen Hilfstext aus

-e leitet die Eingabe durch den **geqn** Präprozessor

-t leitet die Eingabe durch den **gtbl** Präprozessor

-p leitet die Eingabe durch den **gpic** Präprozessor

-s leitet die Eingabe durch den **gsoelim** Präprozessor

-R leitet die Eingabe durch den **grefer** Präprozessor. Die Übergabe von Kommandozeilenargumenten an **grefer** wird nicht unterstützt.

-v die von **groff** aufgerufenen Programme geben ihre Versionsnummer aus

-V gibt die von **groff** zusammengestellte Kommandozeile (Pipeline) auf der Standardausgabe aus anstatt sie auszuführen

-z unterdrückt die Ausgabe von **gtroff**; nur Fehlermeldungen werden ausgegeben.

- Z** unterdrückt den Postprozessor
- P** *Argument* gibt das *Argument* an den Postprozessor weiter. Jedes Argument sollte einzeln übergeben werden; den Argumenten wird kein ‘—’ vorangestellt
- L** *Argument* gibt das *Argument* an den Spooler weiter
- T** *Format* benutzt *AusgabeFormat*; Voreinstellung ist *ascii*
- N** übergibt die -N Option an *geqn*
- a** produziert reinen *ascii* Code, ohne Steuerzeichen
- b** gibt zusätzliche Information bei Fehlermeldungen
- i** liest aus der Standardeingabe nachdem alle Eingabedateien bearbeitet sind
- C** schaltet in den Kompatibilitätsmodus (zu *troff*)
- E** unterdrückt alle Fehlermeldungen
- w** *Name* erlaubt Warnung *Name*
- W** *Name* unterdrückt Warnung *Name*
- m** *Makro* die Datei ‘*tmac.Makro*’ wird gelesen und die darin definierten Makros zum Formatieren des Dokuments benutzt
- o** *Liste* gibt nur die Seiten aus *Liste* aus; die *Liste* ist eine durch Kommata getrennte Liste von Seitenbereichen.
- d** *cs* definiert das Register *c* mit *s*. Dabei ist *c* ein Buchstabe und *s* eine Zeichenkette.
- r** *cn* setzt Register *c* auf *n*. Dabei ist *c* ein Buchstabe und *n* ein numerischer Ausdruck.
- F** *Verzeichnis* sucht im *Verzeichnis* nach den Fonts
- M** *Verzeichnis* sucht in *Verzeichnis* nach den Makros
- H** *Datei* benutzt *Datei* nach der Trennmusterdatei
- f** *Familie* benutzt *Familie* als Fontfamilie
- n** *Nummer* setzt die Nummer der ersten Ausgabeseite auf *Nummer*

Umgebung:

Folgende Umgebungsvariablen werden unterstützt:

GROFF_TMAC_PATH eine durch Doppelpunkte getrennte Liste von Verzeichnissen, in denen nach Makrodateien gesucht wird

GROFF_TYPESETTER das Standardausgabeformat

GROFF_FONT_PATH eine durch Doppelpunkte getrennte Liste von Verzeichnissen, in denen nach den Gerätetreibern und den Zeichensätzen für das Ausgabeformat gesucht wird

GROFF_HYPHEN eine Datei mit Mustern für die automatische Trennung durch *groff*

GROFF_TMPDIR ein Verzeichnis, in dem die temporären Dateien von *groff* angelegt werden; wenn kein Verzeichnis angegeben ist, wird das Verzeichnis */tmp* benutzt

Dateien:

Die Folgenden Dateien werden vom **groff** System benutzt:

/usr/lib/groff/hyphen die Standardtrennmusterdatei

/usr/lib/groff/tmac/tmac.*Name* die Makrodatei für ‘*mName*’

/usr/lib/groff/font/dev*Name*/DESC der Gerätetreiber für das Gerät *Name*

/usr/lib/groff/font/dev*Name*/F die Fontdatei für Font ‘F’ von Gerät *Name*

/usr/lib/groff/font/dev*Name*/eqnchar die Definitionen von (g)eqn für das Gerät *Name*

Siehe auch:

gro(1), gtroff(1), gtbl(1), gpic(1), geqn(1), gsoelim(1), grefer(1), grops(1), grodvi(1), grotty(1), groff_font(5), groff_out(5), groff_msr(7), groff_me(7)

Autor:

James Clark

1.34 groups

Funktion:

groups zeigt alle Gruppen zu denen ein Benutzer gehört

Syntax:

groups [*Benutzer ...*]

Beschreibung:

groups gibt die Namen aller Gruppen vom *Benutzer* aus. Wird kein Name angegeben, werden die Gruppen des aktuellen Prozesses gezeigt.

Siehe auch:

groups ist ein Shellscript und benutzt id

1.35 gzip

Funktion:

gzip komprimiert Dateien

Syntax:

gzip [*-cdfhLrtvV19*] [*Datei ...*]

Beschreibung:

gzip komprimiert Dateien mit dem LZ77 Lempel-Ziv Algorithmus. **gzip** erzielt um einige Prozent bessere Kompressionsraten als das mit dem LZW Algorithmus arbeitende **compress** Programm. Weil es sich ansonsten sehr ähnlich verhält, ist abzusehen daß es **compress** als Standardpacker im Bereich der freien Software verdrängen wird. Mit **gzip** können auch Dateien ausgepackt werden, die mit **compress** gepackt wurden.

gzip ist kein Archivpacker, wie **lharc**, **arj** oder **pkzip**.

gzip komprimiert einzelne Dateien, unabhängig davon ob die resultierende Datei tatsächlich kleiner ist, und ersetzt die Urdatei durch die komprimierte indem es an den Dateinamen die Endung **‘.z’** anhängt. Dabei bleiben die Zugriffsrechte und das Erstellungsdatum der Urdatei erhalten.

Wenn der Dateiname durch Anhängen der Endung unzulässig lang würde, schneidet **gzip** automatisch die erforderliche Anzahl Zeichen vom Dateinamen ab, speichert aber den vollständigen Namen zur Restaurierung beim Auspacken in der Datei ab.

Wenn **gzip** ohne Dateinamen aufgerufen wird, liest es von der Standardeingabe und schreibt auf die Standardausgabe.

Wie bei **compress** kann auch **gzip** auf andere Namen gelinkt werden um bestimmte Aufgaben zu erfüllen.

Unter dem namen **gunzip** arbeitet es wie **gzip -d**, packt also komprimierte Dateien aus. Dabei kann es auch die von **compress** (ab Version 3.0) gepackten Dateien bearbeiten. **gunzip** erwartet die Endung **‘.z’** oder **‘.Z’** an dem Dateinamen. Nach dem Auspacken bleiben die Zugriffsrechte und das Erstellungsdatum der Datei erhalten.

zcat schreibt die entkomprimierte Datei auf die Standardausgabe und läßt die komprimierte Datei unberührt.

Optionen:

- c** schreibt die (ent)komprimierte *Datei* auf die Standardausgabe anstatt die Datei zu ersetzen
- d** dekomprimiert die *Datei*
- f** (force) ersetzt bestehende Dateien mit Endung **‘.z’**; normalerweise fragt **gzip** vor dem Überschreiben solcher Dateien nach
- h** gibt eine Kurzhilfe zum Programm aus
- L** gibt eine Kurzfassung des Lizenztextes aus
- r** (recursive) packt alle Dateien in den angegebenen Unterverzeichnissen
- t** (test) prüft die integrität der angegebenen *Datei*
- v** (verbose) gibt den Kompressionsfaktor für jede *Datei* aus
- V** (Version) gibt die Versionsnummer des Programms aus
- Ziffer** bestimmt mit einer Ziffer von 1 bis 9 die Kompressionstiefe; 1 bedeutet schnell und schlecht komprimiert, 9 bedeutet langsam und optimal komprimiert

Siehe auch:

compress auf Seite 49 und **tar** auf Seite 143

Autor:

Jean-Loup Gailly

1.36 head

Funktion:

head schreibt den Anfang einer Datei auf die Standardausgabe

Syntax:

head [-c *Anzahl*[bkm]] [-n *Anzahl*] [-qv] [--bytes=*Anzahl*[bkm]] [--lines=*Anzahl*] [-quiet] [--silent] [--verbose] [*Datei* ...]

head [-nrbcklmqv] [*Datei* ...]

Beschreibung:

head schreibt die ersten (10) Zeilen von der *Datei* auf den Bildschirm. Wenn keine *Datei* oder '-' angegeben wird, liest **head** von der Standardeingabe. Wird mehr als eine *Datei* angegeben, so wird der Dateiname in '==>' und '<==>' eingeschlossen der Ausgabe vorangestellt.

Optionen:

-c *Anzahl* gibt die angegebene *Anzahl* Bytes aus. Optional kann die Blockgröße durch einen der der Zahl folgenden Buchstaben verändert werden:

b Blocks zu 512 Byte

k Blocks zu 1 Kilobyte

m Blocks zu 1 Megabyte

-l *Anzahl* gibt die ersten *Anzahl* Zeilen aus

-q unterdrückt die Ausgabe des Dateinamen

-v gibt die Dateinamen immer aus

Autor:

David MacKenzie

1.37 hexdump

Funktion:

hexdump zeigt Dateien in hexadezimalen, dezimalen, oktalen oder ascii Zeichenformat an

Syntax:

hexdump [-bcdovx] [-e *Formatstring*] [-f *Formatdatei*] [-n *Anzahl*] [-s *Anzahl*] *Datei* ...

Beschreibung:

hexdump liest aus der Standardeingabe oder aus einer Datei und gibt die einzelnen Bytes formatiert und kodiert auf die Standardausgabe.

Optionen:

- b** die Ausgabe erfolgt in sechzehn Spalten zu je einem Byte in oktaler Darstellung mit hexadezimaler Dateiposition
- c** die Ausgabe erfolgt in sechzehn Spalten zu je einem Byte mit hexadezimaler Dateiposition. Druckbare Zeichen werden als solche angezeigt, nichtdruckbare Zeichen werden oktal dargestellt.
- d** die Ausgabe erfolgt in fünf Spalten zu je zwei Byte in dezimaler Darstellung mit hexadezimaler Dateiposition
- e** *Formatstring* legt das Ausgabeformat mit *Formatstring* fest
- f** *Formatdatei* liest die *Formatstrings* aus der *Formatdatei*. Jede Zeile enthält einen *Formatstring*, leere Zeilen und Zeilen deren erstes Zeichen ein '#' ist werden ignoriert.
- n** *Anzahl* gibt nur *Anzahl* Bytes aus
- o** die Ausgabe erfolgt in sechs Spalten zu je zwei Byte in oktaler Darstellung mit hexadezimaler Dateiposition
- s** *Anzahl* überspringt *Anzahl* Bytes am Anfang der Eingabe. *Anzahl* ist eine dezimale Zahl. Beginnt *Anzahl* mit 0x oder 0X wird sie hexadezimal interpretiert, beginnt sie mit 0 wird sie oktall interpretiert. Außerdem kann der Zahl einer der Buchstaben b, k oder m nachgestellt werden. Dadurch wird der Offset zu einem vielfachen von 512 (block) 1024 (kilo) oder 1048576 (mega).
- v** zeigt alle Eingabezeilen an. Normalerweise werden identische Zeilen durch ein einzelnes Asterisk (*) ersetzt.
- x** die Ausgabe erfolgt in acht Spalten zu je zwei Byte in hexadezimaler Darstellung mit hexadezimaler Dateiposition

Siehe auch:

od auf Seite 122

1.38 hostname

Funktion:

hostname zeigt oder setzt den Netzwerknamen des Systems

Syntax:

hostname [*Name*]

Beschreibung:

Wird **hostname** ohne ein Argument aufgerufen, gibt es den Netzwerknamen des Systems aus. Anderenfalls wird der Netzwerkname auf *Name* gesetzt. Das setzen ist nur dem Superuser (root) möglich.

Siehe auch:

uname auf Seite 147

Autor:

Peter Orbaek

1.39 *id*

Funktion:

id gibt die reale und die effektive User ID und Gruppen ID aus

Syntax:

id [-gnruG] [--group] [--name] [--real] [--user] [--groups] [*Username*]

Beschreibung:

Das **id** Kommando zeigt die reale Benutzerkennzahl (UID) mit dem Benutzernamen und alle Gruppen, in denen der Anwender eingetragen ist mit ihren Kennzahlen (GID) und Namen an. Wenn die effektive Benutzerkennung nicht der realen entspricht, wird die effektive Benutzerkennung ebenfalls angezeigt.

Optionen:

-g gibt nur die Gruppen ID aus

-n gibt den User- bzw. den Gruppennamen anstelle der ID (nur in Verbindung mit **-u**, **-g** oder **-G**)

-r gibt die reale anstelle der effektiven User- bzw. Gruppen ID aus (nur in Verbindung mit **-u**, **-g** oder **-G**)

-u gibt nur die User ID aus

-G gibt die ID's aller Gruppen von User aus

Autor:

Arnold Robbins und David MacKenzie

1.40 *install*

Funktion:

install kopiert Dateien, richtet Verzeichnisse ein und ändert die Zugriffsrechte

Syntax:

install [*Optionen*] [-s] [--strip] *Quelldatei* *Zieldatei*
install [*Optionen*] [-s] [--strip] *Quelldatei* ... *Verzeichnis*
install [*Optionen*] [-d,--directory] *Verzeichnis* ...

Beschreibung:

install wird normalerweise direkt vom **make** Utility aufgerufen, um ein neu übersetztes Programm mit allen Hilfs- und Konfigurationsdateien in die dafür vorgesehenen Verzeichnisse zu kopieren, und mit den nötigen Rechten auszustatten. Dabei können auch neue Verzeichnisse angelegt werden.

Optionen:

- c** ohne Funktion; zur Kompatibilität mit alten Unix Versionen von `install`
- d** erzeugt alle Verzeichnisse und die davor liegenden Verzeichnisse, wenn sie nicht existieren; wenn *Modus*, *Eigentümer* und *Gruppe* bestimmt sind, werden die entsprechenden Werte für alle Verzeichnisse benutzt, sonst gelten die Standardwerte (EUID, EGID und `umask`)
- g *Gruppe*** ändert die Gruppenkennung in den angegebenen Wert; es werden sowohl der Gruppenname, als auch die Gruppenkennzahl verarbeitet
- m *Modus*** setzt die Zugriffsrechte auf den angegebenen Modus; es werden die bei `chmod` (→ Seite 45) erklärten Oktalzahlen oder Buchstabencodes verarbeitet; die Buchstabencodes basieren auf dem Modus 000; der Standardmodus ist 0755
- o *Eigentümer*** ändert den Eigentümer der Datei; diese Option steht nur dem Superuser zur Verfügung; es werden sowohl Namen als auch Benutzerkennzahlen verarbeitet
- s** entfernt die Symboltabellen aus den Programmdateien

Autor:

David MacKenzie

1.41 join

Funktion:

join verknüpft zwei Dateien nach Schlüsselfeldern

Syntax:

```
join [-a 1|2] [-v 1|2] [-e Zeichenkette] [-o Feldliste ...] [-t Buchstabe] [-j1|2] Feldnr] [-1 Feldnr] [-2 Feldnr]
Datei1 Datei2
```

Beschreibung:

join verknüpft zwei (alphabetisch) sortierte Dateien, indem je zwei Zeilen mit identischen Schlüsselfeldern zu einer Ausgabezeile verbunden werden.

Die Schlüsselfelder sind durch Leerzeichen voneinander getrennt. Führende Leerzeichen werden ignoriert. Wenn nicht anders angegeben, ist das erste Feld einer jeden Zeile Schlüsselfeld. Die Ausgabefelder sind ebenfalls durch Leerzeichen voneinander getrennt. Die Ausgabe besteht aus dem Schlüsselfeld, gefolgt von den übrigen Feldern der Datei1 und schließlich aller Felder der passenden Zeilen von Datei2 ohne das Schlüsselfeld.

Optionen:

- a *Dateinummer*** fügt in die Ausgabe eine Leerzeile ein, wenn eine Zeile aus *Dateinummer* (1 oder 2) kein Gegenstück hat.
- e *Zeichenkette*** ersetzt fehlende Eingabefelder in der Ausgabe durch die *Zeichenkette*
- 1 *Feldnr*** benutzt in Datei1 *Feldnr* als Schlüsselfeld
- 2 *Feldnr*** benutzt in Datei2 *Feldnr* als Schlüsselfeld
- j *Feldnr*** benutzt *Feldnr* als Schlüsselfeld

-o *Feldliste* stellt die Ausgabezeilen anhand der *Feldliste* zusammen. Ein Eintrag in der *Feldliste* besteht aus einer *Dateinummer*, einem Punkt und einer *Feldnr*. Beliebig viele solcher Paare *Dateinummer.Feldnr* können durch Komma oder Leerzeichen getrennt in der *Feldliste* stehen.

-t *Buchstabe* verwendet *Buchstabe* als Feldtrenner

-v *Dateinummer* gibt nur die Zeilen aus *Dateinummer* aus, die kein Gegenstück haben.

Autor:

Mike Haertel

1.42 kill

Funktion:

kill beendet einen Prozess

Syntax:

kill [*-Signr*] *Prozeßnr*

Beschreibung:

kill wird benutzt, um außer Kontrolle geratene („aufgehängte“) Prozesse, die sich nicht mehr auf normale Art beenden lassen, zu terminieren (beenden). **kill** sendet dazu das Signal *Signr* an den Prozeß *Prozeßnr*. Standardwert ist SIGTERM (15) zum terminieren des Prozesses. Es können aber auch beliebige andere Signale gesendet werden. Weil das Signal SIGTERM nicht von allen Programmen bearbeitet wird, wird ein Prozeß manchmal erst mit dem Signal SIGKILL (9) vom Kernel beendet. Der „normalen“ Terminierung mit SIGTERM ist aber der Vorzug zu geben, weil dadurch dem Prozeß noch die Möglichkeit gegeben wird, die Bühne geordnet zu Verlassen. Es können nur die eigenen Prozesse beendet werden.

In der **bash** ist ein **kill** Kommando eingebaut, daß dieses externe Programm verdeckt, wenn nicht ausdrücklich mit dem **command** Shellkommando das externe Programm aufgerufen wird. (Siehe beim Shellkommando **kill**, Seite 36)

Optionen:

-Signr sendet *Signr* anstelle von SIGTERM (15)

Autor:

Peter MacDonald

1.43 less

Funktion:

less ist ein Pager wie **more**

Syntax:

less [*-[+]*aBcCdeEfHimMnNqQrsSuUw] [*-b Puffer*] [*-h Zeilen*] [*-j Zeile*] [*-k Schlüsseldatei*] [*-{oO} Datei*] [*-p Muster*] [*-P Prompt*] [*-t Tag*] [*-T Tagdatei*] [*-x Tab*] [*-y Zeilen*] [*-[z] Zeilen*] [*+Lesskommando*] [*Dateiname*] ...

Beschreibung:

less gibt eine (oder mehrere) Datei(en) Seitenweise auf die Standardausgabe. Im Unterschied zu **more** erlaubt **less** auch das Zurückblättern in Texten, die aus einer Pipeline oder der Standardeingabe gelesen wurden. Die voreingestellten Tastaturkommandos sind an die von **more** und **elvis** angelehnt; sie lassen sich aber vom Anwender mit Hilfe des **lesskey** Kommandos beliebig neu definieren.

Kommandos:

Die folgenden Tastaturkommandos zur Steuerung der Bildschirmausgabe sind definiert. Alle Kommandos können von einer ganzen Zahl *N* eingeleitet werden, die die Anzahl der Wiederholungen dieses Kommandos angibt.

h | **H** gibt einen Hilfstext aus

LEERZEICHEN | **CTRL-V** | **f** | **CTRL-F** blättert eine Bildschirmseite vorwärts; wenn eine Zahl vorangestellt ist, werden diese Anzahl Zeilen weitergeblättert

z wie Leerzeichen; wenn eine Zahl vorangestellt ist, wird diese Zahl zur neuen Seitenlänge auch für die weiteren Seiten

RETURN | **CTRL-N** | **CTRL-E** | **CTRL-J** scrollt den Bildschirm eine (oder die gegebene Anzahl) Zeilen weiter

d | **CTRL-D** blättert einen halben Bildschirm (oder die gegebene Anzahl Zeilen) weiter; wenn eine Zahl angegeben ist, wird sie zur Standardweite für alle folgenden **CTRL-D** und **CTRL-U** Kommandos

b | **CTRL-v** blättert einen Bildschirm oder die gegebene Anzahl Zeilen zurück

w wie **ESC-v**, die gegebene Zahl wird aber der neue Standardwert für das Zurückblättern

y | **CTRL-Y** | **CTRL-P** | **k** | **CTRL-K** scrollt den Bildschirm um eine (oder die gegebene Anzahl) Zeile(n) zurück

u | **CTRL-U** blättert einen halben Bildschirm (oder die gegebene Anzahl Zeilen) zurück

r | **CTRL-R** | **CTRL-L** schreibt den aktuellen Bildschirm neu

R schreibt den Bildschirm neu; dabei wird die gesamte gepufferte Eingabe verworfen

F scrollt den Bildschirm vorwärts; dabei wird versucht weiterzulesen, auch wenn das Dateiende bereits erreicht war

g | **<** | **CTRL-<** geht zur ersten Zeile oder der Zeile mit der entsprechenden Zahl

G | **>** | **CTRL->** geht zur letzten Zeile oder zu der Zeile mit der entsprechenden Zahl (vom Dateiende gezählt)

p | **%** macht nur Sinn, wenn es von einer Zahl zwischen 0 und 100 eingeleitet wird, und geht dann zu der dieser Prozentzahl entsprechenden Stelle; wenn von der Standardeingabe gelesen wird, kann dieses Kommando erst benutzt werden, wenn die Eingabe abgeschlossen ist

{ wenn eine geschweifte Klammer auf der ersten Bildschirmzeile steht, wird die Anzeige so gescrollt, daß die dazugehörige schließende Klammer auf der letzten Bildschirmzeile steht; wenn mehr als eine geschweifte Klammer auf der ersten Zeile geöffnet wird, kann mit dem Zahlenargument bestimmt werden, welche Klammer referenziert werden soll

} wenn die geschweifte Klammer auf der letzten Bildschirmseite steht, wird der Bildschirm so gerollt, daß die korrespondierende Klammer auf der ersten Zeile angezeigt wird

(das Gleiche mit runden Klammern

) dito

[diesmal mit eckigen Klammern

] auch nichts neues

ESC-CTRL-**F** wie {, es werden die dem Kommando folgenden beiden Zeichen als Klammern interpretiert und entsprechend bearbeitet

ESC-CTRL-**B** wie }, es werden die dem Kommando folgenden beiden Zeichen als Klammern interpretiert und entsprechend bearbeitet

m gefolgt von einem Kleinbuchstaben markiert die aktuelle Position mit diesem Buchstaben

' | CTRL-**X** CTRL-**X** gefolgt von einem Kleinbuchstaben kehrt zurück zu der mit diesem Buchstaben markierten Stelle

/Muster sucht vorwärts nach dem *Muster*; eine passende Zeile wird als erste Bildschirmzeile angezeigt; das *Muster* kann reguläre Ausdrücke enthalten (ed-Syntax); mit einem Zahlenargument N kann auch das N-te auftreten des *Musters* gesucht werden; die Suche beginnt in der zweiten Bildschirmzeile, wenn nicht durch die Kommandozeilenoptionen '-a' oder '-j' ein anderes Verhalten eingestellt ist; für das erste Zeichen vom *Muster* stehen folgende Sonderzeichen zur Verfügung:

! sucht nach einer Zeile, die das *Muster* nicht enthält

* durchsucht mehrere Dateien; wenn die Suche bis zum Dateiende erfolglos ist, wird die nächste Datei aus der Kommandozeilenliste durchsucht

@ beginnt die Suche in der ersten Datei aus der Kommandozeilenliste, unabhängig davon, welche Datei aktuell angezeigt wird; auch die Kommandozeilenoptionen '-a' und '-j' werden ignoriert

?Muster sucht rückwärts nach einer Zeile mit dem *Muster*; die Suche beginnt vor der ersten Bildschirmzeile; folgende Sonderzeichen können als erstes Zeichen vom *Muster* eingesetzt werden:

! sucht eine Zeile, die das *Muster* nicht enthält

* durchsucht mehrere Dateien; wenn das *Muster* bis zum Dateianfang erfolglos ist, wird die der aktuellen Datei vorhergehende aus der Kommandozeilenliste durchsucht

@ beginnt die Suche auf der letzten Zeile der letzten Datei aus der Kommandozeilenliste, unabhängig von der aktuellen Datei und den Kommandozeileoptionen '-a' und '-j'

ESC-**/Muster** das Gleiche wie '/**Muster*'

ESC-**?Muster** das Gleiche wie '?**Muster*'

n wiederholt die letzte Suche; die Sonderzeichen '*' und '!' behalten ihre Bedeutung, '@' wird ignoriert

N wiederholt die letzte Suche in umgekehrter Richtung

ESC-**n** wiederholt die letzte Suche und sucht in der nächsten (vorhergehenden) Datei weiter, wenn das Dateiende (der Dateianfang) erreicht ist

ESC-**N** wiederholt die Suche rückwärts, und setzt die Suche in weiteren Dateien fort

:**e** [*Dateiname* ...] zeigt die benannte Datei an; wenn keine Datei angegeben ist, wird die aktuelle Datei neu angezeigt; ein Prozentzeichen '%' im Dateiname wird durch den Namen der aktuellen Datei ersetzt; das Nummerzeichen '#' wird durch den Namen der zuvor angezeigten Datei ersetzt; die neue Datei wird in die Kommandozeilenliste der Dateien eingereiht, sodaß sie bei den folgenden 'sf :n' und 'p' Befehlen erreicht werden kann; wenn mehrere Dateinamen angegeben sind, werden alle Dateinamen in die Kommandozeilenliste eingefügt

CTRL-**X** CTRL-**V** | **E** das Gleiche wie 'e'

:**n** zeigt die nächste Datei aus der Kommandozeilenliste an

:p zeigt die vorhergehende Datei aus der Kommandozeilenliste an

:x zeigt die erste (oder N-te) Datei aus der Kommandozeilenliste an

= | CTRL-G | :f zeigt den Namen der aktuellen Datei und die Position in der Datei

– gefolgt von einer der unten aufgeführten Kommandozeilenoptionen ändert ebendiese Option und zeigt die neue Einstellung an; wenn zu einer Option ein Zahlenargument oder eine Zeichenkette angegeben werden müssen, kann dieses Argument nach dem Optionsbuchstaben interaktiv angegeben werden

–+ gefolgt von einer der unten beschriebenen Kommandozeilenoptionen setzt diese Option auf ihren voreingestellten Wert zurück; es können nur die Einstellungen zurückgesetzt werden, die keine Zeichenkette als Argument benötigen

–– gefolgt von einer der unten beschriebenen Kommandozeilenoptionen setzt diese Option auf das Gegenteil der Standardeinstellung; es können nur die Einstellungen verändert werden, die keine Argumente benötigen; im Anschluß an die Änderung wird die aktuelle Einstellung gezeigt

– (Unterstrich) gefolgt von einer der unten beschriebenen Kommandozeilenoptionen zeigt den aktuellen Wert dieser Einstellung; die Einstellung wird nicht verändert

+Lesskommando führt das angegebene *Lesskommando* jedesmal automatisch aus, wenn eine neue Datei angezeigt wird

V zeigt die Versionsnummer von less

q | :q | :Q | ZZ | ESC ESC beendet less

v startet einen Editor mit der aktuellen Datei; wenn in der Shellvariablen EDITOR nichts anderes bestimmt ist, wird der vi als Standardeditor benutzt

! Kommandozeile startet eine Shell und führt das angegebene externe Kommando aus; in der *Kommandozeile* kann der Name der aktuellen Datei mit einem Prozentzeichen '%', der Name der zuletzt davor aktuellen Datei mit dem Nummernzeichen '#' ersetzt werden; ein doppeltes Ausrufezeichen '!!' wiederholt die letzte Kommandozeile; ein einfaches Ausrufezeichen '!' startet eine interaktive Shell; in allen Fällen wird die Standardshell /bin/sh gestartet, wenn in der Umgebungsvariablen SHELL keine andere Shell bestimmt ist

| Marke Kommandozeile leitet die Zeilen von der ersten Bildschirmzeile bis zur Marke durch das in der *Kommandozeile* angegebene Kommando

Optionen:

die folgenden Optionen und Einstellungen können in der Kommandozeile beim Aufruf von less gesetzt werden. Es ist außerdem möglich die entsprechenden Einstellungen mit dem '-' Kommando zur Laufzeit von less vorzunehmen. Zusätzlich bietet less die Möglichkeit, bestimmte Optionen die jedesmal gesetzt werden sollen in der Umgebungsvariablen LESS zu speichern. Die so gesetzten Optionen werden jedesmal gelesen, wenn less gestartet wird, können aber immer von Kommandozeilenoptionen verdeckt werden.

-? zeigt eine kurze Übersicht über die Kommandos und Optionen von less

-a die Vorwärtssuchfunktionen fangen in der letzten Bildschirmzeile an, überspringen also den aktuellen Bildschirm

-b Nr veranlaßt less die mit Nr angegebene Anzahl von Puffern für die Anzeige zu Benutzten; jeder Puffer ist ein Kilobyte groß; wenn less von der Standardeingabe liest, werden die Puffer automatisch angefordert (siehe die Option '-B')

- B** unterdrückt die automatische Anforderung neuer Puffer; es werden nur die mit der ‘-b’ Option bereitgestellten oder die standardmäßig eingestellten zehn Puffer für die Speicherung der von der Standard-eingabe gelesenen Daten benutzt; werden mehr Daten gelesen, als Pufferplatz frei ist, wird der älteste Puffer überschrieben
- c** jede neue Bildschirmseite wird von der ersten Zeile an neu aufgebaut; normalerweise wird jede neue Bildschirmseite durch Scrollen des Bildschirms angezeigt
- C** der Bildschirm wird vor jeder neuen Seite gesösch, sonst wie ‘-c’
- d** unterdrückt Warnungen bzw. Fehlermeldungen auf “dummen” Terminals, wenn bestimmte Funktionen wie Bildschirmlöschen oder Rückwärtsscrollen nicht zur Verfügung stehen
- e** *less* beendet automatisch, wenn das Dateiende zum zweiten mal erreicht wird; normalerweise kann *less* nur ausdrücklich durch ein entsprechendes Kommando verlassen werden
- E** *less* wird automatisch beim (ersten) Dateiende verlassen
- f** erzwingt die Anzeige, auch von Dateien mit nichtdruckbaren Zeichen
- h** *Nr* es werden höchstens die angegebene Anzahl Zeilen rückwärts gescrollt; sollen mehr als die angegebene Anzahl Zeilen zurückgeblättert werden, findet der Bildschirmaufbau wie beim Vorwärtsblättern von der ersten Zeile an statt; wenn das Terminal kein Rückwärtsscrollen unterstützt, wird ‘-h0’ automatisch angenommen
- i** Groß- und Kleinschreibung werden nicht unterschieden; wenn ein Buchstabe im Suchmuster groß geschrieben ist, wird diese Option ignoriert
- j** *Nr* die Zielzeile bei einer Suche oder einer direkten Positionierung wird an der benannten Stelle angezeigt; eine negative Zahl zählt die Zielzeile vom unteren Bildschirmrand anstelle des oberen
- k** *Datei* liest eine alternative Tastaturbelegung aus der benannten Datei; wenn keine Datei angegeben ist, wird die Datei ‘.less’ im Heimatverzeichnis gelesen; eine Datei mit alternativer Tastaturbelegung kann mit dem *lesskey* Kommando erzeugt werden
- m** zeigt die aktuelle Dateiposition in Bytes oder Prozent als Eingabeaufforderung nach jeder Seite; normalerweise dient der Doppelpunkt ‘:’ als Eingabeaufforderung
- M** zeigt die aktuelle Dateiposition in Zeilen und den Dateinamen als Eingabeaufforderung
- n** die Zeilennummer wird weder in der Eingabeaufforderung, noch beim Aufruf des Editors benutzt
- N** jeder Zeile wird in der Anzeige eine Zeilennummer vorangestellt
- o** *Datei* kopiert die gelesene Eingabe in die benannte *Datei*, wenn aus einer Pipeline gelesen wird; wenn die Datei schon existiert, muß der Benutzer vor dem überschreiben die Aktion bestätigen
- O** *Datei* wie ‘-o’, es wird aber vor dem Überschreiben einer existierenden Datei nicht nachgefragt
- p** *Muster* sucht sofort das erste Auftreten vom *Muster* in der ersten angegebenen Datei
- P** *Prompt* ändert die Eingabeaufforderung (*Prompt*); ‘-Pm’ ändert den Prompt für die ‘-m’ Option, und ‘-PM’ ändert entsprechend den Prompt für die ‘-M’ Option; die Syntax der Prompt Zeichenkette mit ‘man less’ in den englischen Manualpages nachzulesen
- q** der Alarmton des Terminals wird nicht bei kleineren Fehlern, wie z. B. das Vorwärtsblättern am Dateiende ausgelöst; wenn das Terminal eine optische Warnung unterstützt, wird die anstelle des akustischen Alarms benutzt
- Q** der Alarmton des Terminals wird unter keinen Umständen ausgelöst

- r** die Sonderzeichen (Control Sequenzen) werden nicht als Caret-Sequenz sondern ‘roh’ angezeigt
- s** mehrere Leerzeilen werden zu einer einzigen Leerzeile zusammengefaßt; das ist die Standardeinstellung für die Anzeige von groff Ausgabe
- S** überlange Zeilen werden einfach abgeschnitten; normalerweise wird der Rest überlanger Zeilen in der nächsten Zeile angezeigt
- t** *Tag* zeigt die Datei, in der vom ctags Programm die Marke ‘Tag’ gefunden wurde; für diese Option muß eine ‘tags’ Datei vom ctags Programm im aktuellen Verzeichnis angelegt werden
- T** *Tagdatei* die benannte Tagdatei wird anstelle der Datei ‘tags’ im aktuellen Verzeichnis für die ‘-t’ Option benutzt
- u** Rückschritt und Wagenrücklauf werden roh (unverändert) an das Terminal geschickt
- U** Rückschritt und Wagenrücklauf werden als Control-Sequenzen der Option ‘-r’ entsprechend an das Terminal geschickt; normalerweise werden Rückschritte, die von einem Unterstrich gefolgt werden durch einen unterstrichenen Buchstaben, Rückschritte die von dem gleichen Buchstaben eingeleitet wie gefolgt werden durch einen fettgedruckten Buchstaben dargestellt; alle anderen Rückschritte und die von einem Zeilenvorschub gefolgt werden Wagenrücklaufzeichen werden ignoriert
- w** die Bildschirmzeilen nach dem Dateiende werden durch Leerzeilen anstelle der voreingestellten Tilde ‘~’ dargestellt
- x** *Nr* die Tabulatorweite wird auf die gegebene Anzahl Stellen gesetzt
- y** *Nr* setzt eine Grenze von Zeilen, bis zu der das vorwärtsscrollen möglich ist; wenn mehr als die gesetzte Anzahl Zeilen vorwärts geblättert werden soll, wird der Bildschirm der ‘-c’ oder ‘-C’ Option entsprechend neu aufgebaut
- z** *Nr* ändert die Schrittweite für das Blättern auf die angegebene Zeilenzahl
- + eine Kommandozeilenoption, die mit einem Pluszeichen beginnt, wird als Initialisierungskommando für *less* interpretiert, und als erstes Kommando von *less* ausgeführt; wenn anstelle eines einzelnen Pluszeichens zwei Pluszeichen benutzt werden, wird das darauffolgende Kommando bei allen Dateien der Kommandozeilenliste als erstes ausgeführt

Umgebungsvariable

less unterstützt die folgenden Umgebungsvariablen:

COLUMNS die Zeilenlänge für den Ausgabebildschirm

EDITOR der Editor für das ‘v’ Kommando

HOME das Heimatverzeichnis des Anwenders; hier wird die ‘.less’ Datei gesucht

LESS eine Zeichenkette mit Kommandozeilenoptionen

LESSBINFMT das Format zur Anzeige binärer Zeichen, die keine Controlzeichen sind

LESSCHARDEF legt einen Zeichensatz fest (ascii oder latin1)

LESSCHARSET beschreibt einen anderen Zeichensatz direkt

LESSEDT die Kommandozeile für den Editor

LESSHELP der absolute Name der Hilfsdatei

LINES die Anzahl der Bildschirmzeilen

SHELL die Shell für die externen Kommandos

TERM die Terminalbezeichnung wie sie in /etc/termcap zu finden ist

Siehe auch:

`more`, Seite 115 und `lesskey(1)`

Autor:

Mark Nudelman

1.44 *ln*

Funktion:

ln (link) erzeugt einen Verzeichniseintrag einer existierenden Datei unter anderem Namen

Syntax:

ln [*Optionen*] *Quelle* [*Ziel*]

ln [*Optionen*] *Quelle* ... *Zielverzeichnis*

Beschreibung:

Jede Datei wird bei ihrer Erzeugung mit ihrem Namen in ein Verzeichnis eingetragen. Dieser Eintrag enthält außerdem einen Verweis auf eine Inode, in der die Zugriffsrechte auf die Datei, der Dateityp und gegebenenfalls die Nummern der belegten Datenblöcke eingetragen sind.

Mit dem *ln* Kommando wird ein neuer Eintrag in einem Verzeichnis angelegt, der auf die Inode einer existierenden Datei zeigt. Diese Art Link wird als 'Hardlink' bezeichnet. Weil die Zugriffsrechte auf die Datei in der Inode bestimmt werden, sind die Zugriffsrechte auf alle Links einer Datei gleich.

Hardlinks können nur auf dem Datenträger angelegt werden, auf dem sich die Datei (und damit die Inode) selbst befindet. Um Links über die Dateisystemgrenzen hinweg anlegen zu können, bietet Linux die Möglichkeit 'symbolischer Links'. In diesen Links ist der absolute Pfad gespeichert, auf dem die gelinkte Datei gefunden werden kann. Ein Zugriff auf diese Datei wird dann vom Betriebssystem automatisch auf die gelinkte Datei umgelenkt.

Ist das letzte Argument des Aufrufs ein existierendes Verzeichnis, so werden alle als *Quelle* aufgelisteten Dateien mit entsprechenden Namen im *Zielverzeichnis* verbunden. Wird nur eine einzige *Quelle* benannt, so wird ein Link unter diesem Namen im aktuellen Verzeichnis angelegt. Normalerweise löscht *ln* keine existierenden Dateien. Es werden standardmäßig „hardlinks“ angelegt. Links auf Verzeichnisse oder auf Dateien in anderen Dateisystemen können nur mit symbolischen Links realisiert werden.

Gelegentlich verändert sich das Verhalten eines Programms, wenn es durch einen Link unter einem anderen Namen aufgerufen wird. (Das funktioniert natürlich nur, wenn diese Änderung im Programm vorgesehen ist.)

Optionen:

- b** sichert Dateien anstatt sie zu löschen
- d** ermöglicht Hardlinks auf Verzeichnisse
- f** löscht bestehende Dateien
- i** fragt vor dem Löschen nach Bestätigung
- s** macht symbolische Links anstelle von harten
- v** gibt die Dateinamen auf den Bildschirm

-S *Endung* setzt die Endung für die Sicherung von Dateien auf *Endung*. Standardwert ist ‘~’. Die Endung kann auch mit der Umgebungsvariablen `SIMPLE_BACKUP_SUFFIX` bestimmt werden. Die Option **-S** hat Vorrang vor der Umgebungsvariablen.

-V {numbered, existing, simple} bestimmt die Art der Sicherungskopien. Die Art der Sicherung kann auch mit der Umgebungsvariablen `VERSION_CONTROL` bestimmt werden. Die Option **-V** hat auch hier die höhere Priorität. Die Optionen bedeuten hierbei:

numbered macht immer nummerierte Backups

existing macht nummerierte Backups nur für bereits nummerierte Dateien

simple macht immer nur einfache Backups

Autor:

Mike Parker und David MacKenzie

1.45 login

Funktion:

login prüft die Identität des Benutzers und startet eine Shell

Syntax:

login [*Name*]

Beschreibung:

Der Zugang zu Unix-Systemen ist normalerweise nur eingetragenen Benutzern möglich. Diesen Anwendern ist dann die Ausführung von Programmen und das Lesen bzw. Schreiben von Dateien in dem Umfang möglich, wie es die *Zugriffsrechte* (→ Seite 184) im Dateisystem erlauben. Damit kann ein Unix-Rechner weitgehend vor Fehlern oder gar Mißbrauch geschützt werden. Um die Identität eines Benutzers festzustellen, wird zu Beginn jeder “Sitzung” ein **login** durchgeführt, bei dem zu dem Benutzernamen noch ein Paßwort abgefragt wird. Erst wenn das richtige Paßwort eingegeben ist, läßt sich das System benutzen. Um während einer Sitzung die Rechte eines anderen Users (z. B. **root** für Verwaltungszwecke) zu erhalten kann außer dem **su** Befehl (→ Seite 140) auch ein **login** wie jeder andere Befehl in der Shell aufgerufen werden. Man erhält nach dem **login** eine neue Shell mit den Rechten des Users. Nach Verlassen dieser Shell mit **exit** ist man wieder in der eigenen Shell mit den eigenen Rechten.

Optionen:

Name es wird ein **login** für den Anwender *Name* durchgeführt

Autor:

Michael Glad

1.46 logname

Funktion:

logname zeigt den Benutzernamen

Syntax:**logname****Beschreibung:**

Das Kommando **logname** zeigt den Benutzernamen wie er in der Datei `/etc/utmp` gespeichert ist. Wenn für das Terminal von dem aus das Kommando gestartet wird kein Eintrag gefunden wird, gibt **logname** eine Fehlermeldung aus und beendet mit dem Status 1; sonst wird mit dem Status 0 beendet.

1.47 *ls*

Funktion:

ls (list) zeigt den Inhalt eines Verzeichnisses

Syntax:

```
ls [-abcdgiklmnpqrstuxABCFLNQRSUX1] [-w Spalten] [-T Spalten] [-l Muster][--all] [--escape]
[--directory] [--inode] [--kilobytes] [--numeric-uid-gid][--hide-control-chars] [--reverse] [--size]
[--width=Spalten] [--tabsize=Spalten] [--almost-all] [--ignore-backups] [--classify] [--file-type]
[--ignore=Muster][--dereference] [--literal] [--quote-name] [--recursive]
[--sort={none, time, size, extension}] [--format={long, verbose, commas, across, vertical, single-column}]
[--time={atime, access, use, ctime, status}] [Pfad ...]
```

Beschreibung:

ls gibt den Inhalt der Verzeichnisse des Dateisystems an.

Das Standardausgabeformat von **ls** hängt vom Typ des Ausgabegerätes ab. Auf einem Terminal ist die mehrspaltige Ausgabe das Standardformat. In allen anderen Fällen wird die Ausgabe einspaltig ausgeführt. Das Verhalten des **ls** Kommandos läßt sich nicht mehr durch umbenennen in `ll` `dir` `vdir` etc. verändern. Stattdessen sind die Kommandos `dir` und `vdir` als separate Binärdateien mit entsprechenden Standardformaten verfügbar.

Optionen:

- a** zeigt alle Dateien im Verzeichnis, auch die deren Name mit `.'` beginnt
- b** zeigt nichtdruckbare Zeichen in Dateinamen als „Backslash Sequenz“ mit alphabetischen oder oktalen Werten wie sie in C üblich sind
- c** sortiert die Dateien nach der Zeit der letzten Statusveränderung
- d** zeigt Unterverzeichnisse wie normale Dateien anstelle ihres Inhaltes
- i** zeigt die Nummer der Inode zu jeder Datei
- k** die Dateigröße wird in Kilobytes angegeben, auch wenn `POSIXLY_CORRECT` gesetzt ist
- l** außer dem Namen werden der Typ, die Rechte, die Anzahl der Hardlinks, der Besitzer, die Gruppe, die Größe und die Zeitmarke angezeigt
- m** gibt die Dateinamen in einer Reihe getrennt durch Kommas aus
- n** gibt die Benutzer und Gruppen mit ihren ID's anstelle der Namen aus

- q** gibt Fragezeichen anstelle von nichtdruckbaren Zeichen in Dateinamen
- r** zeigt das Verzeichnis in umgekehrter Reihenfolge
- s** zeigt die Größe der Dateien in Kilobytes. Wenn `POSIXLY_CORRECT` gesetzt ist, wird die Größe in Blöcken zu 512 Bytes angezeigt
- t** sortiert nach Zeit anstelle des Namens
- u** sortiert nach letzter Zugriffszeit anstelle der Änderungszeit
- x** sortiert in horizontaler Richtung
- A** zeigt alle Dateien außer `‘.’` und `‘..’`
- B** ignoriert Backups (mit Endung `‘~’`)
- C** listet in vertikal sortierten Spalten
- F** hängt verschiedene Symbole an die Dateinamen:
 - * steht hinter ausführbaren Dateien
 - / steht hinter Verzeichnissen
 - @ markiert symbolische Links
 - | markiert FiFo's
 - = markiert socketsAlles andere sind reguläre Dateien
- L** zeigt den Inhalt der symbolisch gelinkten Verzeichnisse anstelle des Linkfiles
- N** gibt Dateinamen ohne Quotes aus
- Q** gibt Dateinamen in Quotes aus
- R** zeigt rekursiv den Inhalt aller Unterverzeichnisse
- S** sortiert nach Größe
- U** unsortiert
- X** sortiert nach Endung
- 1** einspaltig
- w** *Spalten* Bildschirmbreite in *Spalten*
- T** *Spalten* Tabulatorbreite in *Spalten*
- I** *Muster* ignoriert Dateien mit *Muster* im Namen

Autor:

Richard Stallman und David MacKenzie

1.48 **man**

Funktion:

man zeigt die Handbuchseiten zu den Linux-Kommandos

Syntax:

man [-adfhktw] [-m *System*] [-p *Zeichenkette*] [-M *Pfad*] [-P *Pager*] [-S *List*] [*Section*] *Name* ...

Beschreibung:

man gibt die **manualpages** zum Befehl *Name* aus. Diese englischen Handbuchseiten sind Teil des Linux-Systems. Wenn die Seiten im rohen nroff-Format vorliegen, werden sie automatisch formatiert. Die Anzeige erfolgt durch einen „Pager“ der immer nur eine Bildschirmseite zur Zeit anzeigt. Solche „Pager“ stehen als eigenständige Programme zur Verfügung. Als Standardpager wird **less** benutzt.

Die Manualpages werden traditionell in verschiedene Kapitel unterteilt:

1. die Benutzerkommandos
2. die Systemaufrufe
3. die C-Bibliotheksfunktionen
4. die Beschreibungen der Gerädateien
5. die Dateiformate
6. Spiele
7. die Makropakete für die Textformatierer
8. die Kommandos für den Systemverwalter

Außerdem werden noch die Sektionen ‘l’ für lokale Erweiterungen und ‘n’ für neue Kommandos unterstützt,

Optionen:

- M *Pfad*** verdeckt die MANPATH Umgebungsvariable; auf dem *Pfad* wird nach den Manualpages gesucht
- P *Pager*** bestimmt das externe Programm *Pager* als Anzeigefilter
- S *Liste*** *Liste* ist eine durch Komma getrennte Liste von Kapiteln (Sektionen) des Manuals; die *Liste* verdeckt die MANSECT Umgebungsvariable
- a** zeigt alle Manualpages die auf dem *Pfad* gefunden werden
- d** gibt Fehlerinformationen zum entwanzen
- f** wie **whatis**
- h** gibt eine Hilfszeile aus
- k** wie **apropos**
- p *Namen*** gibt die Präprozessoren für **groff** an
- t** formatiert die Manualpages mit dem Kommando ‘**groff -Tascii -mandoc**’ und leitet das Ergebnis auf die Standardausgabe
- w** gibt nur den Pfad der Manualpages aus, nicht deren Inhalt

Siehe auch:

das **info** Kommando für das Texinfo System und das Shellkommando **help** auf Seite 36

Autor:

John W. Eaton

1.49 mcopy

Funktion:

mcopy kopiert Dateien von/nach DOS-Filesystemen nach/von Linux Dateisystemen

Syntax:

mcopy [-tnwm] *Quelldatei Zieldatei*

mcopy [-tnwm] *Quelldatei [Quelldatei ...] Zielverzeichnis*

Beschreibung:

mcopy kopiert Dateien zwischen MS-DOS und Linux Dateisystemen. Ein MS-DOS Dateiname muß mit einer Laufwerksbezeichnung angegeben werden. Wird für eine MS-DOS Datei kein absoluter Pfad benannt, gilt das aktuelle DOS-Verzeichnis. Dieses Verzeichnis wird in der Datei ‘~/mcpwd’ im Heimatverzeichnis des Anwenders bestimmt und durch das **mcd** Kommando verändert.

Optionen:

- t** konvertiert CR/LF in LF (für Texttransfer)
- n** unterdrückt Warnungen vor dem Überschreiben existierender Dateien
- v** gibt die Dateinamen vor dem Kopieren aus
- m** kopiert die Datei inklusive des Zeitstempels

Siehe auch:

mcd(1), **mread** Seite 117, **mwrite**(1), **/etc/mtools**, **~/mcpwd** und **mtools**, Seite 118

1.50 mdel

Funktion:

mdel löscht eine Datei in einem MS-DOS Dateisystem

Syntax:

mdel [-v] *msdosdatei ...*

Beschreibung:

mdel löscht eine oder mehrere Dateien in einem MS-DOS Dateisystem. **mdel** erwartet eine Bestätigung vor dem Löschen von “nur Lesen” Dateien.

Optionen:

- v** zeigt den Namen jeder Datei vor dem Löschen noch einmal an

Siehe auch:

mtools auf Seite 118

1.51 mdir

Funktion:

mdir zeigt den Inhalt eines DOS Verzeichnisses

Syntax:

mdir [-w] [*Verzeichnis*]

mdir [-w] *Datei* ...

Beschreibung:

mdir zeigt den Inhalt eines MS-DOS Verzeichnisses oder Information über MS-DOS Dateien. Wird kein *Verzeichnis* angegeben, so wird der Inhalt des aktuellen DOS Verzeichnisses angezeigt, das in der Datei '~/.mcwd' im Heimatverzeichnis des Anwenders festgelegt ist und mit dem **mcd** Kommando gewechselt wird.

Optionen:

-w gibt die Dateinamen in Spalten nebeneinander aus

Siehe auch:

mtools auf Seite 118

1.52 mformat

Funktion:

mformat richtet ein MS-DOS Dateisystem ein

Syntax:

mformat [-t *Spuren*] [-h *Köpfe*] [-s *Sektoren*] [-l *Label*] *Laufwerk*:

Beschreibung:

mformat richtet auf einer roh (low-level) formatierten Diskette ein MS-DOS Dateisystem ein.

Optionen:

-t Anzahl die Anzahl der Spuren

-h Anzahl die Anzahl der Köpfe (Seiten)

-s Anzahl die Anzahl der Sektoren pro Spur

-l Name das Label der Dislette

Siehe auch:

`fdformat` auf Seite 83 und `mttools` auf Seite 118

1.53 `mkdir`

Funktion:

mkdir erzeugt ein leeres Verzeichnis

Syntax:

mkdir [`-p`] [`-m` *Modus*] [`--path`] [`--mode=Modus`] *Verzeichnis* ...

Optionen:

-m *Modus* setzt die Rechte des Verzeichnisses auf *Modus*; der *Modus* wird wie beim Kommando `chmod` angegeben (→ Seite 45); der Standard und damit der Ausgangswert für relative Modes ist `0777` minus der Bits von `umask` (siehe bei `umask` auf Seite 42)

-p wenn ein Unterverzeichnis in einem nicht existierenden Verzeichnis angelegt werden soll, werden alle fehlenden Verzeichnisse angelegt

Siehe auch:

`ln` auf Seite 107 und `rmdir` auf Seite 130

Autor:

David MacKenzie

1.54 `mkfifo`

Funktion:

mkfifo erzeugt eine FIFO-Datei

Syntax:

mkfifo [`-m` *Modus*] [`--mode=Modus`]

Beschreibung:

mkfifo erzeugt eine FIFO-Datei (Named Pipe). Daten die in diese Datei geschrieben werden, können nur sequenziell in der gleichen Reihenfolge wieder gelesen werden. FIFO steht für “**F**irst **I**n **F**irst **O**ut. Die Zugriffsrechte auf die Datei werden aus der Bitdifferenz zwischen `0666` und der dem Wert von `umask` (→ Seite 42) errechnet, wenn nicht ausdrücklich ein anderer Modus angegeben ist.

Optionen:

-m *Modus* setzt bzw. ändert die Zugriffsrechte der Datei. *Modus* ist dabei wie bei `chmod` beschrieben.

Autor:

David MacKenzie

1.55 mmd

Funktion:

mmd erzeugt ein neues MS-DOS Verzeichnis

Syntax:

mmd [-v] *Verzeichnis* ...

Beschreibung:

mmd erzeugt ein oder mehrere neue MS-DOS Verzeichnisse im aktuellen DOS-Verzeichnis oder in dem absoluten Pfad. Verzeichnisnamen, die nicht der MS-DOS Konvention entsprechen werden automatisch auf die zulässigen Zeichen gekürzt. Existiert bereits eine Datei oder ein Verzeichnis mit dem gleichen Namen, wird das Kommando nicht ausgeführt.

Optionen:

-v zeigt die Namen der erzeugten Verzeichnisse an

Siehe auch:

mtools auf Seite 118

1.56 more

Funktion:

more zeigt Dateien seitenweise

Syntax:

more [-cdfslu] [-n] [+linenumber] [+/*pattern*] [*Name* ...]

Beschreibung:

more gibt Textdateien seitenweise auf dem Bildschirm aus. Nach jeder Bildschirmseite wird die Ausgabe angehalten und auf eine Eingabe des Benutzers gewartet.

In der Shellvariablen **MORE** können Kommandozeilenooptionen für **more** gespeichert werden, die bei jedem Aufruf automatisch ausgeführt werden.

Am Dateiende wird **more** automatisch beendet.

Wenn **more** eine Datei liest, wird auf der letzten Bildschirmzeile im ‘Prompt’ die prozentuale Position der aktuellen Bildschirmseite in der Datei angezeigt.

Wenn dieser Prompt angezeigt wird, erwartet **more** eine Eingabe des Benutzers. Die Eingaben bestehen in der Regel aus einem einzigen Tastendruck. Einige Kommandos können mit einer Zahlenangabe **N** kombiniert werden.

NLEERZEICHEN gibt die nächsten *N* Zeilen aus, oder einen kompletten Bildschirm, wenn keine Zahl angegeben ist

CTRL-D gibt die nächsten 11 Zeilen (einen halben Bildschirm) aus

d das Gleiche wie **^D**

Nz das Gleiche wie Leerzeichen; die Zahl *N* wird die neue Anzahl Zeilen pro Bildschirm

Ns überspringt die nächsten *N* Zeilen und zeigt die darauffolgenden Zeilen an

Nf überspringt die nächsten *N* Bildschirme, und zeigt die darauffolgenden Zeilen an

Nb springt *N* Bildschirmseiten rückwärts

NCTRL-B das Gleiche wie **b**

q oder **Q** beendet **more**

= zeigt die aktuelle Zeilennummer an

v startet den Editor **vi** mit der aktuellen Datei in der aktuellen Zeile

h das Hilfefkommando; gibt eine Übersicht über die **more** Kommandos

N/Ausdruck sucht das *N*-te auftreten des Ausdrucks vom aktuellen Bildschirm an vorwärts

N n sucht das *N*-te auftreten des zuletzt gesuchten Ausdrucks vorwärts

' geht zurück an die Position, von der das letzte Suchkommando gestartet wurde

! Kommandozeile startet eine Shell und führt die angegebene Kommandozeile aus

N:n springt zur nächsten Datei aus der Kommandozeilenliste, bzw. *N* Dateien weiter

N:p springt an den Anfang der aktuellen Datei, oder in die vorhergehende Datei aus der Kommandozeilenliste bzw. *N* Dateien zurück

:f zeigt den Namen der aktuellen Datei und die Position des aktuellen Bildschirms

:q oder **:Q** beendet **more**

. (Punkt) wiederholt das letzte Kommando

Optionen:

-N *N* ist eine ganze Zahl und setzt die Zeilenanzahl für den Bildschirm

-c veranlaßt **more** den Bildschirm beim Weiterblättern von oben nach unten neu aufzubauen, indem jede Zeile unmittelbar vor dem Überschreiben gelöscht wird. Diese Option funktioniert nur auf Terminals, die das Löschen einzelner Zeilen unterstützen

-d gibt einen längeren Prompt mit zusätzlicher Hilfe für den Anwender aus

-f es werden die Textzeilen anstelle der Bildschirmzeilen angezeigt. Dadurch werden Zeilen mit Controlsequenzen, wie sie z.B. von **groff** erzeugt werden, korrekt in einer Zeile angezeigt

-l ignoriert **^L** (Seitenvorschub). Standardmäßig wird die Ausgabe bei jedem Seitenvorschub angehalten. Ein Seitenvorschub am Anfang des Textes bewirkt ein Löschen des Bildschirms.

-s zeigt mehrere Leerzeilen in Folge als eine einzige an

-u unterdrückt die Behandlung von unterstrichenem Text.

+Zeilennummer beginnt die Ausgabe bei *Zeilennummer*

+Muster beginnt die Ausgabe zwei Zeilen vor dem ersten Auftreten von *Muster*

Siehe auch:

less auf Seite 101

Autor:

Eric Shienbrood, Geoff Peck, John Foderaro

1.57 mrd

Funktion:

mrd löscht ein MS-DOS Verzeichnis

Syntax:

mrd *Verzeichnis* ...

Beschreibung:

mrd löscht ein oder mehrere MS-DOS Verzeichnisse. Das Kommando wird mit Fehlermeldung abgebrochen, wenn ein Verzeichnis nicht leer ist.

Siehe auch:

mttools auf Seite 118

1.58 mread

Funktion:

mread liest eine Datei aus einem MS-DOS Dateisystem und schreibt (kopiert) sie in ein Linux-Dateisystem

Syntax:

mread [-tnm] *Msdosdatei Unixdatei*

mread [-tnm] *Msdosdatei* ... *Unixverzeichnis*

Beschreibung:

mread liest eine *MS-DOS Datei* und kopiert sie in eine *Unixdatei*. Wird anstelle einer *Unixdatei* ein *Unixverzeichnis* angegeben, so werden alle MS-DOS Dateien in das *Unixverzeichnis* kopiert.

Optionen:

-t übersetzt CR/LF in LF

-n unterdrückt Warnung vor dem Überschreiben einer Datei

-m kopiert den Zeitstempel der *msdosdatei*

Siehe auch:

mttools auf Seite 118

1.59 mtools

Funktion:

mtools ist eine Sammlung von Werkzeugen zur Bearbeitung von MS-DOS Dateisystemen auf Diskette oder Festplatte.

Beschreibung:

Die verschiedenen Werkzeuge (Tools) haben jeweils ein MS-DOS Vorbild. Die Verwendung und die Optionen entsprechen sich weitgehend.

Dateinamen und Pfade auf MS-DOS Dateisystemen müssen mit einer Laufwerksbezeichnung beginnen. Diese 'Laufwerke' müssen mit den entsprechenden Gerätedateien in der Datei `/etc/mtools` verknüpft werden.

Im Unterschied zu den Originalkommandos kann bei den **mtools** als Trenner von DOS-Verzeichnisnamen sowohl `\` als auch `/` verwendet werden. Die **mtools** akzeptieren auch Wildcards. Dabei gelten die Unix-Regeln. Das heißt, z. B. `*` ersetzt alle Dateien eines Verzeichnisses wie `*.*` bei MS-DOS.

Bei der Benutzung von `\` oder Wildcards muß der Pfad durch einfache Quotes eingeschlossen werden, um ihn vor der Interpretation durch die Shell zu schützen.

Die Optionen werden im Unixstil von einem `-` eingeleitet, nicht von einem `/` wie bei MS-DOS.

In der Datei `~/mcdw` im Heimatverzeichnis des Benutzers ist das aktuelle MS-DOS Verzeichnis für alle **mtools** festgelegt. Dieses Verzeichnis kann mit dem **mcd** Kommando gewechselt werden.

Autor:

Emmet P. Gray

1.60 mv

Funktion:

mv (move) verschiebt eine Datei oder benennt sie um

Syntax:

mv [*Optionen*] *Quelle Ziel*

mv [*Optionen*] *Quelle ... Verzeichnis*

Beschreibung:

mv verschiebt eine oder mehrere Datei(en) bzw. Verzeichnis(se), oder benennt sie um. Ein Verzeichnis kann nicht über die Grenzen eines Dateisystems hinweg verschoben werden.

Optionen:

-b sichert Dateien vor dem Überschreiben

-f überschreibt existierenden Zielf Dateien rücksichtslos

-i erwartet interaktiv eine Bestätigung vor dem Überschreiben existierender Zielf Dateien

-u verschiebt Dateien nur, wenn sie neuer sind als die gleichnamigen Zielf Dateien

-v meldet jede Aktion

-S *Endung* bestimmt die *Endung* für einfaches Backup. Voreinstellung ist `~`

-V {**numbered**, **existing**, **simple**} kontrolliert die Art des Backups. Die Parameter sind beim Befehl **cp** auf Seite 50 erklärt

Siehe auch:

ln auf Seite 107 und cp auf Seite 50

Autor:

Mike Parker und David MacKenzie

1.61 newgrp

Funktion:

newgrp ändert die Gruppenkennung des aktuellen Benutzers

Syntax:

newgrp [*Gruppe*]

Beschreibung:

Das **newgrp** Kommando ändert die aktive Gruppenkennung des Anwenders. Ohne Angabe einer Gruppe wird zu der in der `/etc/passwd` Datei festgelegten Standardgruppe des Benutzers gewechselt. Sonst wird in die angegebene Gruppe gewechselt, wenn der Benutzer in der Datei `/etc/group` als Mitglied dieser Gruppe eingetragen ist. Das **newgrp** Kommando unterstützt keine paßwortgeschützten Gruppen.

Die Gruppe kann nur mit ihrem Namen angegeben werden.

Autor:

Michael Haardt, Peter Orbaek

1.62 nice

Funktion:

nice läßt ein Programm mit veränderter Priorität laufen

Syntax:

nice [`-n` *Nettigkeit*] [`-`*Nettigkeit*] [`--adjustment=`*Nettigkeit*] [*Kommando* [*Argument* ...]]

Beschreibung:

In einem Multitasking Betriebssystem wie Linux muß die Prozessorzeit auf verschiedene Prozesse verteilt werden. Dazu gibt es einen ‘Sheduler’ der dafür sorgt, daß die Prozessorzeit möglichs optimal zugeteilt wird. Prozesse die auf das Ergebnis eines anderen Prozesses, ein Ereignis oder ein Signal warten können beispielsweise solange ‘schlafen’ bis das erwartete Ereignis eingetreten ist. Der Sheduler weckt die schlafenden Prozesse dann auf und teilt ihnen wieder Prozessorzeit zu. Trotzdem gibt es natürlich in der Regel mehr als einen lauffähigen Prozess. Und der Sheduler muß nach bestimmten Regeln den lauffähigen Prozessen Rechenzeit zuteilen. Dabei benutzt er unter anderem den **nice** Wert. Will ein Anwender **nice** das heißt nett zu den anderen Benutzern des Systems sein, startet er die Prozesse, die ruhig etwas länger dauern dürfen mit dem **nice** Kommando. Ein negatives **nice** ist nur dem Superuser **root** erlaubt.

Wenn kein Wert angegeben ist, wird die **Nettigkeit** um 10 Punkte erhöht. Die maximale **Nettigkeit** ist 19 Punkte. Nach unten kann der Superuser seine **Nettigkeit** bis -20 abkühlen.

Optionen:**-n** *Wert***-***Wert* setzt nice auf *Wert***Siehe auch:**

nohup auf Seite 121

Autor:

David MacKenzie

1.63 nl**Funktion****nl** nummeriert die Zeilen in einer Datei**Syntax**

nl [-h *Stil*] [-b *Stil*] [-f *Stil*] [-p] [-d *zwei Zeichen*] [-v *Nummer*] [-i *Nummer*] [-l *Nummer*] [-s *Zeichenkette*]
 [-w *Nummer*] [-n *ln,rn,rz*] [-header-numbering=*Stil*] [-body-numbering=*Stil*] [-footer-numbering=*Stil*] [-first-
 page=*Nummer*] [-page-increment=*Nummer*] [-no-renumber] [-join-blank-lines=*Nummer*] [-number-separator=*Zeichenkette*]
 [-number-width=*Nummer*] [-number-format=*ln,rn,rz*] [-section-delimiter=*zwei Zeichen*] [*Datei ...*]

Beschreibung:

nl gibt die Zeilen einer oder mehrerer Dateien (oder der Standardeingabe) mit Zeilennummern auf die Standardausgabe. Es können dabei die Zeilen einer (logischen) Seite in einen Kopf, einen Körper und einen Fuß unterteilt werden, die jeweils einzeln und in unterschiedlichen Stilen nummeriert werden. Jeder Teil kann auch leer sein. Wenn vor dem ersten Kopfteil bereits Zeilen vorhanden sind, werden diese Zeilen wie ein Seitenkörper nummeriert.

Die Nummerierung beginnt auf jeder Seite neu. Mehrere Dateien werden als ein einziges Dokument betrachtet, und die Zeilennummer nicht zurückgesetzt.

Der Kopfteil wird durch eine Zeile eingeleitet, die nur die Zeichenkette ‘\ : \ : \ :’ enthält. Der Körper wird entsprechend durch ‘\ : \ :’ und der Fuß durch ‘\ :’ eingeleitet. In der Ausgabe werden diese Zeilen als Leerzeilen ausgegeben.

Optionen:

-h *Stil* bestimmt die Art der Zeilennummerierung für die Kopfzeile; das Nummerntrennzeichen wird auch den nicht Nummerierten Zeilen vorangestellt; als *Stil* werden folgende Zeichen erkannt

a alle Zeilen werden nummeriert

t die leeren Zeilen werden nicht nummeriert (Voreinstellung für den Körper)

n die Zeilen werden nicht nummeriert (Voreinstellung für Kopf und Fuß)

p **Ausdruck** nur die Zeilen in denen der reguläre *Ausdruck* vorkommt werden nummeriert

-b *Stil* bestimmt die Art der Zeilennummerierung für den Körper

-f *Stil* bestimmt die Art der Zeilennummerierung für den Fuß

- p** die Zeilen aller Seiten werden fortlaufend nummeriert
- v Nummer** die erste Zeile jeder logischen Seite bekommt die angegebene Nummer
- i Nummer** die Schrittweite für die Nummerierung
- l Nummer** die angegebene Anzahl aufeinanderfolgender Leerzeilen werden als eine Zeile angesehen, und die letzte Zeile nummeriert; wenn weniger Leerzeilen in Folge auftreten, werden sie nicht nummeriert; Leerzeilen dürfen auch keine Leerzeichen oder Tabulatoren enthalten
- s Zeichenkette** setzt die *Zeichenkette* als Nummerntrennzeichen zwischen Zeilennummer und Text; Voreinstellung ist TAB
- w Nummer** die Zeilennummern erhalten die angegebene Anzahl Stellen; Voreinstellung ist 6
- n {ln, rn, rz}** die Zeilennummern werden in dem angegebenen Stil ausgegeben; dabei bedeutet
 - ln** linksbündig, ohne führende Nullen
 - rn** rechtsbündig, ohne führende Nullen
 - rz** rechtsbündig, mit Nullen auf die volle Stellenzahl aufgefüllt
- d zwei Zeichen** die zwei Zeichen werden zur Trennung von Kopf, Körper und Fußteil benutzt, Voreinstellung ist '\ :'

Siehe auch:

pr auf Seite 126

Autor:

Scott Bartram, David MacKenzie

1.64 nohup

Funktion:

nohup läßt ein Programm die Signale SIGHUP SIGINT SIGQUIT und SIGTERM ignorieren

Syntax:

nohup *Kommando* [*Argument ...*]

Beschreibung:

nohup schützt ein Programm vor den Hangup Signalen. Dadurch kann es im Hintergrund weiterlaufen, auch wenn der Benutzer sich ausloggt. Normalerweise würden mit der Loginshell alle Prozesse des Anwenders durch ein SIGHUP beendet.

Der Prozeß geht nicht automatisch in den Hintergrund, sondern muß mit einem '&' am Ende der Kommandozeile dorthin gebracht werden. Die Shedulerpriorität eines mit **nohup** gestarteten Programms wird um 5 erhöht. Wenn die Standardausgabe des Programms ein Terminal ist, so wird sie automatisch gemeinsam mit der Standardfehlerausgabe in die Datei *nohup.out* umgeleitet. Ist das aktuelle Verzeichnis schreibgeschützt, wird die Datei im HOME Verzeichnis angelegt.

nroff

Textformatierer, siehe **groff** auf Seite 93

1.65 od

Funktion:

od (octal dump) zeigt Dateien im oktalen, hexadezimalen oder in anderen Formaten an.

Syntax:

```
od [-abcdphiloxv] [-s Länge] [-w Anzahl] [-A Positionsformat] [-j Anzahl] [-N Anzahl] [-t Format]
[--skip-bytes=Anzahl] [--address-radix=Positionsformat][--read-bytes=Anzahl]
[--format=Format] [--output-duplicates][--strings[=Anzahl]] [--width[=Anzahl]] [Datei ...]
```

Beschreibung:

od liest die angegebenen Dateien oder die Standardeingabe (wenn keine Datei oder anstelle einer Datei ‘-’ angegeben ist), und gibt die Bytes formatiert und kodiert auf die Standardausgabe.

Jede Ausgabezeile enthält in der ersten Spalte die Positionsnummer des ersten in der Zeile dargestellten Bytes (vom Dateianfang gezählt). In den darauffolgenden Spalten werden die Daten aus der Datei in einem durch die Optionen kontrollierten Format angezeigt.

Als Formate für die Darstellung von Bytes sind vor allem fünf Typen von Bedeutung:

Binärzahl Ein Byte besteht auf den meisten Computern aus acht Bits, die jeweils gesetzt oder ungesetzt sein können. Diesen beiden Zuständen kann einfach jeweils eine Zahl zugeordnet werden – die 1 für gesetzt und die 0 für ungesetzt – und damit ein Byte als eine achtstellige Binärzahl dargestellt werden. Eine achtstellige Binärzahl kann $2^8=256$ verschiedene Kombinationen von Nullen und Einsen enthalten, und damit beispielsweise von 0 bis (dezimal) 255 zählen. Mit diesen Zahlen läßt sich sogar rechnen, wie mit den “normalen” Dezimalzahlen. Dieser faszinierenden Eigenschaft, die Gegenstand der Booleschen Algebra ist, verdanken wir erst die Möglichkeit, Computer zum Rechnen zu benutzen.

Character Die modernen Computer werden zu einem großen Teil nicht zum Berechnen von Zahlen, sondern zum Bearbeiten von allgemeineren Daten benutzt. Die meisten dieser Daten sind Texte irgendeiner Art. Daher gibt es schon seit den frühesten Anfängen der elektronischen Datenverarbeitung Tabellen, die den verschiedenen Bytes Buchstaben und andere Zeichen zuordnen. Die wichtigste dieser Tabellen ist der “American Standard for Coded Information Interchange”, die sogenannte ASCII Tabelle. In dieser Tabelle werden 128 Zeichen verschiedenen Bytes zugeordnet. Das höchste Byte ist bei Computern mit acht Bits pro Byte immer null. Neben dem Alphabet werden in dieser Tabelle eine ganze Reihe Satz- und Sonderzeichen sowie Zeichen zur Terminalsteuerung festgelegt. Internationale Zeichen, wie zum Beispiel die deutschen Umlaute, sind in dieser Tabelle nicht enthalten. Für die internationalen Zeichensätze gibt es verschiedene Tabellen.

Dezimalzahl Wahrscheinlich wegen der passenden Anzahl Finger rechnen wir normalerweise im dezimalen Zahlensystem. Dieses Zahlensystem benutzt alle Ziffern von 0 bis 9. Wie schon gesagt, kann ein Byte durch eine Dezimalzahl dargestellt werden, indem alle möglichen Bytes aus je acht Bits von 0 bis 255 abgezählt werden. Es ist aber ebensogut möglich, das höchste Bit eines Bytes als Vorzeichen zu interpretieren. Damit werden alle Bytes auf die Zahlen von -128 bis 127 abgebildet. Bei der Darstellung von Binärzahlen beziehungsweise Bytes durch Dezimalzahlen muß also immer darauf geachtet werden, ob die Dezimalzahlen mit einem Vorzeichen behaftet sind oder nicht.

Oktalzahl Genauso, wie es möglich ist ein Zahlensystem auf den Ziffern 0 und 1 aufzubauen, kann man auch auf anderen Ziffernfolgen Zahlensysteme aufbauen. Die Zahlen aus dem Zahlensystem zur Basis 8 heißen Oktalzahlen. Die Oktalzahlen benutzen nur die Ziffern von 0 bis 7. Genau wie im Dezimalsystem werden beim Zählen die Ziffern der niedrigsten Stelle solange erhöht, bis die höchste Ziffer erreicht ist, in diesem Fall also die 7, und danach wird die nächsthöhere Stelle um eins erhöht, und so weiter. Damit ist die Oktalzahl 10 gleich der dezimalen 8. Der Vorteil des oktalen Zahlensystems besteht in der leichten Umrechenbarkeit von Oktalzahlen in Binärzahlen und umgekehrt. Jede Oktalziffer kann als dreistellige

Binärzahl dargestellt werden, und die Umrechnung einer Oktalzahl in eine Binärzahl erfolgt einfach, indem die den Ziffern entsprechenden Binärzahlen aneinandergehängt werden. Umgekehrt kann aus einer Binärzahl eine Oktalzahl gemacht werden, indem die Binärzahl in Gruppen zu je drei Bits zerlegt wird, und dann für jede Gruppe die entsprechende Oktalziffer eingesetzt wird. Auf diese Weise können also Binärzahlen, die sonst schon wegen ihrer Länge nur sehr schwer lesbar sind, in Handliche Stücke zerlegt werden. Allerdings besteht ein Byte aus acht Bits, eine dreistellige Oktalzahl entspricht aber einer Binärzahl mit 9 Bits.

Wenn aus dem Zusammenhang eines Textes nicht deutlich wird, in welchem Zahlensystem gerechnet wird, kann durch eine kleine tiefgestellte Zahl die entsprechende Zahlenbasis angegeben werden. Zum Beispiel macht die Gleichung

$$\text{ESC} = 0011011_2 = 27_{10} = 033_8 = 1\text{B}_{16}$$

Hexadezimalzahl Es gibt in unserem Kulturkreis nur 10 gebräuchliche Ziffern. Aber es gibt eigentlich keinen Grund, auf Zahlensysteme mit mehr Ziffern zu verzichten. Das im Computerbereich gebräuchliche Hexadezimalsystem benutzt 16 Ziffern. Zusätzlich zu den Ziffern 0 bis 9 werden hier die Buchstaben a bis f als elfte bis sechzehnte Ziffer benutzt. Dieses Zahlensystem erlaubt auch eine relativ einfache Umrechnung von Binärzahlen in Hexadezimalzahlen und umgekehrt. Hierzu werden in der gleichen Weise wie bei den Oktalzahlen die Hexadezimalziffern in vierstellige Binärzahlen und Gruppen zu je vier Binärziffern zu Hexadezimalziffern gewandelt. Im Hexadezimalsystem entspricht ein Byte aus acht Bits genau einer zweistelligen Hexadezimalzahl.

Noch ein Beispiel:

Die Oktalzahl 03_8 entspricht der Binärzahl 011_2 ; die Oktalzahl 06_8 entspricht der Binärzahl 110_2 ; die Oktalzahl 05_8 entspricht der Binärzahl 101_2 . Die Oktalzahl 0365_8 entspricht deshalb der Binärzahl $011\ 110\ 101_2$. Durch einfaches Umgruppieren der Binärzahl läßt sich daraus die entsprechende Hexadezimalzahl errechnen: Die Binärzahl 011110101_2 entspricht der Hexadezimalzahl F5_{16} . Diese Zahl entspricht übrigens der dezimalen 245_{10} .

Zurück zum *od* Kommando:

In der Standardeinstellung ohne Optionen gibt *od* die Position als 7-stellige Oktalzahl und die Daten in 8 Spalten zu je zwei Bytes in oktaler Darstellung aus.

Optionen:

-A *Positionsformat* zeigt die Position des ersten in einer Zeile dargestellten Bytes im Positionsformat; die folgenden Formate stehen zur Auswahl

- d siebenstellige Dezimalzahl
- o siebenstellige Oktalzahl (Voreinstellung)
- x sechsstellige Hexadezimalzahl
- n keine Positionsangabe

-j *Anzahl* überspringt die ersten Anzahl Bytes der Datei und beginnt erst danach mit der Ausgabe; wenn die Zahl mit '0x' oder '0X' beginnt, wird sie als Hexadezimalzahl interpretiert, beginnt sie mit einer Null wird sie als Oktalzahl behandelt, sonst als Dezimalzahl; der Zahl kann einer der Buchstaben *bi* (blocks=512), *k* (kilo=1024) oder *m* (mega=1048576) folgen, die die Anzahl mit den entsprechenden Einheiten multiplizieren

-N *Anzahl* gibt nur die angegebene Anzahl Bytes von der Datei aus; die Anzahl kann wie bei der '-j' Option von einer Einheit gefolgt werden

-t *Format* wählt die Codierung für die Datenausgabe; wenn die -t Option mehrfach benutzt wird, oder mehrere Formate gleichzeitig angegeben werden, gibt *od* für jedes Format jeweils eine entsprechende Zeile aus; folgende Formate werden unterstützt

- a** (ascii) setzt das achte Datenbit aller Zeichen auf null, die druckbaren ASCII Zeichen werden als solche ausgegeben, und die nichtdruckbaren Steuerzeichen werden mit ihren in der ASCII Tabelle verwendeten "Namen" bezeichnet; so wird das Zeilenende als **cr** bezeichnet, ein horizontaler Tabulator mit **tab** und so weiter
- c** (character) gibt die druckbaren ASCII Zeichen als solche aus, die nichtdruckbaren Zeichen werden sofern möglich als Backslashsequenz ausgegeben; **\f** ist hier z. B. ein Zeilenende, **\t** ein Tabulator und so weiter; Bytes die nicht druckbar sind und auch nicht als Backslashsequenz ausgegeben werden können, werden als Oktalzahl dargestellt
- d** (decimal) gibt die Daten als vorzeichenbehaftete Dezimalzahlen aus; voreingestellt sind vier Bytes je Dezimalzahl
- f** (float) gibt die Daten als Fließkommazahlen aus; voreingestellt sind acht Bytes je Fließkommazahl
- o** (octal) gibt die Daten als Oktalzahlen aus; voreingestellt sind vier Bytes je Oktalzahl
- u** (unsigned) gibt die Daten als vorzeichenlose Dezimalzahl aus; voreingestellt sind vier Bytes je Dezimalzahl
- x** (hex) gibt die Daten als Hexadezimalzahlen aus; voreingestellt sind vier Bytes je Hexadezimalzahl

Außer die Typen **a** und **c**, die immer einzelne Bytes anzeigen, kann die Anzahl der Bytes, die in jeweils eine Zahl des bestimmten Typs umgewandelt werden sollen, durch eine dem Typkennzeichner unmittelbar folgende Zahl bestimmt werden. Die Anzahl der Bytes je Zahl kann auch durch Buchstabenkennungen angegeben werden, die den C-Datentypen entsprechen. Für die Ganzzahltypen (**d**, **u**, **x**, **o**) gibt es die folgenden Möglichkeiten:

- C** (Char) ist ein Byte lang
- S** (Short) ist zwei Bytes lang
- I** (Integer) ist vier Bytes lang
- L** (Long) ist auch vier Bytes lang

für Fließkommazahlen können die folgenden Optionen verwendet werden

- F** (Float) ist vier Bytes lang
- D** (Double) ist acht Bytes lang
- L** (Long Double) ist auch acht Bytes lang

- v** gibt auch die doppelten Zeilen aus; normalerweise werden ganze Zeilen, die der zuletzt angezeigten Zeile vollständig entsprechen, durch ein Asterisk '*' dargestellt
- s** [*Länge*] gibt nur die gültigen C-Zeichenketten (Folgen von druckbaren ASCII Zeichen, durch ein Nullbyte beendet) mit mindestens der angegebenen *Länge* aus; Voreinstellung für Länge ist drei Zeichen
- w** [*Anzahl*] setzt die Anzahl der umgewandelten Bytes, die in einer Zeile ausgegeben werden; die Anzahl muß ein vielfaches der Länge jedes Ausgabetyps sein; Voreinstellung ist 16

Die folgenden Optionen übersetzen die nicht dem POSIX Format entsprechenden alten Optionen in die neuen dem POSIX Standard entsprechenden Optionen, wie sie oben aufgeführt sind.

- a** gibt benannte ASCII Zeichen aus, wie **-t a**
- b** gibt die Daten Byteweise als Oktalzahlen aus, wie **-t oC**
- c** gibt druckbare Zeichen oder Backslashsequenzen aus, wie **-t c**
- d** gibt die Daten als vorzeichenlose kurze Dezimalzahlen aus, wie **-t u2**
- f** gibt die Daten als Fließkommazahlen mit vier Bytes je Zahl aus, wie **-t fF**

- h** gibt die Daten als vierstellige Hexadezimalzahl aus, wie `-t xL`
- i** gibt die Daten als vorzeichenbehaftete Dezimalzahl mit zwei Bytes je Zahl aus, wie `-t d2`
- l** gibt die Daten als vorzeichenbehaftete Dezimalzahl mit vier Bytes je Zahl aus, wie `-t dL`
- o** gibt die Daten als Oktalzahlen mit zwei Bytes je Zahl aus, wie `-t oS`
- x** gibt die Daten als Hexadezimalzahlen mit zwei Bytes je Zahl aus, wie `-t x2`

Autor:

Jim Meyering

1.66 passwd

Funktion:

passwd ändert das Paßwort zum System

Syntax:

passwd [*Username*]

Beschreibung:

Die Paßwörter aller Benutzer werden in der Datei `/etc/passwd` gespeichert. Diese Datei ist lesbar aber schreibgeschützt. Um dem Benutzer die Möglichkeit zu geben sein eigenes Paßwort zu ändern läuft **passwd** SUID **root**. Deshalb hat der Anwender zur Laufzeit des Programms Rootprivilegien und darf in die Datei schreiben.

Bei einigen Linuxinstallationen wird das Benutzerpaßwort in einer separaten Datei namens **shadow** gespeichert, um den normalen Benutzern den Lesezugriff auf diese Daten zu verwehren. Die Einzelheiten zu diesem Paßwortsystem sind in den englischen Manualpages beschrieben.

Siehe auch:

chsh auf Seite 46 und **newgrp** auf Seite 119

Autor:

Peter Orbaek

1.67 paste

Funktion:

paste fügt die Zeilen von zwei oder mehr Dateien horizontal zusammen

Syntax:

paste [`-s`] [`-d` *Liste*] [`--serial`] [`--delimiters=`*Liste*] [*Datei* ...]

Beschreibung:

paste fügt die Zeilen mehrerer Dateien zusammen. Die Zeilen werden standardmäßig durch TAB getrennt und die Ausgabe einer kompletten Zeile (das heißt die Ausgabe der entsprechenden Zeilen aller Dateien) wird mit einem Zeilenende abgeschlossen.

Optionen:

- d *Liste*** benutzt die Zeichen aus *Liste* zur Trennung der Zeilen aus den einzelnen Dateien beim zusammenfügen; *Liste* ist dabei ein Wort oder eine Zeichenkette aus beliebigen druckbaren Zeichen oder den Sonderzeichen \n \t \\ und \0 für Zeilenende, Tabulator, Backslash oder Leerstring; wenn die *Liste* abgearbeitet ist, wird sie von vorne angefangen
- s** fügt alle Zeilen einer ganzen Datei zu einer Zeile zusammen; werden mehrere Dateien angegeben, so werden die Zeilen der nächsten Dateien als jeweils eine neue Zeile angefügt

Autor:

David M. Ihnat

1.68 pr**Funktion:**

pr formatiert Textdateien zur Druckerausgabe

Syntax:

pr [+SEITE] [-SPALTEN] [-abcdFmrtv] [-e[*Eintab*[*Schritt*]]] [-h *Kopf*][-i[*Austab*[*Schritt*]]] [-l *Seitenlänge*]
 [-n[*Separator*[*Stellen*]]] [-o *Lrand*] [-s[*Separator*]] [-w *Breite*] [*Datei* ...]

Beschreibung:

pr produziert Seitenweise nummerierte und einfach formatierte Ausgabe von Textdateien.

Optionen:

- +Seite** beginnt die Ausgabe mit der angegebenen *Seite*
- Spalten** setzt den Text in die angegebene Anzahl *Spalten*. Ein Zeilenumbruch findet nicht statt; die Spaltenbreite wird an die Seitenbreite angepaßt
- a** teilt den Text Zeilenweise in die Spalten anstatt Seitenweise
- b** schließt die letzte Seite mit balancierten Spalten ab, das heißt alle Spalten sind gleichlang
- c** zeigt Controlzeichen als Caretsequenz (‘^G’ für Control-G)
- d** gibt den Text mit doppeltem Zeilenabstand aus
- e *Eintab Schritt*** übersetzt das Zeichen *Eintab* in *Schritt* Leerzeichen; Voreinstellungen sind TAB für *Eintab* und 8 für *Schritt*
- f** gibt einen Seitenvorschub anstelle von Zeilenvorschüben am Seitenende
- h *Kopf*** schreibt die Zeichenkette *Kopf* anstelle des Dateinamen als Seitenkopf

- i **Austab Schritt** ersetzt Folgen von *Schritt* Leerzeichen durch ein *Austab* Zeichen; Voreingestellt sind TAB für *Austab* und 8 für *Schritt*
- l **Seitenlänge** setzt die Druckseitenlänge; Voreinstellung ist 66; bei einer Seitenlänge unter 10 Zeilen werden die Kopf und Fußzeilen weggelassen
- m gibt alle Dateien parallel in Spalten aus
- n **Separator Stellen** setzt Zeilennummern vor jede Spalte; werden Dateien parallel angezeigt, bekommt jede Zeile nur eine Nummer; der *Separator* steht zwischen der Nummer und der Zeile, Voreinstellung ist TAB; die Zeilennummer hat die angegebene Anzahl *Stellen*, Voreinstellung ist 5
- o **Lrand** setzt den linken Rand auf *Lrand*; die Seitenbreite ist die Summe von *Lrand* und *Breite* von Option -w
- r gibt keine Fehlermeldung für Dateilesefehler aus
- s **Separator** trennt die Spalten durch den Buchstaben *Separator*; Voreinstellung ist TAB
- t Der 5-Zeilige Kopf und der 5-Zeilige Fußbereich werden nicht ausgegeben, und die Seiten werden nicht durch Leerzeilen oder mit einem Seitenvorschub aufgefüllt
- v gibt nichtdruckbare Sonderzeichen als Oktalzahl aus
- w **Breite** setzt die druckbare *Breite*, Voreinstellung ist 72 Zeichen

Autor:

Pete TerMaat

1.69 printenv

Funktion:

printenv zeigt die Umgebungsvariablen für Programme

Syntax:

printenv [*Variable* ...]

Beschreibung:

printenv zeigt alle oder einzelne Umgebungsvariablen für Programme. Werden *Variable* in der Kommandozeile übergeben, so gibt der Status Null an, daß alle Variablen in der Umgebung gesetzt sind, sonst ist der Status Eins.

Autor:

David MacKenzie und Richard Mlynarik

1.70 ps

Funktion:

ps (process status) zeigt die Prozesse mit ihrem Status an

Syntax:

ps *-acehjlmnrsuvwxSU -ttx Systempfad Swappfad*

Beschreibung:

Mit **ps** lassen sich Daten über die Prozesse in der Prozeßtabelle anzeigen. Die Prozesstabelle wird mit einer Titelzeile ausgegeben. Die Spalten haben folgende Bedeutung:

PRI ist die Priorität eines Prozesses; je niedriger der Wert ist, desto mehr Rechenzeit bekommt der Prozeß

NI ist der Nicewert des Prozesses; Nice erhöht die Priorität des Prozesses und gibt damit Prozessorzeit für andere Prozesse frei

SIZE ist die Größe von Text, Daten und Stack

WCHAN ist der Name der Kernelfunktion in der der Prozeß schläft

STAT ist der Status des Prozesses

R lauffähig

S schlafend

D nicht störbare Schlaf

T angehalten

Z Zombie

W der Prozeß belegt keine Seiten im Arbeitsspeicher

%CPU Anteil an der Prozessorzeit

TT die Nummer des kontrollierenden Terminal

TPGID Gruppen ID des kontrollierenden Terminal

PAGEIN Anzahl der Seitenfehler (das ist der Versuch auf eine ausgelagerte Seite zuzugreifen)

TRS Größe des Textsegments (enthält keine shared Librarys)

DRS Größe des Datensegments (enthält benutzte Libraryseiten)

SWAP ausgelagerte Speicherseiten in Kilobyte (oder Seiten mit -p)

SHRD shared Memory

DT benutzte Libraryseiten in Kilobyte (oder Seiten mit -p)

F Flags

04 Prozeß hat die Mathe Emulation benutzt

10 Der Prozeß wurde verfolgt (traced)

Es gibt zwei unterschiedliche Versionen von **ps**. Das eine arbeitet direkt auf dem Kernelspeicher, aus dem es die Prozesstabelle ausliest. Dazu braucht es die Datei `/etc/psdatabase`, in der die Speicheradressen für die entsprechenden Kernelvariablen abgelegt sind. Diese Datei muß für jeden Kernel mit dem Kommando '**ps -U**' neu erzeugt werden. Bei größeren Veränderungen am Kernel (in der Regel bei neuen Kernelversionen) wird auch ein Neuübersetzen von **ps** notwendig.

Das andere **ps** hat die gleiche Funktionalität und mit Ausnahme der **-U** Option auch die gleichen Optionen, es arbeitet aber mit dem Prozeßdateisystem. Dieses Dateisystem enthält Verzeichnisse für alle Prozesse des Systems, in deren Unterverzeichnissen und Dateien alle für **ps** interessanten Daten zu finden sind. Das **ps** Kommando bereitet diese Daten auf und zeigt sie dem Anwender in genau der gleichen Weise an wie die andere Version. Der Vorteil der Methode mit dem Prozeßdateisystem besteht in der Unabhängigkeit von der Kernelversion.

Das Verzeichnis auf dem das Prozeßdateisystem aufgesetzt ist kann in der aktuellen Version des **procps** nicht angegeben werden. Es erwartet das Prozeßdateisystem unter dem Verzeichnis `'/proc'`.

Optionen:

- a** zeigt die Prozesse aller User
- c** zeigt den Namen des Kommandos
- e** zeigt die Prozeßumgebung
- h** unterdrückt die Kopfzeile
- j** jobs Format: PGID und SID
- l** langes Format: FLAGS WCHAN NICE PRIO
- m** zeigt Speichernutzung
- X** zeigt EIP ESP TIMEOUT und ALARM
- n** gibt numerische Werte für USER und WCHAN
- r** zeigt nur die laufenden Prozesse
- s** zeigt die Signale
- u** zeigt die Besitzer der Prozesse
- v** vm Format
- w** ausführliche Ausgabe, kann mehrmals angegeben werden
- x** zeigt Prozesse, die von keinem Terminal kontrolliert werden
- S** addiert die Prozessorzeit der Kindprozesse zu den Eltern
- U** aktualisiert die Datei /etc/psdatebase, die den Zugang zu den Kerneln vermittelt; diese Aktualisierung muß immer durchgeführt werden, nachdem der Kernel neu übersetzt wurde; diese Option fällt bei dem **ps** Programm das mit dem Prozessdateisystem arbeitet weg
- t xx** zeigt nur die Prozesse die von Terminal *xx* kontrolliert werden

Autor:

Branko Lankester

1.71 pwd

Funktion:

pwd gibt den Namen des aktuellen Verzeichnisses aus

Syntax:

pwd

Beschreibung:

pwd ist in den meisten Shells ein integriertes Kommando. (z. B. in der **bash**, Seite 37)

read

read ist ein eingebautes Shellkommando zum Einlesen von Daten von der Tastatur in ein Shellsript. Siehe dazu die Kommandobeschreibung auf Seite 37.

1.72 **rm**

Funktion:

rm löscht Dateien

Syntax:

rm [-dfirvR] [--directory] [--force] [--interactive] [--recursive] [--verbose] *Pfad* ...

Beschreibung:

rm löscht Dateien. Normalerweise werden die Verzeichnisse nicht mitgelöscht. Wenn eine Datei gelöscht werden soll für die keine Schreibberechtigung besteht, muß der Befehl für diese Datei extra bestätigt werden. In Verzeichnissen, bei denen das Stickybit gesetzt ist, kann eine Datei nur von ihrem Eigentümer gelöscht werden.

Die Option ‘--’ zeigt an daß die folgenden Argumente keine Optionen mehr sind. Dadurch ist es möglich auch Dateinamen die mit einem ‘-’ anfangen zu löschen.

Optionen:

-d löscht Verzeichnisse mit dem ‘unlink’ Systemaufruf, anstelle von **rmdir** (nur für den Superuser **root**); weil die in einem so gelöschten Verzeichnis enthaltenen Dateien nicht mitgelöscht werden ist eine anschließende Reparatur des Dateisystems angesagt

-f keine Nachfragen, keine Fehlermeldungen

-i vor dem Löschen jeder Datei wird nochmal nachgefragt

-r der Inhalt aller Unterverzeichnisse und die Verzeichnisse werden mitgelöscht

-v zeigt die Namen aller Dateien noch ein letztes mal an, bevor sie gelöscht werden

Autor:

Paul Rubin, David MacKenzie und Richard Stallman

1.73 **rmdir**

Funktion:

rmdir löscht Verzeichnisse

Syntax:

rmdir [-p] [--path] *Verzeichnis* ...

Beschreibung:

rmdir löscht leere Verzeichnisse. Es gibt keine Möglichkeit, Verzeichnisse zu löschen, die noch normale Dateien enthalten.

Optionen:

-p löscht mehrere Verzeichnisse rekursiv, wenn alle Verzeichnisse leer sind (nachdem das das Verzeichnis im Verzeichnis gelöscht ist)

Autor:

David MacKenzie

1.74 sed

Funktion:**sed** (stream editor) ist ein Editor zur nicht-interaktiven Textbearbeitung**Syntax:**

```
sed [-nV] [--quiet] [--silent] [--version] [-e Editorkommando] [-f Scriptdatei][--expression=Editorkommando]
[--file=Scriptdatei] [Datei ...]
```

Beschreibung:**sed** ist ein Editor zur automatischen Textbearbeitung.

Die Bearbeitung erfolgt mit Editorkommandos, die dem **sed** in einer separaten *Scriptdatei* oder direkt in der Kommandozeile übergeben werden. Um in der Kommandozeile mehrere *Editorkommandos* zu übergeben, kann die '-e' Option mehrfach verwendet werden. Die Editorkommandos können auch durch ein Semikolon getrennt werden. Wird nur ein einziges Editorkommando in der Kommandozeile übergeben, kann die Kennzeichnung mit der '-e' Option auch weggelassen werden. Damit die Shell keine Veränderungen an der Zeichenkette mit dem Editorkommando vornimmt, muß sie in Hochkommata eingeschlossen werden.

Eine Scriptdatei kann beliebig viele Editorkommandos enthalten, die durch Zeilenende oder Semikolon voneinander getrennt werden müssen.

Jedes Kommando besteht aus einem Adressteil und einem Funktionsteil. Der Adressteil gibt an welche Zeilen einer Textdatei mit diesem Kommando bearbeitet werden sollen, und der Funktionsteil beschreibt die Veränderung, die an den im Adressteil bestimmten Zeilen vorgenommen werden soll. Wenn kein Adressteil angegeben ist, wird die Funktion jedesmal ausgeführt.

Die Bearbeitung eines Textes erfolgt, indem die Eingabe zeilenweise in einen Arbeitsspeicher gelesen wird und dann die Adressteile aller Editorkommandos der Reihe nach mit dem Text im Arbeitsspeicher verglichen werden. Die Funktionen der passenden Kommandos werden in der Reihenfolge ihres Auftretens ausgeführt. Normalerweise wird nach der Bearbeitung aller Kommandos der Inhalt des Arbeitsspeichers auf die Standardausgabe ausgegeben und danach durch die nächste Eingabezeile ersetzt. Die automatische Ausgabe des Arbeitsspeichers nach jeder Zeile kann mit der Option '-n' unterdrückt werden.

Zusätzlich zu dem Arbeitsspeicher gibt es noch einen Zwischenspeicher (Puffer), der von verschiedenen Funktionen benutzt werden kann.

Der Arbeitsspeicher kann auch mehrere Zeilen auf einmal enthalten.

Im Adressteil können die Zeilen entweder durch ihre Zeilennummern oder durch reguläre Ausdrücke ausgewählt werden. Alle Funktionen außer den 'a', 'i', 'q' und '=' akzeptieren einen Adressbereich, bei dem eine Start- und eine Endadresse durch ein Komma getrennt angegeben werden. Ein Dollarzeichen steht für die letzte Zeile. Wenn eine Endadresse mit einem regulären Ausdruck bezeichnet ist, wird die erste passende Zeile als Bereichsende eingesetzt.

Reguläre Ausdrücke müssen in einfachen Schrägstrichen (Slashes '/') eingeschlossen werden. **sed** benutzt die gleichen Routinen zur Auswertung regulärer Ausdrücke wie **emacs** oder **grep** (→ Seite 91). Darüberhinaus kann auch die an die **ed** Syntax angelehnte Konstruktion '\#Muster#' (mit jedem beliebigen Zeichen für '#') benutzt werden, die wie /Muster/ interpretiert wird.

Im *Muster* kann auch ein '\n' vorkommen, das auf das Zeilenende paßt.

Der **sed** kann folgende Funktionen ausführen:

a\Text schreibt den *Text* in die Standardausgabe, bevor die nächste Eingabezeile gelesen wird

- b** *Marke* springt zur der mit der *:Marke* markierten Zeile im Script (nicht im Text) und fährt dort mit dem Programm fort
- c** \ *Text* die im Arbeitsspeicher von **sed** befindliche Zeilen werden gelöscht, und der Text in die Standardausgabe geschrieben; wenn ein Adressbereich angegeben ist, wird der Text erst am Bereichsende einmal ausgegeben
- d** alle aktuell im Arbeitsspeicher von **sed** befindlichen Zeichen werden gelöscht und die nächste Eingabezeile gelesen; die auf diesen Befehl folgenden Befehle werden nicht mehr bearbeitet, auch wenn die Zeilen im Arbeitsspeicher im passenden Bereich liegen
- D** die erste Zeile im Arbeitsspeicher von **sed** wird gelöscht, und die nächste Zeile wird gelesen; die auf diesen Befehl folgenden Befehle werden nicht mehr bearbeitet, auch wenn die Zeilen im Arbeitsspeicher im passenden Bereich liegen
- g** der Arbeitsspeicher von **sed** wird durch den Inhalt des Puffers ersetzt; der Inhalt des Arbeitsspeichers geht dabei verloren
- G** der Pufferinhalt wird an den Inhalt des Arbeitsspeichers angehängt
- h** der Inhalt des Arbeitsspeichers wird in den Puffer geschrieben; der Inhalt des Puffers geht dabei verloren
- H** der Inhalt des Arbeitsspeichers von **sed** wird an den Puffer angehängt
- i** \ *Text* (insert) der Text wird sofort in die Standardausgabe geschrieben
- l** der Inhalt des Arbeitsspeichers von **sed** wird ausgegeben; nichtdruckbare Zeichen werden als Oktalzahl dargestellt
- n** der Inhalt des Arbeitsspeichers wird unverändert in die Ausgabe geschrieben und der Arbeitsspeicher durch die nächste Eingabezeile ersetzt
- N** die nächste Eingabezeile wird an den Arbeitsspeicher angehängt; das Zeilenende wird mit in den Arbeitsspeicher geschrieben; die Zeilennummer des aktuellen Bereiches erhöht sich um eins
- p** der Inhalt des Arbeitsspeichers wird in die Standardausgabe geschrieben
- P** die erste Zeile im Arbeitsspeicher wird in die Standardausgabe geschrieben
- q** beendet **sed**; es werden keine weiteren Befehle ausgeführt und keine Zeilen mehr gelesen
- r** *Datei* der Inhalt der Datei wird ausgegeben, bevor die nächste Zeile gelesen wird
- s** / *Ausdruck* / *Ersetzung* [/ *Modus*] (substitute) ersetzt den (ersten) auf den regulären Ausdruck passenden Text durch den Ersetzungstext; es kann auch ein beliebiges anderes Zeichen anstelle von ‘/’ benutzt werden; als Modus können ein oder mehrere der folgenden angegeben werden
 - n** eine Zahl von 1 bis 512 ersetzt nur das n-te Auftreten des Musters
 - g** (global) alle auf den Ausdruck passenden Textteile werden ersetzt
 - p** wenn eine Ersetzung stattgefunden hat, wird der Inhalt des Arbeitsspeichers von **sed** in die Standardausgabe geschrieben
 - w** *Datei* wenn eine Ersetzung stattgefunden hat, wird der Inhalt des Arbeitsspeichers in die *Datei* geschrieben
- t** *Marke* verzweigt zur mit der *:Marke* versehenen Zeile in der Programmdatei, wenn eine Ersetzung am Inhalt des Arbeitsspeichers vorgenommen wurde, seit die letzte Eingabezeile gelesen wurde, oder seit der letzte **t** Befehl bearbeitet wurde; wenn keine *Marke* angegeben ist, wird an das Ende der Programmdatei verzweigt
- w** *Datei* schreibt den Inhalt des Arbeitsspeichers in die benannte *Datei*

x vertauscht den Inhalt des Puffers mit dem Arbeitsspeicher

y/*Zeichenkette1*/*Zeichenkette2*/ vertauscht jedes auftretenden Zeichen aus der *Zeichenkette1* mit dem entsprechenden Zeichen der *Zeichenkette2*; die beiden Zeichenketten müssen gleich lang sein

!Funktion führt die Funktion für alle Zeilen aus, die NICHT in den Bereich passen

:Marke setzt eine *Marke* für den **b** und den **t** Befehl

{...} die von den Klammern eingeschlossenen und durch Zeilenende oder Semikolon getrennten Funktionen werden als Einheit behandelt

= gibt die aktuelle Eingabezeilennummer aus

leitet einen Kommentar ein; alle folgenden Zeichen bis zum Zeilenende werden ignoriert

Optionen:

-n gibt nur die Zeilen aus, die explizit (durch die Anweisung **p**) ausgedruckt werden sollen

-V gibt die Versionsnummer und eine Kurzhilfe aus

-e *Zeichenkette* wendet die Editorbefehle aus *Zeichenkette* auf den Text an

-f *Datei* liest die Editorbefehle aus der *Datei*

Autor:

Unbekannt

set

set ist eine interne Shellfunktion. Eine Beschreibung ist im Kapitel über die Bash zu auf Seite 38 finden

1.75 sleep

Funktion:

sleep läßt die Zeit verstreichen

Syntax:

sleep *Zeit* [*smhd*] ...

Beschreibung:

sleep wartet, daß die *Zeit* verstreicht. Voreingestellt sind Sekunden, es können aber auch Minuten, Stunden oder Tage sein. Diese Funktion wird vor allem in Shellscripts zur vorübergehenden Anzeige von Informationen benutzt.

Optionen:

s Sekunden

m Minuten

h Stunden

d Tage

Autor:

Free Software Foundation

1.76 sort**Funktion:**

sort sortiert die Zeilen einer Textdatei

Syntax:

sort [-cmus] [-t *Separator*] [-o *Ausgabedatei*] [-bdfiMnr] [+*POS1* [-+sl *POS2*]] [-k *POS1*[,*POS2*]] [*Datei...*]

Beschreibung:

sort wird normalerweise zum Sortieren von Dateien verwendet. Es kann aber auch Dateien daraufhin überprüfen, ob sie sortiert sind; oder mehrere sortierte (oder unsortierte) Dateien zu einer sortierten zusammenfügen.

Dazu existieren drei Modi:

- der **check** Modus, der prüft, ob eine Datei bereits sortiert ist. Dieser Modus wird durch die Option **-c** eingeleitet
- der **merge** Modus, der mehrere vorsortierte Dateien zusammenfügt. Die Dateien so zusammenzufügen ist schneller als sie komplett sortierten zu lassen. Dieser Modus wird durch die Option **-m** eingeleitet
- Der Standardmodus ist **sort**

Wenn Schlüsselfelder bezeichnet sind, vergleicht **sort** die Schlüsselfelder in der Reihenfolge ihrer Bezeichnung bis ein Unterschied gefunden wurde oder keine weiteren Felder vorhanden sind.

Wenn eine der globalen Optionen **Mbdfnr** benutzt wird, und kein Schlüsselfeld angegeben ist, vergleicht **sort** die ganzen Zeilen.

Optionen:

- b** ignoriert führende Leerzeichen
- c** stellt fest, ob die Dateie(en) bereits sortiert sind; wenn eine Datei nicht sortiert ist, wird eine Fehlermeldung ausgegeben, und mit dem Status 1 abgebrochen
- d** sortiert in alphabetischer Reihenfolge
- f** unterscheidet nicht zwischen Groß- und Kleinschreibung
- i** ignoriert alle nichtdruckbaren Zeichen (außerhalb 040-126 ASCII)
- M** sortiert die (amerikanischen) Monate jan feb mar ... dec in der korrekten Reihenfolge; führende Leerzeichen werden wie bei **-b** ignoriert
- m** fügt bereits sortierte Dateien Zeilenweise zusammen
- n** sortiert Zeilen mit Zahlen; ignoriert führende Leerzeichen und behandelt ‘-’ als Vorzeichen
- r** sortiert in umgekehrter Reihenfolge

- o *Datei* schreibt in die *Datei* anstelle der Standardausgabe; wenn eine der Eingabedateien als Ausgabedatei bestimmt wird, legt **sort** erst eine Kopie der Eingabedatei an und sortiert dann in die Ausgabedatei
- t *Separator* benutzt *Separator* als Feldtrenner für die Suchschlüssel; Standard ist der Leerstring zwischen einem Nichtblank und einem Blank; Der Trenner ist nicht Teil eines der getrennten Felder
- u im merge Modus wird nur die erste von einer Reihe gleichwertiger Zeilen ausgegeben; im check Modus wird geprüft, ob nicht zwei Zeilen gleichwertig sind
- +**POS1** [-**POS2**] setzt das Schlüsselfeld auf die Zeichen zwischen *POS1* und *POS2*; Wenn *POS2* fehlt, wird das Zeilenende angenommen; die Felder und Buchstaben zählen von 0
- k **POS1** [-**POS2**] setzt das Schlüsselfeld; wird das Schlüsselfeld so spezifiziert, zählen die Felder und Buchstaben von 1

Eine Position hat die Form *feld.buchstabe*

Autor:

Mike Haertel

1.77 split

Funktion:

split spaltet eine Datei in mehrere kleinere

Syntax:

split [-*Zeilen*] [-l *Zeilen*] [-b *Bytes*[**bkm**]] [-C *Bytes*[**bkm**]] [--lines=*Zeilen*][--bytes=*Bytes*[**bkm**]]
 [--line-bytes=*Bytes*[**bkm**]] [*Datei* [*Prefix*]]

Beschreibung:

split teilt eine Datei in mehrere Teile. Wenn keine weiteren Optionen gegeben sind, wird die *Datei* in Teile zu je 1000 Zeilen aufgeteilt. Die Ausgabe erfolgt in Dateien mit der Endung *Prefix* oder **x** wenn kein Prefix angegeben wird.

Optionen:

-*Zeilen* die Ausgabedateien sind *Zeilen* lang

-l *Zeilen* die Ausgabedateien sind *Zeilen* lang

-b *Bytes* [**bkm**] die Ausgabedateien sind *Bytes* lang; die optionale Endung setzt die Einheit auf

b 512 Byte Blöcke

k 1 Kilobytes (1024) Blöcke

m 1 Megabyte Blöcke

-C *Bytes* [**bkm**] schreibt so viele Zeilen wie möglich in die Ausgabedatei, ohne *Bytes* zu überschreiten; ist eine Zeile länger als *Bytes*, wird die Zeile auf mehrere Dateien aufgeteilt bis der Rest weniger als *Bytes* lang ist; die Optionen **bkm** werden benutzt wie bei -b

Siehe auch:

csplit auf Seite 51

Autor:

tege@sics.se

1.78 stty

Funktion:

stty setzt die Terminalparameter oder zeigt sie an

Syntax:

stty [**-ag**] [**--all**] [**--save**] [*Einstellungen ...*]

Beschreibung:

Wenn **stty** ohne Argumente aufgerufen wird, gibt es die Leitungsgeschwindigkeit und alle Parameter die von der Einstellung **sane** abweichen aus.

Alle Anzeigen und Einstellungen beziehen sich auf die Standardeingabe.

Mit den folgenden Argumenten lassen sich die Eigenschaften des Terminals ändern. Wird einem Argument ein ‘-’ vorangestellt, so wird die entsprechende Eigenschaft abgeschaltet.

Allgemeine Einstellungen:

parenb schickt und erwartet ein Paritätsbit

parodd setzt ungerade Parität (gerade mit **-parodd**)

cs5 cs6 cs7 cs8 setzt die Zeichengröße auf 5, 6, 7 oder 8 Bits

hupcli | **hup** sendet ein SIGHUP, wenn der letzte Prozeß das Terminal schließt

cstopb benutzt zwei Stopbits pro Zeichen (eins mit **-cstopb**)

cread ermöglicht das Lesen von Eingabe

clocal ignoriert Modemkontrollsignale

crtscts ermöglicht RTS/CTS Handshake

Einstellungen für die Terminaleingabe:

ignbrk ignoriert BREAK

brkint BREAK erzeugt ein SIGINT

ignpar ignoriert Paritätsfehler

parmrk markiert Paritätsfehler mit einer 255-0-Zeichen Sequenz

inpck ermöglicht Eingabe-Paritätsprüfung

istrip löscht das 8te Bit der eingehenden Zeichen

inlcr übersetzt Zeilenvorschub in Zeilenvorschub und Wagenrücklauf

igncr ignoriert Wagenrücklauf

icrnl übersetzt Wagenrücklauf in Zeilenvorschub

ixon ermöglicht XON/XOFF Datenflußkontrolle

ixoff | **tandem** ermöglicht das Senden eines Stopzeichens (^S), wenn der Eingabepuffer fast voll ist, und das Senden eines Startzeichens (^Q), wenn er wieder fast leer ist

iucLC übersetzt Großbuchstaben in Kleinbuchstaben

ixany ermöglicht jedes Zeichen als Startzeichen (nur ^Q mit `-ixany`)

imaxbel erlaubt eine akustische Warnung, wenn der Eingabepuffer voll ist; der Puffer wird nicht gelöscht

Einstellungen der Terminalausgabe:

opost Nachbearbeitung der Ausgabe (z. B. TAB in LEERZECHEN)

olcuc übersetzt Kleinbuchstaben in Großbuchstaben

ocrnl übersetzt Wagenrücklauf in Zeilenvorschub

onlcr übersetzt Zeilenvorschub in Zeilenvorschub und Wagenrücklauf

onocr unterdrückt Wagenrücklauf in der ersten Spalte

onlret übersetzt Zeilenvorschub in Wagenrücklauf

ofill sendet Füllzeichen zur Verzögerung

ofdel benutzt Delete Zeichen anstelle von Nullbytes als Füllzeichen

nl1 nl0 Zeilenvorschubverzögerung

cr3 cr2 cr1 cr0 Wagenrücklaufverzögerung

tab3 tab2 tab1 tab0 Tabulatorverzögerung

bs1 bs0 Rückschrittverzögerung

vt1 vt0 vertikale Tabulatorverzögerung

ff1 ff0 Seitenvorschubverzögerung

Lokale Einstellungen:

isig ermöglicht Signalzeichen

icanon ermöglicht Löschrzeichen

ixexten ermöglicht Spezialzeichen außerhalb des POSIX-Standard

echo wiederholt gelesene Zeichen auf der Ausgabe

echoe | **crterase** zeigt Rückschritt als Rückschritt-Leerzeichen-Rückschritt an

echok gibt einen Zeilenvorschub nach einem Killzeichen (^U) aus

echohl wiederholt den Zeilenvorschub, auch wenn andere Zeichen nicht wiederholt werden

noffsh unterdrückt das Löschen des Eingabepuffers nach einer Unterbrechung

xcase ermöglicht die Benutzung von Großbuchstaben, wenn **icanon** gesetzt ist, indem den entsprechenden Kleinbuchstaben ein ‘\’ vorangestellt wird

tostop hält Hintergrundprozesse an, die auf das Terminal schreiben wollen

echoprt | **prterase**

echoctl | **ctlecho** gibt Controlzeichen als Caret-Sequenz aus (^C für Control-C)

echoke | **crtkill**

Kombinationen von Einstellungen:

evenp | **parity** das Gleiche wie **parenb** **—parodd** und **cs7** (mit einem ‘—’ wie **—parenb cs8**)

oddp das Gleiche wie **parenb parodd** und **c7** (mit ‘—’ wie **—parenb cs8**)

nl das Gleiche wie **icrnl** (mit ‘—’ wie **—icrnl —inlcr —igncr**)

ek setzt die Löschezeichen auf ihre voreingestellten Werte

sane setzt alle Einstellungen auf einen Standardwert (nicht unbedingt die gleichen Werte wie beim Einschalten); das Gleiche wie **cread —ignbrk brkint —inlcr —igncr icrnl —ixoff —iucLc —ixany imaxbel opost —olcuc —ocrnl onlcr —onocr —onlret —ofill —ofdel nl0 cr0 tab0 bs0 vt0 ff0 isig icanon iexten echo echoe echok —echohl —noffsh —xcase —tostop —echoprt echoctl echoke**; außerdem werden alle Spezialzeichen auf ihre voreingestellten Werte zurückgesetzt

cooked ermöglicht primitive Editorfunktionen für die Standardeingabe, mit Löschen einzelner Zeichen, Worte oder ganzer Zeilen etc.; die Eingabe wird erst nach einem Zeilenende dem bearbeitenden Programm übergeben; das Gleiche wie **brkint ignpar istrip icrnl ixon opost isig icanon**; außerdem werden die **eof** und **eol** Zeichen auf ihre voreingestellten Werte zurückgesetzt, wenn sie die Gleichen wie die **min** und **time** Zeichen sind; mit dem optionalen ‘—’ das Gleiche wie **raw**

raw setzt die Terminalparameter auf “rohe” Eingabe, jedes Zeichen wird sofort und roh an das bearbeitende Programm weitergegeben; das Gleiche wie **—ignbrk —brkint —ignpar —parmrk —inpck —istrip —inlcr —igncr —icrnl —ixon —ixoff —iucLc —ixany —imaxbel —opost —isig —icanon —xcase min 1 time 0**; mit dem optionalen ‘—’ das Gleiche wie **cooked**

cbreak das Gleiche wie **—icanon**

pass8 das Gleiche wie **—parenb —istrip cs8**; mit ‘—’ das Gleiche wie **parenb istrip cs7**

litout das Gleiche wie **—parenb —istrip —opost cs8**; mit ‘—’ das Gleiche wie **parenb istrip opost cs7**

decctlq das Gleiche wie **—ixany**

tabs das Gleiche wie **tab0**; mit dem optionalen ‘—’ das Gleiche wie **tab3**

lcase | **LCASE** das Gleiche wie **xcase iucLc olcuc**

crt das Gleiche wie **echoe echoctl echoke**

dec das Gleiche wie **echoe echoctl echoke —ixany**; außerdem wird das Spezialzeichen ‘intr’ mit ^C, ‘erase’ mit DEL und ‘kill’ mit ^U belegt

Spezialzeichen:

Die Spezialzeichen können mit der Syntax '*Name* = *Wert*' definiert werden. Der Wert kann entweder als Tastenkombination, als hexadezimale Zahl mit '0x' am Anfang, als oktale Zahl mit '0' am Anfang oder als einfache Dezimale Zahl angegeben werden. Der Wert '^-' oder 'undef' schaltet ein Spezialzeichen ab. Folgende Spezialzeichen werden unterstützt:

intr sendet ein SIGINT (^C)

quit sendet ein SIGQUIT (^\\)

erase löscht das zuletzt eingegebene Zeichen (^?)

kill löscht die aktuelle Zeile (^U)

eof sendet ein Dateienendezeichen (beendet die Eingabe) (^D)

eol Zeilenende (undef)

eol2 alternatives Zeilenende (undef)

start fährt mit einer angehaltenen Ausgabe fort (^Q)

stop hält die Ausgabe an (^S)

susp sendet ein SIGSTOP an ein Terminal (^Z)

rprnt erneuert den Bildschirm (^R)

werase löscht das letzte Wort (^W)

lnext fügt das nächste Zeichen uninterpretiert ein, auch wenn es ein Spezialzeichen ist (^V)

Unabhängig vom **stty** Programm bietet der Terminaltreiber von Linux einen zweiten Zeichensatz an, der unter anderem grafische Zeichen zur Umrandung von Menüs oder Boxen enthält. Zwischen dem normalen und dem Sonderzeichensatz wird mit ^N und ^O umgeschaltet.

Spezielle Einstellungen:

min *N* bestimmt die minimale Zeichenzahl zum Abbrechen eines Lesezyklus im **cbreak** Modus

time *N* bestimmt die Zeit in zehntel Sekunden nach der ein Lesezyklus im **cbreak** Modus automatisch beendet wird, auch wenn die bei 'min' angegebene Anzahl Zeichen noch nicht gelesen ist

ispeed *N* setzt die Eingabegeschwindigkeit auf *N* Zeichen pro Sekunde

ospeed *N* setzt die Ausgabegeschwindigkeit auf *N* Zeichen pro Sekunde

rows *N* zeigt dem Kernel an, daß das Terminal *N* Zeilen auf dem Bildschirm darstellen kann

cols *N* zeigt dem Kernel an, daß das Terminal *N* Spalten darstellen kann

size ist keine Einstellung, sondern gibt die aktuell eingestellte Bildschirmgröße (Zeilen und Spalten) aus

line *N* setzt die Leitungsparameter auf *N*

speed zeigt die Leitungsgeschwindigkeit an

N setzt die Eingabe- und die Ausgabegeschwindigkeit auf *Geschwindigkeit* (in Zeichen pro Sekunde); *Geschwindigkeit* kann dabei einer der folgenden Werte sein
0 50 75 110 134 134.5 150 200 300 600 1200 1800 2400 4800 9600 19200 38400; mit dem Wert 0 kann eine Leitung unterbrochen werden, für die 'local' gesetzt ist

Optionen:

-a zeigt alle Einstellungen in lesbarer Form

-g gibt alle Einstellungen in einer Form, die beim Zurücklesen (als Kommandozeilenargument) die gleichen Einstellungen reproduziert

Autor:

David MacKenzie

1.79 su**Funktion:**

su (superuser) ändert User und Gruppen ID

Syntax:

```
su [-flmp] [-c Befehl] [-s Shell] [--login] [--fast] [--preserve-environment][--command=Befehl]
[--shell=Shell] [-] [ Name [Argument ...]]
```

Beschreibung:

su startet eine neue Shell unter einer neuen Benutzerkennung (UID) und Gruppenkennung (GID). Wie bei einem neuen Login wird das Paßwort des Benutzers abgefragt. Wenn kein Name angegeben wird, ist das Standardargument **root**, man ändert die ID also in die des "Superusers". Allein der Superuser kann mit dem **su** Kommando die Identität jedes beliebigen Users annehmen, ohne nach einem Paßwort gefragt zu werden. Damit (und nur damit) ist es möglich auch unter den Verwaltungsnamen wie *bin*, *news* oder *daemon* zu arbeiten, die normalerweise durch ein Sperrpaßwort gesichert werden.

Das **su** Kommando setzt die Umgebungsvariablen **HOME**, **SHELL**, **USER** und **LOGNAME** entsprechend der Eintragungen in der Datei */etc/passwd*.

Wenn außer dem Namen weitere Argumente in der Kommandozeile angegeben sind, werden sie der Shell übergeben.

Optionen:

-f (fast) startet die Shell mit der Option '-f', die bei der (t)csh das Lesen der Initialisierungsdatei unterdrückt; bei der **bash** wird die Pfadnamenerweiterung unterdrückt, was nicht unbedingt das gewünschte Verhalten ist

-l (login) die Shell wird aufgerufen wie bei ein **login** als *Name*

-p und **-m** (preserve environment) erhält die alte Systemumgebung, das heißt die Umgebungsvariablen **HOME**, **SHELL**, **USER** und **LOGNAME** werden nicht verändert; es wird die in der Umgebungsvariablen **SHELL** bestimmte Shell gestartet, wenn nicht mit der '-s' Option eine andere Shell angegeben ist

-c *Befehl* [*Argument*] führt nur den *Befehl* mit dem *Argument* aus

-s *Shell* startet die *Shell* anstelle der in der Paßwortdatei festgelegten Shell (bzw. */bin/sh*)

Siehe auch:

newgrp auf Seite 119

Autor:

David MacKenzie

1.80 sum

Funktion:

sum berechnet eine Prüfsumme zu einer Datei.

Syntax:

sum [-rs] [-sysv] [*Datei* ...]

Beschreibung:

sum liest eine Datei, oder die Standardeingabe wenn keine Datei bzw. anstelle einer Datei '-' angegeben wurde, und errechnet daraus eine 16 Bit Prüfsumme und die Dateilänge (gerundet in Kilobytes). Wenn diese Prüfsumme festgehalten wird, kann später durch den Vergleich dieser Prüfsumme mit einer neu errechneten eine eventuelle Veränderung dieser Datei festgestellt werden. Wenn die Prüfsummen gleich sind, bedeutet das aber umgekehrt nicht, daß die Dateien mit Sicherheit gleich sind!

Es gibt verschiedene Verfahren zur Prüfsummenberechnung. Das **sum** Kommando benutzt normalerweise das bei BSD Unix übliche Verfahren. Es bietet aber auch die Möglichkeit, einen System 5 kompatiblen Algorithmus zur Prüfsummenberechnung zu verwenden.

Um eine Prüfsumme nach dem neuen, im POSIX-2 Standard festgelegten CRC Verfahren zu berechnen, gibt es das **cksum** Kommando.

Optionen:

-r erzwingt die Berechnung nach dem bei BSD Unix üblichen Verfahren (Voreinstellung)

-s die Prüfsumme wird nach dem für System 5 üblichen Verfahren berechnet; außerdem wird die Dateilänge in Blöcken zu 512 Bytes berechnet und der Dateiname mit ausgegeben

Autor:

Kayvan Aghaiepour und David MacKenzie

Siehe auch:

cksum auf Seite 47

1.81 sync

Funktion:

sync schreibt die gepufferten Blöcke auf die Platte

Syntax:

sync

Beschreibung:

Linux unterhält zur Optimierung der Festplatten und Diskettenzugriffe ein sogenanntes Blockdepot, in dem wie in einem Cache einige Datenblöcke des Massenspeichers im Arbeitsspeicher bereitgehalten werden. Veränderungen an diesen Daten werden zuerst nur im Arbeitsspeicher vorgenommen. Mit dem `sync` Befehl können die veränderten Daten sofort auf den Massenspeicher gesichert werden. Normalerweise wird in regelmäßigen abständen vom `update`-Prozeß ein `sync` Systemaufruf ausgeführt.

Gelegentlich, zum Beispiel vor dem Ausschalten des Rechners ist es sinnvoll ein manuelles `sync` vorzunehmen. Wird der Rechner z. B. infolge eines Defektes ohne Datensynchronisation abgeschaltet, kommt es meist zu Datenverlusten und Fehlern im Dateisystem.

1.82 `tac`

Funktion:

tac wie `cat`, nur umgekehrt

Syntax:

tac [`-br`] [`-s Trenner`] [`--before`] [`--regex`] [`--separator=Trenner`] [*Datei* ...]

Beschreibung:

tac arbeitet in vieler Hinsicht wie `cat`, es werden die einzelnen Felder der *Datei* ausgegeben, allerdings das letzte zuerst. Als Feldtrenner dient das Zeilenende, wenn kein anderer angegeben ist. Der Feldtrenner wird zu dem Feld gezählt, das er abschließt, also an dessen Ende ausgegeben.

Optionen:

- b** (before) der Feldtrenner wird zum darauffolgenden Feld gezählt, also an dessen Anfang ausgegeben.
- r** der Feldtrenner ist ein regulärer Ausdruck
- s *Trenner*** benutzt *Trenner* als Feldtrenner

Siehe auch:

`cat` auf Seite 44

Autor:

Jay Lepreau, David MacKenzie

1.83 `tail`

Funktion:

tail zeigt das Ende einer Datei

Syntax:

tail [`-c` *[+]*N** [*bkm*]] [`-n` *[+]*N**] [`-fqv`] [`--bytes=[+]N [bkm]] [--lines=[+]N] [--follow] [--quiet] [--silent] [--verbose] [Datei ...]
tail [-. [+]N bcmqkv] [Datei ...]`

Beschreibung:

tail druckt die letzten (10) Zeilen einer *Datei* oder von der Standardeingabe wenn keine Datei angegeben wird. Ein einzelnes ‘-’ anstelle eines Dateinamens meint ebenfalls die Standardeingabe. Werden mehrere Dateien angegeben, so wird das Ende jeder Datei mit dem Dateinamen eingeschlossen in ‘==>’ und ‘<==’ eingeleitet.

Optionen:

-c *N* zeigt *N* Bytes vom Ende der *Datei*. Der Anzahl kann eine Einheit folgen. Möglich sind:

b Blöcke mit 512 Bytes

k Blöcke mit Kilobytes

m Blöcke mit Megabytes

-f (follow) gibt immer wieder das Dateiende aus. Dadurch kann die Entwicklung einer wachsenden Datei beobachtet werden. Diese Option funktioniert nur, wenn nur eine einzige Datei angegeben ist.

-l *N* gibt *N* Zeilen aus

-q unterdrückt die Dateinamen zu Beginn der Ausgabe

-v druckt immer die Dateinamen zu Beginn der Ausgabe

Autor:

Paul Rubin, David MacKenzie

1.84 tar

Funktion:

tar (tape archiver) verwaltet Dateiearchive auf Blockdevices

Syntax:

tar [-Acdrux] [+delete] [-b *N*] [-BgGhiklmMoOpPPsSvwWz] [-C *Verzeichnis*] [-f *Datei*] [-F *Datei*] [-K *Datei*] [-L *Länge*] [-N *Datum*] [-T *Datei*] [-V *Name*] [-X *Datei*] [0-7] [{|mh}]

Beschreibung:

tar ist ursprünglich ein Tool zur Verwaltung von Bandarchiven. Das GNU **tar** kann aber auch auf “rohen” Disketten oder in normalen Dateien Archive im **tar** Format anlegen und verwalten. Normalerweise werden Archive mit **tar** nicht komprimiert.

Optionen:

-A hängt ein komplettes Archiv an ein anderes Archiv an

-c erzeugt ein neues Archiv

-d vergleicht das Archiv mit dem Dateisystem

-D *Datei* löscht die *Datei* aus dem Archiv (für Bandarchive nicht empfehlenswert)

-r hängt Dateien an das Archiv an

- t** zeigt den Inhalt des Archivs
- u** ersetzt Dateien die neuer als eine bereits archivierte Version sind. Ist eine Datei noch nicht archivierte, so wird sie eingefügt
- x** kopiert *Datei* oder alle Dateien aus dem Archiv

Weitere Optionen

- b** *N* setzt die Blockgröße auf *N*x512 Bytes (Voreinstellung ist *N*=20)
- B** unterdrückt den Abbruch von **tar** beim Lesen unvollständiger Blöcke; zum Lesen von 4.2BSD Pipes
- C** *Verzeichnis* wechselt während der Ausführung in das *Verzeichnis* um von dort weitere Dateien zu archivieren
- f** *Datei* benutzt *Datei* oder das damit verbundenen Gerät als Archiv
- F** *Datei* bei mehrteiligen Archiven (Option -**M**) wird das Shellsript *Datei* wird ausgeführt, wenn das Medium voll ist
- G** erzeugt einen speziellen Eintrag für jedes archivierte Verzeichnis; spezielles GNU Format (alt)
- g** erzeugt auch einen Eintrag für jedes Verzeichnis, auch spezielles GNU Format (neu)
- h** archiviert die referenzierten Dateien anstelle der Links
- i** ignoriert Blöcke mit Nullbytes im Archiv
- k** existierende Dateien werden nicht überschrieben
- K** *Datei* beginnt eine Aktion bei *Datei* im Archiv
- l** verhindert Archivierung von Dateien aus anderen Dateisystemen
- L** *Länge* wartet auf Medienwechsel nach *Länge* Bytes
- m** das Datum der letzten Änderung wird nicht mit archiviert
- M** das Archiv ist auf mehrere Medien verteilt (Multi-Volume)
- N** *Datum* archiviert nur Dateien die neuer sind als *Datum*
- o** benutzt das alte V7 tar-Format anstelle des ANSI Formates
- O** schreibt die Dateien in die Standardausgabe
- p** erhält die Zugriffsrechte der Dateien
- P** archiviert mit absoluten Dateinamen
- R** gibt zu jeder Meldung die Blocknummer des Archivblocks aus, von dem die Meldung verursacht wurde
- s** zeigt an, daß die Liste von Dateien im Argument die gleiche Reihenfolge hat, wie die Dateien im Archiv
- T** *Datei* holt die Namen der zu archivierenden Dateien aus *Datei*
- v** meldet jede Aktion
- V** *Name* erzeugt ein Archiv mit dem (internen) Namen *Name*
- w** erwartet interaktiv Bestätigung jeder Aktion
- W** verifiziert die geschriebenen Daten im Archiv

-X *Datei* liest aus der *Datei* Namen oder reguläre Ausdrücke von bzw. für Dateien, die nicht archiviert werden sollen

-z erzeugt ein mit **compress** komprimiertes Archiv

-{0..7}{l_mh} spezifiziert das Gerät und die Dichte des Speichermediums (für Diskettenarchive ohne Bedeutung)

Autor:

John Gilmore

1.85 tee

Funktion:

tee verzweigt die Ausgabe auf eine Datei

Syntax:

tee [-ai] [--append] [--ignore-interrupts] [*Datei* ...]

Beschreibung:

tee liest von der Standardeingabe und verzweigt die Ausgabe auf die Standardausgabe und *Datei*. Wird auf eine existierende Datei verzweigt, so wird sie überschrieben, anderenfalls wird sie angelegt.

Die Ausgabe von **make** bzw **gcc** beim kompilieren eines Programms kann beispielsweise mit `'make -k 2>&1 | tee make.out'` (bash-Syntax) in der Datei `make.out` gesichert werden.

Optionen:

-a die *Datei* wird nicht überschrieben, sondern die Ausgabe daran angehängt

-i ignoriert Interrupt Signale

Autor:

Mike Parker, Richard M. Stallman und David MacKenzie

test

test bewertet einen Ausdruck, und gibt den Wahrheitswert als Status zurück. Das **test** Kommando ist idenisch mit dem **test** Shellkommando, das auf Seite 40 des Handbuchs beschrieben ist.

1.86 touch

Funktion:

touch ändert die Zeitmarkierung einer Datei

Syntax:

touch *[-acm] [-r Referenzdatei] [-t MMDDhhmm[[CC]YY][.ss]] [-d Zeit][--time={atime, access, use, mtime, modify}] [--date=Zeit] [--file=Referenzdatei] [--no-create] Datei ...*

Beschreibung:

touch setzt die Zugriffs- und die Änderungszeit der *Datei* auf die aktuelle Zeit. Wenn die *Datei* nicht existiert, wird sie erzeugt.

Wenn als erster Dateiname ein gültiges Argument für die *-t* Option übergeben wird, und keine der Optionen *-d*, *-r* oder *-t* ausdrücklich angegeben ist, wird dieses Argument als Zeitmarke für die folgenden Dateien verwendet.

Optionen:

-a ändert nur die Zugriffszeit

-c unterdrückt die Erzeugung nicht existierender Dateien

-d Zeit setzt *Zeit* anstelle der aktuellen Uhrzeit. Für *zeit* können verschiedene gebräuchliche Formate verwendet werden.

-m ändert nur die Änderungszeit

-r Referenzdatei setzt die Zeit von *Referenzdatei* anstelle der aktuellen Zeit

-t MMDDhhmm [[CC]YY][.ss] benutzt das Argument als Zeitangabe

Autor:

Paul Rubin, Arnold Robbins, Jim Kingdon, David MacKenzie, und Randy Smith

trap

trap ist ein eingebautes Shellkommando, das bestimmte Signale abfangen kann und dararaufhin benutzerdefinierte Aktionen ausführt. Die Beschreibung ist auf Seite 41 des Handbuchs zu finden.

1.87 tty**Funktion:**

tty gibt die Bezeichnung des aktuellen Terminals aus

Syntax:

tty *[-s] [--silent] [--quiet]*

Beschreibung:

tty gibt den Namen des Terminals inclusive Pfad aus, das die Standardeingabe repräsentiert. Der Status wird dabei wie folgt gesetzt:

0 wenn die Standardeingabe ein Terminal ist

1 wenn die Standardeingabe nicht ein Terminal ist

2 wenn ein anderer Fehler aufgetreten ist

Optionen:

-s keine Ausgabe. Nur der Status wird gesetzt

Autor:

David MacKenzie

umask

umask ist eine eingebaute Shellfunktion. Eine Beschreibung ist im Abschnitt über die **bash** auf Seite 42 zu finden.

1.88 uname**Funktion:**

uname gibt Auskunft über Betriebssystem und Hardware

Syntax:

uname [**-snrvma**] [**--sysname**] [**--nodename**] [**--release**] [**--version**] [**--machine**] [**--all**]

Optionen:

-s zeigt den Betriebssystemnamen (Voreinstellung wenn keine Option angegeben wird)

-n zeigt den Netzwerknamen des Systems

-r zeigt die Release (Versionsnummer) des Betriebssystems

-v zeigt das Erstellungsdatum des Betriebssystems

-m zeigt den Prozessortyp

-a zeigt alle der oben genannten Daten

Autor:

David MacKenzie

1.89 uniq**Funktion:**

uniq löscht doppelte Zeilen aus einer sortierten Datei

Syntax:

uniq [**-cd**] [**-f Nummer**] [**-s Nummer**] [**-w Nummer**] [**-# Nummer**] [**+# Nummer**] [**--count**] [**--repeated**] [**--unique**] [**--skip-fields=Nummer**][**--skip-chars=Nummer**] [**--check-chars=Nummer**] [*Eingabedatei*] [*Ausgabedatei*]

Optionen:

- u** gibt nur die einzigartigen Zeilen aus, die keine gleichen Nachbarzeilen haben.
- d** gibt nur die Zeilen mit Doppelgängern aus
- c** gibt am Anfang jeder Zeile mit Doppelgänger die Anzahl des Vorkommens aus
- f Nummer** überspringt *Nummer* Felder beim Vergleich. Felder sind durch Leerzeichen und TAB getrennt. Führende Leerzeichen werden ignoriert.
- s Nummer** überspringt *Nummer* Zeichen beim Vergleich. Führende Leerzeichen werden ignoriert. Werden Felder und Zeichen übersprungen, so werden zuerst die Felder und dann in dem erreichten Feld die Zeichen übersprungen.
- w Nummer** es werden nur *Nummer* Zeichen verglichen. Normalerweise wird die komplette Zeile (oder der Zeilenrest) verglichen

Autor:

Richard Stallman und David MacKenzie

vdir

vdir arbeitet wie 'ls -l'. **vdir** ist nicht mehr ein Link auf **ls**, sondern ein eigenes Kommando. Die Eigenschaft dieses Kommandos ändert sich durch Umbenennung nicht.

Die Verwendung und die Optionen entsprechen denen von **ls**. Eine Beschreibung ist auf Seite 109 nachzulesen.

vi

vi ist ein Link auf **elvis**, den freien vi-Clone. Die Beschreibung ist ab Seite 60 zu finden.

view

view ist ein Link auf **elvis**. Unter diesem Namen werden die Dateien nur zum Lesen geöffnet, können also nicht verändert werden. Die Bedienung von **view** ist ab Seite 60 des Handbuchs beschrieben.

wait

wait ist ein eingebautes Shellkommando, mit dem die Shell veranlaßt werden kann, auf ein bestimmtes Signal zu warten. Eine Beschreibung ist auf Seite 43 des Handbuchs zu finden.

1.90 wc**Funktion:**

wc zählt die Anzahl von Zeichen, Worten oder Zeilen

Syntax:

wc [-c|w] [--bytes] [--chars] [--lines] [--words] [*Datei* ...]

Beschreibung:

wc zählt in jeder *Datei* oder in der Standardeingabe die Zeilen, die Worte und die Zeichen. Für jede Datei wird eine Zeile mit den Zahlen gefolgt vom Dateinamen ausgegeben. Wird mehr als eine Datei angegeben, wird zusätzlich die Summe der einzelnen Werte in der letzten Zeile ausgegeben.

Ohne weitere Optionen gibt wc alle drei Werte aus

Optionen:

-l gibt nur die Anzahl der Zeilen aus

-w gibt nur die Anzahl der Worte aus

-c gibt nur die Anzahl der Zeichen aus

1.91 who

Funktion:

who gibt Information über die aktiven Anwender des Systems

Syntax:

who [**-imqsuwHT**] [*Datei*] [**am i**]

Beschreibung:

who zeigt die Namen, das Terminal und die Loginzeit der aktiven Benutzer. Außerdem gibt es Antwort auf die existentielle Frage **“wer bin ich?”**!

Wird eine *Datei* angegeben, so wird die Information über die aktiven Anwender aus dieser Datei genommen. Voreingestellt ist die Datei `/etc/utmp`

Optionen:

-i zeigt die Dauer der aktuellen Sitzungen

-m gibt den eigenen Benutzernamen mit E-mail Adresse aus

-q gibt nur die Benutzernamen und die Gesamtzahl der aktiven Benutzer aus

-s Ausgabe im Standardformat

-u wie **-i**

-w markiert alle Benutzer mit `+` die eine Nachricht empfangen können, oder mit einem `-` wenn der Empfang von Nachrichten abgeschaltet wurde

-H gibt zusätzlich eine Kopfzeile mit der Bedeutung der einzelnen Spalten aus

-T wie **-w**

Autor:

jla, überarbeitet von David MacKenzie

1.92 write

Funktion:

write schreibt eine Nachricht auf das Terminal eines anderen Benutzers

Syntax:

write *Name* ...

Beschreibung:

write schreibt auf das Terminal von *Name*. Die Nachricht wird von der Standardeingabe gelesen. Wenn von der Tastatur gelesen wird, muß die Eingabe mit CONTROL-D abgeschlossen werden. Mit dem Kommando 'mesg no' kann das eigene Terminal vor fremdem Beschreiben geschützt werden.

Wenn das Kommando unter dem Namen **wall** aufgerufen wird, schreibt es an alle eingeloggten Benutzer. Der Superuser (root) kann auch an die Anwender schreiben, deren Terminal vor dem Beschreiben geschützt ist.

Autor:

Peter Orbaek

Kapitel 2

Die Kommandos für root

2.1 chown

Funktion:

chown (change owner) ändert den Eigentümer der Datei(en)

Syntax:

chown [-Rcfv] [--recursive] [--show-changes] [--silent] [--quiet] [--verbose] [*Besitzer*][*..*][*Gruppe*] *Datei*
...

Beschreibung:

Normalerweise ist der Erzeuger einer Datei bzw. eines Verzeichnisses auch deren/dessen Eigentümer. Die besondere Bedeutung der Eigentumsrechte ist auf Seite 184 erklärt. Mit **chown** kann der Superuser (root) den Eigentümer und die Gruppe einer Datei oder eines Verzeichnisses ändern. *Besitzer* und *Gruppe* können als Name oder Kennzahl angegeben werden. Name und Zahl müssen mit den entsprechenden Einträgen in */etc/passwd* und */etc/group* übereinstimmen.

Optionen:

- c** (changes) es werden nur die Dateien angezeigt, deren Besitzer tatsächlich verändert werden
- f** (force) Fehlermeldungen wegen fehlgeschlagener Änderungsversuche werden unterdrückt
- v** (verbose) alle Aktionen werden angezeigt
- R** (recursive) die Besitzer aller Dateien in den Unterverzeichnissen werden ebenfalls geändert

Siehe auch:

chgrp auf Seite 44 und chmod auf Seite 45

Autor:

David MacKenzie

2.2 fdisk

Funktion:

fdisk ist der Partitionstablenneditor für Linux

Syntax:

fdisk [-v] [*Gerätedatei*]

Beschreibung:

fdisk teilt eine formatierte Festplatte in verschiedene Partitionen ein, löscht bestehende Partitionen oder gibt vorhandenen Partitionen eine andere Systemkennung.

Mit **fdisk** wird nicht eine einzelne Partition, sondern die ganze (rohe) Festplatte bearbeitet. Die Gerätedatei ist also `/dev/hda` für die erste ‘normale’ Festplatte, `/dev/hdb` für die zweite ‘normale’ Festplatte, `/dev/sda` für die erste SCSI Festplatte, `/dev/sdb` für die zweite SCSI Festplatte usw...

Als normale Festplatten werden von Linux sowohl AT-Bus Platten, als auch Festplatten für einfache RLL oder MFM Controller unterstützt.

Optionen:

-v gibt die Versionsnummer aus

Siehe auch:

Eine ausführliche Beschreibung des **fdisk** Kommandos ist im Kapitel “Die Festplatte partitionieren” ab Seite 191 zu finden.

Autor:

A. V. Le Blanc

2.3 fsck und efsck

Funktion:

fsck und **efsck** prüfen die Konsistenz des Dateisystems

Syntax:

(e)fsck [-larvsm] */dev/Name*

Beschreibung:

Die Programme **fsck** und **efsck** haben die gleiche Aufgabe, und werden mit den gleichen Kommandozeilenoptionen aufgerufen. Der Unterschied zwischen den beiden Programmen besteht darin, daß **fsck** für das ‘alte’ Minix Dateisystem konfiguriert ist, während **efsck** für das erweiterte (extended) Dateisystem angepasst wurde.

(e)fsck überprüft das Dateisystem auf Blocks, die belegt, aber keiner Datei zugeordnet sind, und auf korrekte Vernetzung der Inodes. Wenn ein Fehler festgestellt wird, kann er mit **(e)fsck** auch repariert werden.

Fehler im Dateisystem entstehen leicht durch unkontrolliertes Abschalten des Rechners. Dabei gehen die letzten Veränderungen der geöffneten Dateien und des Superblocks verloren, die aus dem Blockdepot nur

bei einem **sync** auf die Festplatte geschrieben werden. (**e**)**fsck** kann die Integrität des Dateisystems meistens wieder herstellen, allerdings gehen dabei die Daten der betroffenen Dateien verloren. Damit die Veränderungen am Dateisystem auch tatsächlich wirksam werden, darf (**e**)**fsck** mit den Optionen **-a** und **-r** nur auf abgesetzte Dateisysteme angewandt werden. Anderenfalls würde der reparierte Superblock bei der nächsten Synchronisierung wieder durch den Defekten aus dem Blockdepot ersetzt werden.

Optionen:

- l** (list) zeigt die Dateien und Verzeichnisse auf dem Device (Partition einer Festplatte oder Diskette)
- a** (automatic) repariert alle gefundenen Fehler automatisch
- r** (repair) repariert gefundene Fehler einzeln nach Rückfrage
- v** (verbose) gibt Information über das Dateisystem
- s** (superblock) zeigt den Superblock des Dateisystems
- m** (mode) wenn ich das wüßte ...

Autor:

Linus Torvalds

2.4 mkfs

Funktion:

mkfs erzeugt ein neues Dateisystem im Minix Format auf einem formatierten Datenträger

Syntax:

mkfs [**-c** | **-l** *Datei*] **/dev/*Gerätedatei*** *Blocks*

Beschreibung:

Mit Hilfe von **mkfs** werden auf leeren, formatierten Datenträgern (Disketten oder Festplatten) verschiedene notwendige Verwaltungsinformationen eingetragen. Das Format des von **mkfs** angelegten Dateisystems ist identisch mit dem ausgereiften Minix Dateisystem. Deshalb ist die Typbezeichnung weiterhin 'minix'.

Die *Gerätedatei* ist der Eintrag der entsprechenden Partition im Dateisystem. Der Name setzt sich aus der Bezeichnung für die gesamte (rohe) Festplatte (wie er beim **fdisk** Kommando auf Seite 152 beschrieben ist) und der Partitionsnummer zusammen. Die zweite Partition der erste AT-Bus Platte wird beispielsweise als **/dev/hda2** angesprochen; die erste Partition der zweiten SCSI Festplatte heißt entsprechend **/dev/sdb1**.

Mit *Blocks* wird die Größe des Dateisystems, also die Kapazität der Festplattenpartition oder der Diskette angegeben. Die Anzahl der Blocks ist genau doppelt so hoch wie die Partitionsgröße in Kilobyte. Die genaue Anzahl kann für 'normale' Festplatten im **fdisk** Kommando mit dem **p** Befehl angezeigt werden. Für SCSI Festplatten sollte dieser Wert unter MS-DOS mit dem DOS-**fdisk** Kommando ermittelt werden.

Das **mkfs** Kommando legt dann einen entsprechend großen Superblock und eine Anzahl Inodes an. Das Verhältnis von Dateisystemgröße zur Anzahl der Inodes kann bei **mkfs** nicht verändert werden.

Mit dem Minix Dateisystem können Partitionen bis zu einer Größe von maximal 64 Megabyte verwaltet werden. Die Dateinamen in diesem Dateisystem können bis zu 14 Zeichen lang sein. Eine Unterteilung des Dateinamens in einen Stamm und eine Erweiterung, wie sie z. B. bei MS-DOS erzwungen wird, gibt es nicht. Es steht dem Benutzer frei, beliebig viele Punkte in einem Dateinamen zu verwenden.

Optionen:

-c überprüft das Medium auf defekte Blocks; jeder Block wird einmal beschrieben, wieder gelesen und verglichen; fehlerhafte Blocks werden automatisch von einer `.badblocks` Datei belegt

-l *Datei* liest die Nummern der defekten Blocks aus der *Datei*

Autor:

Linus Torvalds

2.5 mkfs

Funktion:

mkfs erzeugt ein neues erweitertes (extended) Dateisystem auf einem formatierten Datenträger

Syntax:

mkfs [**-c** | **-l *Datei***] [**-i *Anzahl***] **/dev/*Gerätedatei* *Blocks***

Beschreibung:

mkfs arbeitet im Prinzip wie **mkfs**. Der Unterschied besteht vor allem darin, daß die von **mkfs** angelegten Dateisysteme Dateinamen mit einer Länge von bis zu 255 Zeichen zulassen. Die maximale Größe einer Partition ist 4 Terabytes (4000 Gigabytes, 4000 000 000 Blöcke).

Achtung! Das erweiterte Dateisystem ist erst im Teststadium. Die Entwicklung eines zuverlässigen Dateisystems ist eine langwierige und schwierige Aufgabe, vor allem solange auch die anderen Funktionen des Kernels einer dauernden Entwicklung unterworfen sind. Deshalb ist das erweiterte Dateisystem offiziell noch immer in der α -Testphase. Es sind sicher unentdeckte Fehler darin enthalten, und ebenso sicher werden in zukünftigen Versionen Veränderungen daran vorgenommen. Wer also großen Wert auf Datensicherheit in seinem Dateisystem legt, sollte sich mit den Möglichkeiten des Minix Dateisystems begnügen.

Optionen:

-c überprüft das Device auf defekte Blocks

-l *Datei* liest die Nummern der defekten Blocks aus der *Datei*

-i *Anzahl* setzt das Inode/Datenblock Verhältnis fest; jeder Inode wird die angegebene *Anzahl* (mindestens 1024) Bytes Daten zugeordnet

Autor:

Linus Torvalds und Remy Card

2.6 mknod

Funktion:

mknod erzeugt eine Spezialdatei

Syntax:

mknod [-m *Modus*] [--mode=*Modus*] *Name* {bcu} *Major Minor*

mknod [-m *Modus*] [--mode=*Modus*] *Pfad* *p*

Beschreibung:

mknod erzeugt ein FIFO, eine Gerätedatei für ein zeichenorientiertes Gerät (character device) oder für ein blockorientiertes Gerät (block-device) mit dem angegebenen *Namen*.

Die Gerätedateien werden über die Major Device Nummern mit den entsprechenden Gerätetreibern im Kernel verbunden. Mehrere Geräte der gleichen Art werden vom Gerätetreiber durch die Minor Device Nummern unterschieden.

Die Major Device Nummern sind folgendermaßen belegt:

- 0** wird vom Prozeßdateisystem und vom NFS benutzt
- 1** der Arbeitsspeicher (RAM)
- 2** die Diskettenlaufwerke
- 3** die 'normalen' Festplatten (AT-Bus, MFM, RLL)
- 4** die virtuellen Terminals und die seriellen Schnittstellen (seit 0.99.5 speziell für login-Ports)
- 5** der Terminaltreiber im Kernel und seit Version 0.99.5 die seriellen Ports zum rauswählen
- 6** die parallelen Ports (Drucker)
- 7** named Pipes
- 8** SCSI Festplatte
- 9** (SCSI) Bandlaufwerk
- 10** Busmaus
- 11** (SCSI) CD-ROM
- 12** QIC Tape
- 13** XT Festplatte mit 8-Bit Controller
- 14** Soundkarte (?)
- 15** nicht belegt

Die Minor Device Nummern sind gerätespezifisch. Bei den Floppylaufwerken werden damit z. B. neben den zwei möglichen physikalischen Laufwerken auch die Diskettenformate unterschieden.

Die Zugriffsrechte auf die Datei werden aus der Bitdifferenz von 0666 und der Umgebungsvariablen `umask` gebildet.

Der erste Buchstabe nach dem *Namen* gibt den Typ der Datei an:

p erzeugt eine FIFO Spezialdatei (wie `mkfifo`, → Seite 114)

b erzeugt eine Gerätedatei für ein blockorientiertes Gerät

c erzeugt eine ungepufferte Gerätedatei für ein zeichenorientiertes Gerät

u das Gleiche wie **c**

Optionen:

-m *Modus* setzt die Rechte der Dateien auf *Modus* wie bei `chmod` (→ Seite 45)

Autor:

David MacKenzie

2.7 mkswap

Funktion:

mkswap bereitet eine (Geräte)Datei für die Auslagerung von Prozessen vor

Syntax:

mkswap [*-c*] *Datei Blocks*

Beschreibung:

In einem multitasking Betriebssystem wie Linux “schlafen” fast immer einige Prozesse, weil sie auf das Ende eines anderen Prozesses warten. Daher ist es leicht möglich bei großer Systembelastung schlafende Prozesse auf Festplatte auszulagern, um dadurch Platz im Arbeitsspeicher für neue Prozesse zu erhalten. Damit eine Partition entsprechend verwaltet werden kann, muß sie mit **mkswap** als Swapbereich eingerichtet werden. */dev/name* ist die Gerätedatei zu der Partition auf der der Swapbereich angelegt werden soll, und *Blocks* ist die Größe in kilobyte Blöcken. Um eine Swappartition zu aktivieren muß das Kommando **swapon** *Swappartition* aufgerufen werden.

Es ist auch möglich für kurzfristigen Speichermehrbedarf eine Swapdatei einzurichten. Dazu wird mit dem **dd** Kommando eine leere Datei der benötigten Größe erzeugt, und diese Datei dann mit dem **mkswap** Kommando vorbereitet.

Beispiel:

```
dd bs=1024 if=/dev/zero of=/dev/swapfile count=5120
mkswap -c /dev/swapfile 5120
swapon /dev/swapfile
```

Diese Kommandofolge erzeugt eine Swapdatei von 5 Megabyte im Verzeichnis */dev*.

Der Swapper von Linux benutzt **paging**. Das heißt, es werden nicht sofort die kompletten Prozesse ausgelagert, sondern nur so viele Speicherseiten, wie ein anderer Prozeß gerade anfordert.

Optionen:

-c überprüft die Partition auf defekte Blocks

Autor:

Linus Torvalds

2.8 mount

Funktion:

mount setzt das Dateisystem zusammen

Syntax:

mount [-afrwuv] [-t minix | ext | msdos]

mount [-frwuv] *Gerätedatei* | *Verzeichnis*

mount [-frwu] [-t minix | ext | msdos | proc] [-o *Optionen*] *Gerätedatei* *Verzeichnis*

Beschreibung:

mount fügt die einzelnen Dateisysteme auf den verschiedenen Massenspeichern zu einem einzigen Dateisystembaum zusammen. Linux unterstützt drei verschiedene Arten von Dateisystemen:

minix ist das "originale" FS das von A. Tanenbaum's MINIX übernommen wurde. Die Dateinamen sind auf 14 Zeichen begrenzt; eine Partition kann maximal 64MB groß sein.

ext ist das neue erweiterte FS für Linux. Die Dateinamen können 256 Zeichen lang sein. Die maximale Größe für eine Partition liegt bei 4TB.

msdos ist das Feilsystem für Dosen.

proc ist das Prozeßdateisystem. Es wird direkt vom Kernel angeboten, und enthält Verzeichnisse für alle Prozesse und weitere Dateien mit Daten über den Systemzustand. Das Prozeßdateisystem wird durch keine Gerätedatei abgebildet. Damit das **mount** Kommando korrekt arbeitet, muß anstelle einer Gerätedatei das Schlüsselwort **none** angegeben werden. Als Verzeichnis zum Aufsetzen des Prozeßdateisystems empfiehlt sich '/proc'. In diesem Verzeichnis sucht z. B. das neue **ps** Kommando danach.

Die einzelnen Partitionen und Laufwerke sind als Gerätedateien (special-files) im Ordner /dev abgebildet. Die zu den Partitionen gehörende Dateisysteme werden durch den **mount** Befehl auf beliebige leere Verzeichnisse aufgesetzt.

mount erstellt eine Liste der aufgesetzten Dateisysteme in /etc/mtab. Wenn ein **mount** Befehl unvollständig angegeben wird, werden die fehlenden Parameter automatisch aus der Datei /etc/fstab ergänzt (→ Seite 163). Wird **mount** ganz ohne Parameter aufgerufen, werden die Einträge aus /etc/mtab angezeigt.

Optionen:

-f (fake) führt alle Schritte des Befehls mit Ausnahme des eigentlichen Systemaufrufs aus; diese Option ist sinnvoll im Zusammenhang mit der **-v** Option

-o optionen ... erlaubt folgende *Optionen*:

noexec verbietet die Ausführung von Programmen aus diesem Dateisystem

nosuid unterdrückt die Wirkung der SUID und SGID-Bits

nodev die zeichenorientierten- und blockorientierten Gerätedateien werden nicht angesprochen

synchronous alle Lese/Schreibaktionen werden ungepuffert ausgeführt

-r in diesem Dateisystem ist nur Lesen zugelassen

-t {minix, ext, msdos, proc} spezifiziert den Typ der Dateisysteme die aufgesetzt werden sollen. Die Verneinung ist durch ein vorangestelltes **no** möglich. Beispielsweise **mount -a -t nomsdos,ext** setzt alle Dateisysteme die nicht vom Typ msdos oder ext sind auf.

-u (update) verändert den Status des bereits zusammengesetzten Dateisystembaums.

-v schreibt allerhand Zeugs auf den Bildschirm

-w erlaubt das schreiben auf dem Dateisystem

-a alle Dateisysteme werden wie in /etc/fstab beschrieben zusammengefügt

Autor:

Doug Quale

2.9 rdev

Funktion:**rdev** (root device) zeigt und verändert einige Kernelkonfigurationen**Syntax:**

```
rdev [-help] [-o Image [Offset]] [-r Image [Größe]] [-s Image [Gerätedatei]] [-v Image [Modus]] [Image [Gerätedatei]]
```

Beschreibung:

rdev zeigt und verändert einige der Kernelkonfigurationen. Der Name ist von “root device” abgeleitet, und deutet auf die Hauptaufgabe des Programms hin, nämlich die Festplattenpartition mit dem Rootfilesytem neu festzulegen, ohne den Kernel neu zu übersetzen. Die Rootpartition muß im Kernel selbst gespeichert sein. Es gibt noch weitere Kernelparameter, die normalerweise vor dem Kompilieren eines neuen Kernels im Makefile gesetzt werden. Das **rdev** Kommando bietet dem Systemverwalter die Möglichkeit einige dieser Parameter auf der Bootdiskette oder an der fertigen Kerneldatei zu verändern, bevor diese auf eine Bootdiskette geschrieben wird. Wenn der Kernel mit einem Loader direkt von der Festplatte geladen wird, kann natürlich diese Kerneldatei genauso verändert werden.

Um eine Bootdiskette zu erzeugen, muß die Kerneldatei auf eine roh formatierte Diskette geschrieben werden. Dazu reicht es aus, sie mit dem **cp** Programm auf die Gerätedatei für das Diskettenlaufwerk zu kopieren. Das **dd** Kommando kann ebenfalls benutzt werden.

Die Kerneldatei heißt normalerweise **Image** und befindet sich nach einem erfolgreichen Compilerdurchlauf im Verzeichnis `/usr/src/linux`. Bei kompletten Diskettendistributionen ist die Image Datei meistens im Wurzelverzeichnis zu finden.

Das **rdev** Kommando kann auch auf der fertigen Bootdiskette arbeiten, wobei anstelle der **Image** Datei die Gerätedatei des Diskettenlaufwerkes angegeben werden muß.

Optionen:

-s Image [Gerätedatei] legt die Swappartition im Kernel fest (wird seit der Kernelversion 0.98.3 nicht mehr unterstützt)

-v Image [Modus] zeigt oder setzt den Videomodus beim Systemstart; die folgenden Modi werden unterstützt

-3 der Videomodus wird interaktiv erfragt (Voreinstellung)

-2 der erweiterte VGA Modus mit 80x50 Zeichen

-1 der standard VGA Modus mit 80x25 Zeichen

1 der erste unterstützte Videomodus

2 der zweite unterstützte Videomodus (und so weiter)

die unterstützten Videomodi hängen von der VGA Karte ab

-r Image [Größe] zeigt oder setzt die Ramdiskgröße auf die angegebene *Größe*

-o Image [Offset] zeigt oder setzt den **Offset** (Anzahl Bytes vom Diskettenanfang) bei dem das erste Byte der Ramdisk auf der Bootdiskette gesucht wird

2.10 shutdown

Funktion:

shutdown fährt das System herunter

Syntax:

shutdown *[-h | -r] [-fqs] [now|hh:ss|+Minuten]*

Beschreibung:

shutdown sorgt für ein geregeltes Ende des Betriebes. Alle Prozesse werden beendet, das Dateisystem demontiert und das System schließlich angehalten. Wenn keine Zeit angegeben ist, wird das System nach zwei Minuten heruntergefahren. Das shutdown Kommando legt automatisch die Datei /etc/nologin an, die vom login Ausgewertet wird und das einloggen normaler Anwender während des shutdown verhindert.

Es gibt eine Reihe von Links auf **shutdown**, die jeweils verschiedene Optionen voreingestellt haben. Alle Links fahren das System sofort herunter.

halt hat die Option *'-h'* gesetzt

reboot hat die Option *'-r'* gesetzt

fasthalt hat die Optionen *'-h'* und *'-f'* gesetzt

fastboot hat die Optionen *'-r'* und *'-f'* gesetzt

Die Datei *'/fastboot'* wird von einigen *'init'* Versionen ausgewertet. Wenn diese Datei existiert, wird ein **fsck** während des Hochfahrens unterdrückt.

Optionen:

-h hält das System an, ohne neu zu booten

-r startet das System nach dem Herunterfahren sofort neu

-f erzeugt die Datei /fastboot

-q unterdrückt die Frage nach einer Begründung (Meldung für **wall**)

-s fährt das System nur in den Einbenutzerbetrieb zurück

now fährt das System sofort herunter

hh:mm fährt das System zu der angegebenen Uhrzeit herunter

+Minuten fährt das System nach Ablauf der Minuten herunter

Autor:

Peter Orbaek

2.11 umount

Funktion:

umount setzt ein aufgesetztes Dateisystem ab

Syntax:

umount [-a] [-t minix | ext | msdos]

umount *Gerätedatei* | *Verzeichnis*

Beschreibung:

umount setzt das aufgesetzte Dateisystem des mit der *Gerätedatei* verbundenen Gerätes von dem *Verzeichnis* ab.

Beispielsweise eine Diskette mit Dateisystem muß erst auf diese Weise abgesetzt werden, bevor sie aus dem Laufwerk genommen werden kann. **umount** führt automatisch einen **sync** Befehl zur Sicherung aller Daten aus dem Blockdepot aus. Weil das eine Weile dauern kann ist es von großer Wichtigkeit, mit dem Herausnehmen einer Diskette solange zu warten, bis der Schreibvorgang beendet ist.

umount kann nur inaktive Dateisysteme absetzen. Das bedeutet, daß kein anderes Dateisystem auf dem abzusetzenden System aufgesetzt sein darf, und daß kein Prozess ein Verzeichnis des abzusetzenden Systems als Arbeitsverzeichnis benutzen darf. Insbesondere darf sich kein Benutzer in dem Abzusetzenden Dateisystem aufhalten.

Optionen:

-a setzt alle in /etc/fstab aufgeführten Devices ab

-t *Typ* setzt nur Dateisysteme vom *Typ* ab. Beschreibung des Parameters *Typ* ist beim Kommando mount zu finden.

Kapitel 3

Reise durch's Dateisystem

Hinter dem Linux Dateisystem (Unix Filesystem) steckt ein sehr umfangreiches Konzept. Um die Möglichkeiten dieses Dateisystems voll zu nutzen, ist es sinnvoll, ein wenig über die dahinterstehenden Ideen zu verstehen.

Zuerst einmal befinden sich in jedem Dateisystem Dateien und Verzeichnisse. Dateien sind (auf den ersten Blick) Texte, Programme oder sonstige Daten, die in ihrer binären Form als "Strom" von Bytes in Blöcken auf einem Massenspeicher (Festplatte oder Diskette) festgehalten und wieder gelesen werden können. Dateien werden bei der Erzeugung mit einem Namen in ein Verzeichnis eingetragen, und können danach unter diesem Namen angesprochen werden. Verzeichnisse können neben Dateien auch noch weitere Verzeichnisse enthalten, die ihrerseits wieder Dateien und Verzeichnisse enthalten können. Darin unterscheidet sich das Linux Dateisystem nicht von dem anderer Betriebssysteme.

Diese verschachtelte Struktur dient nicht der Verwirrung des Benutzers, sondern umgekehrt soll dadurch eine Ordnung in die Vielzahl verschiedener Dateien gebracht werden. Eine der grundlegenden Ideen des Linux (Unix) Dateisystems ist die Standardisierung einer Grundstruktur. Auf diese Weise können Unixrechner aller Sorten vernetzt werden, und die Benutzer auf dem einen oder dem anderen Rechner arbeiten, ohne jedesmal ein ganz anderes Ordnungsprinzip zu finden.

Um ein geeignetes Ordnungssystem zu finden, müssen Kriterien zu dessen Bewertung aufgestellt werden. Hier sind ein paar Forderungen an ein gutes Dateisystem:

Ein gutes Dateisystem soll übersichtlich, sicher, schnell und flexibel sein. Für jede der Forderungen lassen sich ein paar Faktoren benennen, die einen bestimmten Aspekt beeinflussen. Natürlich finden auch negative Wechselwirkungen mit anderen Aspekten statt, sodaß die folgenden Überlegungen nur als Anregung verstanden werden können.

- Der Übersichtlichkeit dient ein klar strukturierter Verzeichnisbaum, in dem Äste und Blätter deutlich getrennt sind
 - Verzeichnisse sollten so angelegt werden, daß sie entweder Daten oder Verzeichnisse enthalten.
 - Verzeichnisse, die hauptsächlich Daten enthalten, sollten höchstens so viele Einträge enthalten, wie bei einem Spaltenlisting auf einem Bildschirm angezeigt werden können. Wenn mehr Dateien in den gleichen Zusammenhang gehören, sollte das Verzeichnis besser unterteilt werden.
 - Verzeichnisse, die hauptsächlich Unterverzeichnisse enthalten, sollten nicht mehr Einträge enthalten, als bei einem langen Listing auf einem Bildschirm angezeigt werden können.
- Der Sicherheit dient es, den Systembereich vom Benutzerbereich und den veränderbaren Bereich vom 'statischen' Bereich logisch und physikalisch zu trennen.
 - Die Heimatverzeichnisse der Anwender sollten in einem eigenen Ast des Dateisystems, auf einer eigenen Partition angelegt werden. Dieser Ast kann beispielsweise an /home oder /usr/home anschließen.

Wenn ein lokaler Systembereich für die Benutzer zum Beschreiben freigegeben werden soll, ist es sinnvoll auch diesen Bereich vom übrigen System gesondert zu installieren. Traditionell wird dieser Bereich in dem Verzeichnis `/usr/local` angelegt.

- Das Rootfilesystem muß alle notwendigen Programme und Dateien für den Systemverwalter enthalten. Vor allem eine Shell und die Programme zum Montieren und Reparieren des Dateisystems sind hier wichtig. Aber auch ein Editor und die grundlegenden Werkzeuge können hier untergebracht werden.
 - Wenn das Rootfilesystem auf der Festplatte angesiedelt ist, und nicht in einer RAM-Disk, sollte kein Programm darin irgendwelche temporären Dateien oder Logfiles anlegen. Ein Dateisystem in das nicht geschrieben wird, kann bei einem Systemabsturz (fast) nicht beschädigt werden.
- Der Schnelligkeit dienen kleine Verzeichnisse auf dem Pfad (PATH).
 - Das Wurzelverzeichnis und das Verzeichnis `/usr` sollten so klein gehalten werden, daß jedes von ihnen nur einen Block (1024 Bytes) belegt. Auf größere Verzeichnisse muß das Betriebssystem indirekt zugreifen, was die Geschwindigkeit verringert.
 - Wenn ein Verzeichnis einmal größer geworden ist, wird es auch durch Löschen von Einträgen nicht mehr kleiner!

Die verwendete Grundstruktur des Linux Dateisystems ist nicht das Ergebnis konsequenter, rationaler Überlegung, sondern entspricht dem “traditionellen” Aufbau eines System V Dateisystems. Es ist das Produkt der fast zwanzigjährigen (AT&T) Unixgeschichte. Das Dateisystem von BSD sieht ein wenig anders aus. Beiden Strukturen gemeinsam ist aber die (nicht immer konsequente) technische Orientierung: Es werden tendenziell alle ausführbaren Programme in einem Verzeichnis zusammengefaßt, alle Hilfstexte in einem anderen, und die Initialisierungsdateien in einem dritten. Die Zusammenfassung eines Programms mit allen für dieses Programm nötigen Dateien in einem Verzeichnis ist unüblich, obwohl sie im Fall besonders umfangreicher Programme vorkommen kann.

Eine weitere grundlegende Idee des Linux (Unix) Dateisystems besteht darin, die technische Realisierung des konkreten Systems vollkommen vor dem Anwender zu verbergen. Es gibt für den Benutzer nur ein einziges Dateisystem. Einzelne Laufwerke oder Partitionen werden nicht unterschieden.

Das Konzept des Linux Dateisystems beinhaltet noch einen wichtigen Punkt: Alle Geräte und Komponenten (Devices) der Systemhardware, die einem Benutzer zur Verfügung gestellt werden können, haben ein “Bild” im Dateisystem. Nur über dieses Bild kann ein Anwender Zugriff auf die Geräte und Komponenten bekommen.

Dieses Prinzip ermöglicht es dem Kernel, den Zugriff so zu steuern, daß die “gleichzeitige” Benutzung ohne Fehler abgewickelt werden kann.

3.1 Namen und Pfade

Das Dateisystem ist schön mit einem Baum zu vergleichen, dessen Äste die Verzeichnisse sind, und dessen Blätter die Dateien. Die Stelle an denen zwei Verzeichnisse oder ein Verzeichnis und eine Datei zusammenstoßen wird durch einen einfachen Schrägstrich ‘/’ (Slash) bezeichnet.

Die Wurzel des Baumes wird mit einem einzelnen Slash ‘/’ bezeichnet.

Jede Datei hat einen **Dateinamen**, mit dem sie in einem Verzeichnis eingetragen ist. Wenn die Datei in einem anderen als dem aktuellen Verzeichnis liegt, muß sie mit einem **Pfadnamen** angesprochen werden. Dieser Name setzt sich aus einer Reihe von **Verzeichnisnamen** und einem Dateinamen zusammen. Die Reihe der Verzeichnisnamen wird auch als **Pfad** bezeichnet. Die einzelnen Verzeichnisnamen werden alle durch ‘/’ (Slash) Zeichen getrennt.

Um nun eine Datei in einem anderen Verzeichnis, oder auch einfach nur das andere Verzeichnis anzusprechen, kann der Pfadname entweder relativ zum aktuellen Verzeichnis, oder als **absoluter** (Pfad-) Name vom Wurzelverzeichnis aus angegeben werden.

Um bei einem relativen Pfadnamen im Verzeichnisbaum in Richtung der Wurzel zu gehen, muß anstelle eines regulären Verzeichnisnamens das Verzeichnis `‘..’` angegeben werden. Dieses Verzeichnis ist in jedem Verzeichnis enthalten und bezeichnet das nächste Verzeichnis in Richtung Wurzel.

Neben dem Bild des Verzeichnisbaums gibt es noch das einer Hierarchie, die im Prinzip die gleiche Ordnung erzeugt, aber in genau der umgekehrten Richtung. Wenn ein Verzeichnis beim Baum näher an der Wurzel, also tiefer ist, ist es in der Hierarchie höher.

3.2 Das Wurzelverzeichnis

Den oben aufgestellten Regeln entsprechend, enthält das Wurzelverzeichnis nur eine kleine Zahl von Verzeichnissen, die alle eine klar umgrenzte Funktion haben.

Weil nur das Rootfilesystem mit dem Wurzelverzeichnis vom Kernel selbst installiert wird, und alle anderen Dateisysteme erst bei der Systeminitialisierung vom `mount` Kommando zusammengebaut werden, müssen bereits im Rootfilesystem eine Anzahl “lebenswichtiger” Programme vorhanden sein. Das ist der Sinn der Verzeichnisse `/bin`, in dem die ausführbaren Programme liegen, `/lib`, in dem die Shared Librarys für alle Programme sind, und `/etc`, in dem eine Reihe von Konfigurationsdateien und ein paar Kommandos für den Systemverwalter abgelegt sind.

Das Verzeichnis `/mnt` dient als Aufsetzpunkt für Dateisysteme, die nur vorübergehend in den Dateisystembaum eingefügt werden sollen.

Das Verzeichnis `/tmp` wird von einigen Programmen benutzt, um temporäre Dateien darin abzulegen, die normalerweise spätestens beim beenden des Programms wieder gelöscht werden. Wenn das Rootfilesystem nicht im Arbeitsspeicher (RAM-Disk) sondern auf der Festplatte gehalten wird, sollte das Verzeichnis `/tmp` als symbolischer Link auf das Verzeichnis `/usr/tmp` oder ein anders Verzeichnis auf einer weniger sensiblen Partition verweisen.

3.3 Das Verzeichnis `/etc`

Im Verzeichnis `/etc` befinden sich Dateien und Programme zur Systemverwaltung. Ein Teil der Dateien kann und muß vom Systemverwalter bearbeitet werden. Diese Dateien sind normale Textdateien, die mit jedem Editor bearbeitet werden können. Der Inhalt muß meist eine ganz bestimmte Form haben, damit die Programme, die damit arbeiten den Inhalt “verstehen”.

Aus Gründen der Systemsicherheit sollten “normale” Benutzer die Dateien im Verzeichnis `/etc` nur lesen, aber nicht beschreiben können.

`/etc/fstab`

In der Datei `/etc/fstab` sind die dauerhaften Parameter für die Zusammensetzung des Dateisystems gespeichert. Hier legt der Systemverwalter fest, welche Partitionen an welcher Stelle in das Dateisystem eingehängt werden. Die Datei wird von dem Kommando `mount` ausgewertet. Normalerweise wird dieses Kommando automatisch bei der Systeminitialisierung vor den anderen Programmen ausgeführt. Dabei wird die Datei `/etc/fstab` Zeile für Zeile abgearbeitet.

Für jede Partition muß eine eigene Zeile mit 4 Einträgen angelegt werden. Die Einträge werden durch Leerzeichen oder Tabulator voneinander getrennt.

Der erste Eintrag ist die zu der Partition gehörende Gerätedatei im `/dev` Verzeichnis (Device). Der Name dieser Datei muß mit absolutem Pfadnamen angegeben werden, also z.B. `‘/dev/hda2’` oder `‘/dev/sdb1’`. Im Ausnahmefall des Prozeßdateisystems sollte anstelle des Dateinamens das Schlüsselwort `none` stehen.

Der zweite Eintrag bezeichnet das Verzeichnis, auf dem das Teilsystem aufgesetzt werden soll. Durch die Reihenfolge der Zeilen in der Datei `/etc/fstab` muß sichergestellt sein, daß das Verzeichnis, auf dem ein Teilsystem aufgesetzt wird, auch tatsächlich im Dateisystem vorhanden ist.

Der dritte Eintrag bezeichnt den Typ des Dateisystems. Linux unterstützt das “alte” Dateisystem von Minix und ein eigenes, erweitertes Dateisystem. Und es erlaubt die Einbindung von MS-DOS Dateisystemen in das

Linux Dateisystem sowie die Benutzung von "Swapdateien" oder "Swappartitionen" für die Auslagerung von schlafenden Prozessen. Entsprechend muß der dritte Eintrag 'minix', 'ext', 'msdos' oder 'swap' heißen. Wenn die Partition (oder ein Diskettenlaufwerk) nicht mit festen Parametern belegt werden kann, oder wenn es nicht automatisch beim Systemstart mit den anderen zu einem Dateisystem verbunden werden soll, kann an Stelle des dritten Eintrags auch das Wort 'ignore' stehen.

Neben den "realen" Dateisystemen unterstützt Linux auch ein sogenanntes Prozessdateisystem. In diesem Dateisystem werden die einzelnen Prozesse der Kernels abgebildet. Dieses Dateisystem wird zum Beispiel vom neuen **ps** Kommando benutzt, um unabhängig von der Kernelversion die Daten aus der Prozeßtafel zu lesen. Das Prozeßdateisystem hat die Typenbezeichnung 'proc'.

Der vierte Eintrag schließlich entspricht den Optionen, die dem **mount** Kommando für die Behandlung der kompletten Partition übergeben werden können. Es kann eine durch Kommata getrennte Liste der folgenden Optionen angegeben werden:

noexec verbietet die Ausführung jedes (binären) Programms von dieser Partition

nosuid unterdrückt die Wirkung der SUID und SGID Bits bei der Ausführung von Programmen aus dieser Partition

nodev die zeichen- und blockorientierten Gerätedateien in dieser Partition werden nicht angesprochen

synchronous die Lese-/Schreibfunktionen werden ungepuffert ausgeführt

ro verbietet das Schreiben auf diese Partition

sw kennzeichnet eine Swappartition

xx ist für Partitionen, die nicht aufgesetzt werden sollen

defaults erlaubt alles, was oben verboten wurde

Wenn das erste Zeichen einer Zeile ein '#' ist, wird die komplette Zeile als Kommentar behandelt und ignoriert.

/etc/group

In dieser Datei sind die Benutzergruppen und ihre Mitglieder festgehalten. Mit dem **newgrp** Kommando kann jeder hier eingetragene Anwender die aktuelle Benutzergruppe wechseln.

Die Datensätze der group Datei haben folgendes Format:

Gruppenname:Paßwort:Gruppennummer:Mitgliederliste

Neben den Gruppen, die vom Systemverwalter für "lebendige" Benutzer einrichten kann, sind hier eine ganze Reihe Gruppen für Systemaufgaben eingetragen. Beispielsweise können die folgenden Gruppen zu finden sein:

root (0) ist die privilegierte Gruppe des Superusers root

other (1) möglicherweise eine Benutzergruppe

bin (2) für die ausführbaren Systemkommandos in /bin und /usr/bin

sys (3)

adm (4)

uucp (5) für das uucp Programm und seine Verwandten

news (6) für cnews, tin und alle Newsdateien

mail (7) für elm, smail und Verwandte

disk (8) die Gerätedateien für Festplattenpartitionen

floppy (9) die Gerätedateien für Diskettenlaufwerke

tty (10) die Gerätedateien für serielle Terminals

tape (11) die Gerätedateien für Bandlaufwerke

daemon (12) für Dämonen aller Sorten

cron (16) für den Crondämon und seinen Konfigurationsdateien

lp (18) die Gerätedateien für die Drucker und der Druckerdämon

mem (20) die Gerätedateien für den Arbeitsspeicher und die Programme, die damit arbeiten

kmem (21) der Speicherbereich des Kernels und die Programme, die damit arbeiten

sysadmin (23) Programme und Dateien, die nur der Systemverwaltung dienen

group (50) eine Benutzergruppe

user (75) eine andere Benutzergruppe

Sinn vieler der oben aufgeführten Gruppen ist es, den Benutzern den kontrollierten Zugriff auf bestimmte Teile des Systems (News, UUCP, Diskettenlaufwerke, Drucker ...) zu ermöglichen, indem bestimmte Programme SGID laufen, das heißt das die Benutzer zur Laufzeit des Programms die Rechte der entsprechenden Gruppe erhalten, ohne selbst Mitglied dieser Gruppe zu sein.

/etc/hosts

In der Datei /etc/hosts sind die Netzwerknummern (IP-Nummern) und die Namen der in einem Netzwerk erreichbaren Rechner gespeichert. Sie wird von der **bash** für die Hostnamenerweiterung benutzt.

Die Einträge sind einfacher Text und bestehen aus den Netzwerknummern der Hostrechner am Anfang einer Zeile, und den offiziellen oder inoffiziellen Namen dieser Rechner, jeweils durch Leerzeichen oder Tabulatoren getrennt.

Wenn in dem Netzwerk kein Nameserver eingerichtet ist, benutzen Netzwerkprogramme wie **ftp**, **telnet** oder **rlogin** diese Datei ebenfalls, zum Übersetzen von verbalen Namen der Netzrechner in die entsprechenden IP-Nummern.

Die Datei kann in diesem Fall auch ein (symbolischer) Link auf eine gleichnamige Datei im Verzeichnis /usr/etc/inet sein.

/etc/inittab

Die Datei /etc/inittab wird vom **init** Programm benutzt, das die darin festgelegten Prozesse startet und so das Benutzersystem initialisiert.

Es gibt mehrere Versionen des **init** Programms, die sich erheblich voneinander unterscheiden.

In der **inittab** Datei für das einfache **init** (**simpleinit**) von Peter Orbaek besteht jede Zeile aus drei durch Doppelpunkt getrennte Einträge.

Terminal: Termcapeintrag: Gettykommando

Der erste Eintrag bezeichnet das Terminal (**tty1**, **tty2**, **ttys1** ...) wie es in der /etc/utmp Datenbank eingetragen wird, und wie es vom **who** Kommando angezeigt wird. Der zweite Eintrag wird automatisch in die **TERM** Umgebungsvariable der auf dem entsprechenden Terminal gestarteten Shell geschrieben, und sollte deshalb mit einem Eintrag in der /etc/termcap Datei übereinstimmen.

Der dritte Eintrag schließlich ist die Kommandozeile, mit der das **getty** Kommando für das Terminal aufgerufen werden muß. Da es auch vom **getty** Programm verschiedene Versionen gibt, sollte das entsprechende Format am besten der Beschreibung dieses **getty** entnommen werden.

Das System 5 kompatible init (**“sysvinit”**) Programm von Mike Jagdis hat im Prinzip auch die Aufgabe, die **getty** Prozesse für die virtuellen Terminals zu erzeugen und immer neu zu starten. Es bietet aber noch weitere sehr flexible Möglichkeiten der Systeminitialisierung.

Die Einträge für die **inittab** sehen hier grundsätzlich anders aus. Jede Zeile die nicht mit einem **‘#’** (Kommentar) beginnt ist ein Datensatz mit vier durch Doppelpunkt voneinander getrennte Einträgen.

ID:Runlevel:Aktion:Prozess

Der erste Eintrag ist eine zweistellige Zeichenkette als Bezeichner für die Zeile.

Im zweiten Eintrag werden die Runlevel festgelegt, für die eine Zeile gültig ist. Die Runlevel werden mit Ziffern von 0 bis 9 und mit allen Buchstaben (ohne Unterscheidung zwischen Groß und Kleinschreibung) bezeichnet. Es können auch mehrere Runlevel in einer Zeile angegeben werden. Bei den Buchstabenkennungen für die Level steht S für den Einbenutzermodus, Q für Quit zum Neueinlesen der **inittab**. Die anderen Buchstaben werden für **“ondemand”** Aufrufe verwendet, bei denen ein Kommando beim Moduswechsel nicht mehr abgebrochen wird.

Wenn das Feld für den Runlevel leer ist, wird die Aktion bei jedem Moduswechsel ausgeführt.

Im dritten Eintrag wird bestimmt, welche Aktion von init ausgeführt wird:

initdefault Das zweite Feld dieser Zeile bestimmt den Runlevel beim Systemstart. Dieser Eintrag kann mit der Datei **/etc/initrundl** oder durch einen entsprechenden Parameter auf der Kommandozeile (wie er z. B. von LILO übergeben wird) verdeckt werden. Wenn keine Zeile mit **initdefault** in der **inittab** vorkommt, wird das System im Einbenutzermodus gestartet.

sysinit Das Kommando im vierten Feld dieser Zeile wird einmal unmittelbar nach dem Systemstart ausgeführt, noch bevor irgendein Anwender Zugang zum System erhält, also auch vor dem Einbenutzermodus

bootwait Das Kommando dieser Zeile wird einmal ausgeführt wenn in einen Mehrbenutzermodus gewechselt wird. Das init Programm wartet mit der Bearbeitung weiter Zeilen in der **inittab** bis das in dieser Zeile aufgerufenen Kommando beendet ist.

boot Das Kommando im vierten Feld dieser Zeile wird einmal beim Wechsel in einen Mehrbenutzermodus ausgeführt.

respawn Das Kommando im vierten Feld dieser Zeile wird beim Übergang in einen im zweiten Feld aufgeführten Modus gestartet (wenn es nicht bereits läuft) und es wird jedesmal neu gestartet, wenn es beendet wurde (zum Beispiel **getty**).

ondemand hat die gleiche Funktionalität wie **respawn**, und wird benutzt um im Zusammenhang mit den durch Buchstaben gekennzeichneten Leveln einzelne Kommandos **“auf Anfrage”** zu starten.

wait Das Kommando in dieser Zeile wird beim Übergang in einen im zweiten Feld aufgeführten Modus ausgeführt, und mit der Bearbeitung weiterer Zeilen der **inittab** gewartet bis das Kommando beendet wurde.

once Das Kommando im vierten Feld dieser Zeile wird beim Übergang in einen passenden Modus einmal aufgerufen. Auf die Beendigung wird nicht gewartet.

off Wenn das Kommando im vierten Feld läuft, wird es angehalten, sonst wird der Eintrag ignoriert.

update Das init Programm sorgt selbst für die Synchronisation des Dateisystems, so das **update** Kommando. Im vierten Feld wird angegeben, in welchem Abstand (in Sekunden) der **sync** Systemaufruf ausgeführt werden soll.

runlevel Mit der **runlevel** Aktion wird einem Runlevel, der im zweiten Feld angegeben wird, ein Name zugeordnet, der im vierten Feld angegeben wird. Mit diesem Namen kann der entsprechende Runlevel dann mit dem **telinit** Kommando anstelle der Nummer bzw. Buchstabenkennung aufgerufen werden.

/etc/inittunlv

Diese Datei wird vom **sysvinit** benutzt, das daraus den Runlevel nach dem Systemstart ausliest. Dieser Parameter verdeckt den entsprechenden Eintrag in der Datei **/etc/inittab**.

Der Eintrag besteht aus einem einzelnen Zeichen (Buchstabe oder Ziffer).

/etc/issue

Die Datei **/etc/issue** wird nur vom **getty** Programm benutzt. Wenn die Datei **/etc/issue** existiert, wird ihr Inhalt als Meldung vor dem Loginprompt ausgegeben. Der Inhalt ist ein einfacher Text.

/etc/magic

Die Datenbank **/etc/magic** wird von dem Kommando **file** benutzt, das die hier gespeicherten Merkmale mit den zu untersuchenden Dateien vergleicht, und dadurch eine Zuordnung der Datei zu einem Typ versucht. Wenn eine Datei auf diese Weise zugeordnet werden konnte, wird die hier festgelegte Meldung ausgegeben.

/etc/motd

In dieser Datei soll eine Mitteilung abgelegt werden, die jedem Benutzer nach dem **login** angezeigt wird. Je nach Version des **login** Kommandos findet die Anzeige automatisch statt, oder sie wird bei der Initialisierung der Loginshell durch die Datei **/etc/profile** ausgeführt.

/etc/nologin

Die Datei **/etc/nologin** wird nur vom **login** Programm benutzt. Wenn diese Datei existiert, ist jedes "normale" Einloggen im System unmöglich. Nur der Superuser (**root**) kann sich trotzdem beim System anmelden. Wenn ein anderer Benutzer versucht sich einzuloggen, wird der Inhalt der Datei **/etc/nologin** ausgegeben.

Es ist ratsam, in der Datei **/etc/rc** bei der Initialisierung des Systems mit dem Kommando **'rm -f /etc/nologin'** eine eventuell noch vorhandene Sperrung zu lösen.

/etc/passwd

Die Datei **/etc/passwd** ist die Benutzerdatenbank des Systems. Hier werden die Namen, die Benutzernummer und das Heimatverzeichnis der Anwender gespeichert. Außerdem werden in der "normalen" **passwd** Datei auch die verschlüsselten Paßworte der Benutzer gespeichert. Für öffentlich zugängliche Systeme mit besonders hohen Sicherheitsansprüchen gibt es ein sogenanntes shadow Paßwortsystem, bei dem die Paßworte in einer separaten Datei gespeichert werden. Die Funktionen dieses Paßwortsystems können in den englischen Hilfstexten mit dem **man** Kommando nachgelesen werden.

Die Datensätze der Paßwortdatei bestehen aus:

Benutzername:Paßwort:Benutzernummer:Gruppennummer:Name:Heimat:Shell

Jeder Benutzer kann mit dem **passwd** Kommando sein Paßwort selbstständig ändern. Er sollte das in regelmäßigen Abständen und zu besonderen Anlässen auch tun.

In der Paßwortdatei sind neben den lebendigen Benutzern auch eine Reihe von Benutzernamen zu finden, die für bestimmte Verwaltungszwecke eingerichtet werden. Beispielsweise können folgende Benutzernamen in der Paßwortdatei zu finden sein:

root (0) der Superuser

daemon (1) der unbekannte Dämon

bin (2) der Eigentümer der ausführbaren Dateien in **/bin** und **/usr/bin**

sync (3) das Kommando **sync**

uucp (5) Eigentümer des UUCP Systems

news (6) Eigentümer des News Systems

ftp (10) der anonyme FTP Zugang

nuucp (11) der anonyme UUCP Zugang

Einige der oben genannten Accounts (z.B. bin oder daemon) sind mit Sperrpaßworten versehen, die das Einloggen unter diesem Benutzernamen unmöglich machen. Diese Benutzernamen dienen wie die gleichnamigen Gruppen dazu, den normalen Benutzern den kontrollierten Zugriff auf bestimmte Teile des Systems zu ermöglichen (siehe bei /etc/group auf Seite 164). Andere Accounts sind nicht durch ein Passwort geschützt, und erlauben so den anonymen Zugang zum System für Gäste. Eine besondere Aufgabe hat der sync Account, der es jedem Anwender auf jedem Terminal ermöglicht, das Dateisystem zu synchronisieren, auch ohne sich ordentlich beim Betriebssystem anzumelden.

/etc/printcap

Die Datei /etc/printcap enthält eine stark formalisierte Beschreibung des oder der Ducker des Systems. Sie wird vom **lpd** Druckerdämon ausgewertet, der die Druckjobs im System verwaltet. Eine Beschreibung der Einträge für diese Datei ist in der englischen Manualpage mit dem **man** Kommando nachzulesen.

/etc/profile

Die Datei /etc/profile wird von allen Loginshells (aller Benutzer) gelesen und als Shellsript ausgeführt. Hier werden grundsätzliche Einstellungen der Shellumgebung für alle Anwender gemeinsam vorgenommen. Wenn sie nicht vor dem Überschreiben geschützt werden (siehe beim Shellkommando **typeset**) können alle Einstellungen von einer benutzereigenen Initialisierungsdatei wieder geändert werden.

In /etc/profile kann z. B. der Hostname gesetzt werden, die Begrüßung aus der Datei /etc/motd angezeigt werden (wenn das nicht schon vom **login** erledigt wird), der Pfad für alle Benutzer eingestellt werden, ein **sync** als **trap** auf **EXIT** gelegt werden, und vieles andere mehr.

/etc/psdatabase

Die Datenbank /etc/psdatabase ist so etwas wie eine Landkarte des laufenden Betriebssystems. Über diese Datenbank können einige Programme (z. B. **ps**, **top**, **free**, **fstat**) direkt auf den (Kernel-) Speicher zugreifen, um bestimmte Informationen über den Zustand des gesamten Systems zu erhalten. Für diese Programme muß das SUID oder der SGID Bit gesetzt sein, wenn den normalen Benutzern das entsprechende Kommando zur Verfügung stehen soll.

Die Datei /etc/psdatabase wird vom Programm **ps** mit der Option ‘-U’ angelegt. Dazu wird die Datei /etc/system gelesen, in der Daten über den Kernel enthalten sind.

Neuere Versionen von **ps**, **free**, **tload** und **uptime** benutzen anstelle der psdatabase das sogenannte Prozessdateisystem (proc filesystem). Damit entfällt die Notwendigkeit, nach jeder Veränderung am Kernel eine neue psdatabase zu erstellen.

/etc/rc

Die Datei /etc/rc ist die Systeminitialisierungsdatei. Sie wird vom ‘init’ Programm der Shell übergeben, die die darin enthaltenen Kommandozeilen ausführt. Diese Initialisierung findet beim Übergang des Systems in einen Mehrbenutzermodus statt.

/etc/securesingle

Die Datei /etc/securesingle wird vom einfachen init Programm gesucht, und wenn sie vorhanden ist, wird der Anwender im Einbenutzermodus nach dem Rootpaßwort gefragt.

/etc/securetty

In der Datei `/etc/securetty` werden die Ports (Terminals) angegeben, an denen sich der Superuser (root) einloggen darf. Diese Datei wird vom `login` Programm gelesen und ausgewertet.

/etc/shells

In der Datei `/etc/shells` sind alle verfügbaren (zugelassenen und uneingeschränkten) Loginshells eingetragen. Sie wird vom `chsh` Kommando ausgewertet. Dem Anwender wird damit die Möglichkeit gegeben, die in dieser Datei zeilenweise aufgelisteten Programme als Loginshell in der Datei `/etc/passwd` einzutragen.

Ausserdem wird sie vom `ftp` Dämon benutzt, um festzustellen ob ein Benutzer einen uneingeschränkten Shellaccount hat, um ihm in diesem Fall auch den uneingeschränkten `ftp` Zugang zu gestatten.

/etc/singleboot

Die Datei `/etc/singleboot` wird vom einfachen `init` Programm gesucht. Wenn sie vorhanden ist, wird das System im Einbenutzermodus gestartet.

/etc/system

Die Datei `/etc/system` ist (meist) ein symbolischer Link auf die Datei `/usr/src/linux/tools/system`. Diese Datei wird vom `make` Programm bei der Übersetzung eines neuen Kernels erstellt. Sie enthält Daten über den Aufbau des Kernels, und wird vor allem vom `ps` Kommando für die Erstellung der Datei `/etc/psdatabase` benötigt.

/etc/termcap

Die Datei `/etc/termcap` ist eine Datenbank, in der die Steuersequenzen für verschiedene Terminals abgespeichert sind. Diese Datenbank ist stark formalisiert, kann aber mit jedem Editor gelesen und bearbeitet werden.

Viele Programme benutzen diese Datenbank, indem sie die Steuerzeichen für die Bildschirmausgabe und die ankommenden Tastaturcodes mit den entsprechenden Einträgen in der Datenbank übersetzen. Die zu einem Terminal passende Übersetzungstabelle wird in der `TERM` Umgebungsvariablen bestimmt, die vom `getty` Programm oder von bei der Shellinitialisierung gesetzt wird.

/etc/utmp

Der Inhalt der Datei `/etc/utmp` kann mit dem `who` Kommando angezeigt werden. Sie enthält Informationen über die aktiven Benutzer. Sie wird automatisch erstellt und aktualisiert, wenn eine Shell für ein Terminal gestartet oder beendet wird. Um das Rootfilesystem vor Beschädigung zu schützen, wird diese Datei auch häufig durch ein Link mit einer gleichnamigen Datei in `/usr/adm` oder einem andern Verzeichnis in einem weniger sensiblen Systembereich verbunden.

Beim Hochfahren des Systems sollte diese Datei geleert werden.

/etc/wtmp

Der Inhalt der Datei `/etc/wtmp` kann mit dem Kommando `last` angezeigt werden. Hier werden die Namen und Benutzungszeiten aller Benutzer gespeichert.

Aus den oben bei der Beschreibung von `/etc/utmp` angeführten Gründen wird auch diese Datei häufig durch einen symbolischen Link mit einer gleichnamigen Datei im Verzeichnis `/usr/adm` verbunden.

3.4 Das Verzeichnis /dev

In dem Verzeichnis /dev sind die “Bilder” der Hardwarekomponenten und Geräte (Devices) des Systems abgelegt. Die Bilder sind keine Dateien im herkömmlichen Sinne. Sie belegen keinen Platz auf der Festplatte. Sie sind Gerätedateien, die mit dem Betriebssystemkern (Kernel) in Verbindung stehen, der die eigentlichen Geräte verwaltet. In diese Dateien kann geschrieben oder aus ihnen gelesen werden, wie in normale Dateien. Der Datenstrom wird vom Kernel übernommen und an das entsprechende Gerät zu einem passenden Zeitpunkt weitergeleitet. Natürlich gelten auch für die Gerätedateien alle Zugriffsbeschränkungen wie bei normalen Dateien.

Die Verbindung zum Kernel wird über Slots oder Kanäle hergestellt, die nummeriert sind, und hinter denen sich die Treiber für die Geräte verbergen. Die Nummer des Gerätetreibers wird als **Major Device Number** bezeichnet. Ein Treiber kann mehrere Geräte des gleichen Typs verwalten. Um die einzelnen Geräte zu unterscheiden, wird dem Treiber eine zweite Zahl, die **Minor Device Number** übergeben. Diese beiden Zahlen charakterisieren jede Datei im /dev Verzeichnis. Zusätzlich werden noch zwei Arten von Geräten unterschieden: Die blockorientierten Geräte, wie z. B. Disketten oder Festplatten, und die zeichenorientierten Geräte, wie Drucker, Terminal oder Maus. Für jede Gerätedatei kann festgelegt werden, ob sie block- oder zeichenorientiert arbeiten soll.

Bei einem langen Listing des Verzeichnisses mit ‘ls -l’ werden die Major und Minor Device Numbers anstelle der Dateigröße angezeigt. Ob ein Gerät als Zeichen- oder Blockdevice angesprochen wird, kann man am ersten Buchstaben der Zugriffsrechte sehen. Da steht ‘b’ für block- und ‘c’ für zeichenorientierte Gerätedatei.

Die Namensgebung für die Gerätedateien ist willkürlich. Jede Datei kann beliebig umbenannt werden oder durch Links mit anderen Namen verbunden werden. Die Benutzung des Gerätes ändert sich dadurch nicht.

Für den Anwender sind praktisch nur die Dateien /dev/fd?, /dev/lp?, /dev/null, /dev/zero und /dev/tty? von Interesse. Alle anderen Dateien werden nur vom Systemverwalter benötigt.

/dev/cua?

Die Dateien /dev/cua? werden seit der Kernelversion 0.99.5 unterstützt, und stellen die mit den Dateien /dev/ttyS? bereits abgebildeten seriellen Schnittstellen nochmal speziell für ausgehende Modemverbindungen zur Verfügung. Der Kernel blockiert ein auf der Modemleitung wartendes **getty** für die Dauer einer ausgehenden Modemverbindung, und läßt umgekehrt keine ausgehenden Verbindungen zu, solange ein Benutzer auf einer eingehenden Modemleitung aktiv ist.

/dev/fd*

Die Dateien /dev/fd* repräsentieren die Diskettenlaufwerke. Weil der Kernel das Diskettenformat automatisch erkennen kann, sind die beiden Dateien /dev/fd0 und /dev/fd1 für die meisten direkten Diskettenzugriffe ausreichend. Trotzdem werden für alle gängigen Diskettenformate spezielle Einträge mit entsprechenden Minor Device Numbers angeboten. Auf diese Weise kann der Treiber gezwungen werden, ein bestimmtes Diskettenformat anzunehmen, was zum Beispiel zum Formatieren von Disketten notwendig ist.

Das System der Namensgebung für die Dateien ist willkürlich, aber leicht zu verstehen. Nach den Buchstaben ‘fd’ für Floppy Disk kommt die Laufwerksnummer (0 oder 1) und dann folgt ein großer Buchstabe für ein 3.5 Zoll Laufwerk und ein kleiner Buchstabe für ein 5 1/4 Zoll Laufwerk, schließlich kommt noch die Speicherkapazität in Kilobytes. Eine Liste der Dateinamen ist beim Kommando **fdformat** auf Seite 83 zu finden.

Die Diskettenlaufwerke sind normalerweise für alle Anwender beschreibbar, indem entweder die Gerätedateien selbst den Schreibzugriff erlauben, oder indem die schreibenden Programme das SUID oder SGID Bit gesetzt haben, sodaß jeder Benutzer seine eigenen Daten auf Diskette sichern kann. Allerdings ist es aus Gründen der Systemsicherheit nur dem Superuser (root) erlaubt, das Dateisystem zu verändern. Deshalb kann ein normaler Benutzer kein Diskettendateisystem anmelden. (Siehe auch unter **mttools** auf Seite 118.)

/dev/hd*

Die Dateien /dev/hd* repräsentieren die “normalen” (IDE, RLL, MFM) Festplatten. Die Major Device Nummer des Festplattentreibers ist 3. Die erste Festplatte wird über die Minor Device Nummern ab 0 angesprochen, die zweite Festplatte belegt die Minor Device Nummern ab 64. Dabei stehen hda und hdb (0 und 64) für die gesamte (rohe) Festplatte, und die durchlaufend nummerierten Dateien mit den entsprechenden Minor Device Nummern stehen für die einzelnen Partitionen.

Im Gegensatz zu den Diskettenlaufwerken sind die Festplatten nur für den Superuser (root) direkt ansprechbar. Alle anderen Anwender sind an die normalen Zugänge über das Dateisystem gebunden.

/dev/lp*

Die Gerätedateien /dev/lp* bilden die parallelen Druckerschnittstellen im Dateisystem ab. Die Druckerdevices sind für jeden Benutzer beschreibbar. Wenn kein Druckerdämon im Hintergrund läuft, der die Druckjobs der Benutzer sofort abnimmt und zu einem geeigneten Zeitpunkt an einen Drucker weiterleitet, kann jeder Anwender seine Dokumente (in entsprechendem Format) direkt an einen Drucker schicken. Dazu kann zum Beispiel einfach das Kommando ‘**cat Datei > /dev/lp1**’ benutzt werden, das den Inhalt der *Datei* einfach als Datenstrom in die Gerätedatei schreibt. Der Kernel leitet dann den Datenstrom an den Drucker weiter, wenn der Drucker dazu bereit ist, und wenn kein anderer Benutzer den Drucker belegt.

Wenn der Drucker aus irgendwelchen Gründen nicht bereit ist, Daten anzunehmen, gibt der Kernel automatisch eine Fehlermeldung aus. Wenn das Gerät durch einen andern Job belegt ist, wird der Anwender darauf hingewiesen, daß das Gerät beschäftigt ist.

/dev/mem und /dev/kmem

Die Dateien /dev/mem und /dev/kmem bilden den Arbeitsspeicher des Rechners ab. Die Major Device Nummer ist 1 und die Minor Device Nummern sind 1 und 2. Beide Devices sind Zeichenorientiert.

Auf diese Devices greifen nur spezielle Programme wie **free** oder **ps** zu. Das direkte Lesen und Schreiben im Arbeitsspeicher des Rechners muß den Benutzern immer verboten sein.

/dev/null

Diese Spezialdatei ist für alle Anwender zum Lesen und Schreiben frei. Sie ist der Mülleimer des Systems. Alles was dorthin geleitet oder verschoben wird, verschwindet auf Nimmerwiedersehen.

/dev/port

Über die Spezialdatei /dev/port können einzelne IO Ports angesprochen werden.

/dev/ram

Die Datei /dev/ram bildet die Ramdisk in das Dateisystem ab, wenn beim Übersetzen des Kernels ein Teil des Speichers als solche bestimmt wurde. Die Ramdisk kann benutzt werden wie eine Diskette. Wenn die Ramdisk in das Dateisystem eingebunden werden soll, muß vorher mit dem **mkfs** Kommando ein Dateisystem darauf eingerichtet werden. Dann kann sie wie eine Diskette mit dem **mount** Kommando auf ein Verzeichnis aufgesetzt werden.

/dev/sd*

Die Dateien /dev/sd* bilden die SCSI Festplatten ab. Die Major Device Nummer für den SCSI Host ist 8. Die einzelnen Festplatten werden in Schritten zu 16 Minor Device Nummern für die Partitionen angesprochen. Die Namensgebung entspricht dem System bei den anderen Festplatten, ein Buchstabe für die Festplatte und eine Zahl für die Partition.

/dev/tty? und /dev/console

Hinter den Gerätedateien /dev/tty? verbergen sich die virtuellen Terminals. Ein Terminal besteht aus einem Bildschirm und einer Tastatur. Die Anzahl der virtuellen Terminals wird beim Übersetzen des Kernels festgelegt, und beträgt normalerweise acht. /dev/console und /dev/tty0 sind Synonyme für das aktuelle Terminal. Zwischen den virtuellen Terminals kann mit den Tastenkombinationen ALT-F1 bis ALT-F8 umgeschaltet werden.

Auf allen virtuellen Terminals kann ein `login` erfolgen, wenn für das entsprechende Terminal vom `init` Prozeß ein `getty` gestartet wurde.

Das reale Terminal und die reale Tastatur werden durch die Gerätedatei /dev/tty im Dateisystem abgebildet.

/dev/ttyp* und /dev/ptyp*

Die Spezialdateien /dev/ptyp? werden vom X11 Server (Master) bedient, der darüber mit den X-Terminals an den /dev/ttyp? (Slaves) kommunizieren kann.

/dev/ttyS*

Die /dev/ttyS? Dateien bilden die seriellen Schnittstellen (Maus, ASCII-Terminals, Drucker oder Modem) in das Dateisystem ab. Normalerweise sind bei den /dev/ttyS? Gerätedateien die Schreib- und Leserechte für alle Benutzer gesetzt, sodaß die Maus beispielsweise jedem Anwender zur Verfügung steht.

Im Zusammenhang mit Modems an der seriellen Schnittstelle werden seit der Kernelversion 0.99.5 die Gerätedateien ttyS? speziell für eingehende Verbindungen vorgesehen, auf denen ein `getty` Prozess erzeugt wird, während für ausgehende Verbindungen auf dem gleichen Port die Gerätedateien /dev/cua? vorgesehen sind.

/dev/zero

Aus der Spezialdatei /dev/zero können beliebig viele Nullbytes gelesen werden. Diese Datei sollte nur mit Vorsicht als Quelle für Daten benutzt werden, weil die Menge der gelieferten Bytes nicht begrenzt ist.

3.5 Das Verzeichnis /usr

Während die anderen Verzeichnisse der ersten Hierarchie für den "normalen" Anwender wenig interessantes zu bieten haben, eröffnet sich im /usr Verzeichnis die ganze Welt der Linux Anwendungen. Hier ist der X11-Server ebenso zu finden wie das \TeX Satzsystem, der GNU C-Compiler, der emacs Editor oder das News&Mail System. Im Gegensatz zu den anderen Verzeichnissen der ersten Hierarchie, die alle einer genau abgegrenzten Aufgabe zugeordnet werden können, finden sich hier alle möglichen Programme. Trotzdem gibt es auch hier den Ansatz einer verallgemeinerten Struktur im Dateisystem.

/usr/lib

Wie der Name schon andeutet, ist die ursprüngliche Bestimmung dieses Verzeichnisses die einer "Bibliothek".

Der GNU C-Compiler Vor allem die C-Programmbibliotheken haben hier ihren angestammten Platz. Die "traditionelle" Aufteilung wird vom GNU C-Compiler aufgebrochen, weil dieses Entwicklungssystem sehr stark die Verwendung als Crosscompiler unterstützt. Daher ist dieser Compiler mit den dazugehörigen Bibliotheken im Unterverzeichnis gcc-lib zu finden. Entgegen der Regel, daß ein Verzeichnis mit nur einem einzigen Eintrag sinnlos ist, gibt es in diesem Verzeichnis nur einen einzigen Weg vorwärts bis zu /usr/lib/gcc-lib/i386-linux/2.2.2d/ wo schließlich der Preprozessor, der Compiler, der Linker und die Bibliotheken zu finden sind. Diese Aufteilung erhält ihren Sinn erst, wenn zu dem Linux Entwicklungssystem noch eins für die SUN oder für i386-SCO dazukommt, oder wenn eine neue Version des Compilers mit entsprechend neuen Bibliotheken installiert werden soll. Die Verwendung des alten Compilers ist mit einem einfachen Schalter beim Compilertreiber '`gcc -VVersionsnummer`' einzustellen.

Das News&Mail System Wieder aus “Tradition” sind die Konfigurations- und Binärdateien der News- und Mailprogramme in entsprechenden Unterverzeichnissen von /usr/lib untergebracht.

Diverse Hilfs- und Konfigurationsdateien Schließlich befinden sich im /usr/lib Verzeichnis noch eine Reihe Textdateien, in denen die Hilfsmeldungen für die Onlinehilfe verschiedener Programme abgelegt sind, und andere Hilfsdateien im weitesten Sinne; Makros für **groff**, **flex** und **bison** ebenso wie die Wörterbücher für **ispell**.

/usr/include

Das Verzeichnis /usr/include mit seinen Unterverzeichnissen enthält ausschließlich ‘header’ Dateien, die vom C-Preprozessor bearbeitet werden.

Das Verzeichnis /usr/include/linux sollte ein symbolischer Link auf das Verzeichnis /usr/src/linux/include/linux sein; /usr/include/asm sollte in gleicher Weise auf das Verzeichnis /usr/src/linux/include/asm zeigen.

/usr/local

Um eine deutliche Trennung der “originalen” Distribution und der systemspezifischen (lokalen) Erweiterungen zu erreichen, gibt es hinter dem Verzeichnis /usr/local eine Hierarchie von Verzeichnissen, die im Prinzip der von /usr entspricht.

/usr/man

Hier sind die “manual pages” für das **man** Kommandos abgelegt. Es gibt zwei Gruppen von Unterverzeichnissen. Die Namen der einen beginnen mit “man”, und enthalten die unformatierten Manualpages, die von **man** automatisch mit **groff** formatiert werden. Die Namen der anderen beginnen mit “cat”, und enthalten die bereits formatierten Manpages. Die dort abgelegten Dateien können mit **compress** komprimiert sein. Das **man** Kommando dekomprimiert den Inhalt automatisch vor der Ausgabe. Die Verzeichnisse jeder Gruppe sind nummeriert. Die Nummerierung entspricht den in der Erklärung vom **man** Kommando beschriebenen Aufteilung der Manpages auf Seite 110.

/usr/spool

In Verzeichnis /usr/spool werden die Daten der einzelnen News&Mail Programme in entsprechende Unterverzeichnisse abgelegt. Je nach Auslegung des Systems kann hinter dem Verzeichnis /usr/spool/news eine sehr große Hierarchie beginnen, die für jede Nachrichtenrubrik ein Verzeichnis enthält. In diesen Verzeichnissen sind dann die einzelnen Artikel in nummerierten Dateien abgelegt. Im Verzeichnis /usr/spool/uucp werden die für eine Netzwerkkopie mit dem **uucp** Programm bzw. mit dem **uucico** Daemon vorgesehenen Dateien zwischengelagert. Außerdem werden hier traditionell von den Kommunikationsprogrammen die ‘Lockfiles’ zum Sperren der seriellen Schnittstelle angelegt. Wenn ein solches Programm irregulär abgebrochen worden ist, können noch solche ‘LCK.*’ Dateien übrig sein. Mindestens bei jedem Systemstart sollten solche Dateien automatisch gelöscht werden.

Im Verzeichnis /usr/spool/mail sind die Postfächer aller Systembenutzer abgelegt. Jedes Postfach ist eine einfache Datei, in der alle persönlichen Briefe aneinandergehängt sind. Die Programme zur Mailverwaltung können mit diesen Dateien arbeiten, ohne sie durcheinander zu bringen. Die Datensicherheit der persönlichen Post ist wie im Heimatverzeichnis durch die ausschließlich auf den Eigentümer begrenzten Zugriffsrechte weitgehend sichergestellt. Allein der Systemverwalter (root) kann alle Dateien unabhängig von den Zugriffsrechten lesen.

/usr/X386 und /usr/TeX

Wegen des großen Umfangs mancher Distributionen sind Verzeichnisse wie /usr/X386 und /usr/TeX sinnvoll. Hinter /usr/X386 verbirgt sich eine Hierarchie, die wieder prinzipiell wie die /usr Hierarchie aufgebaut ist, und den X11 Server, alle Clients und alle in diesen Zusammenhang gehörenden Dateien enthält.

Hinter /usr/TeX schließt sich eine entsprechende Hierarchie für das **T_EX** Satzsystem an.

3.6 Das Verzeichnis `/home` oder `/usr/home`

Im `/home` Verzeichnis befinden sich alle privaten Verzeichnisse der Benutzer. Jeder Anwender hat hier ein eigenes Heimatverzeichnis. Die Zuordnung eines Heimatverzeichnisses zu einzelnen Benutzern findet in der Datei `/etc/passwd` statt.

Die Dateien `~/.bashrc` und `~/.profile`

Im Heimatverzeichnis jedes Anwenders werden vom Systemverwalter bei der Einrichtung eines Accounts (normalerweise wenigstens eine von) zwei Dateien, `~/.bashrc` und `~/.profile`, abgelegt. Diese Dateien werden von der `bash` bzw. der `sh` Shell beim `login` gelesen und ausgeführt. Sie enthalten Kommandozeilen, wie sie auch über die Tastatur eingegeben werden könnten. Der Sinn dieser Dateien ist es, Kommandos und Einstellungen, die jedesmal für jede Loginshell ausgeführt werden sollen, zu automatisieren. Diese Dateien können vom Benutzer beliebig editiert werden.

Die möglichen Einstellungen, und die Form in der sie durchgeführt werden müssen, ist am besten im Abschnitt über die `bash` nachzulesen.

Kapitel 4

Systemverwaltung

4.1 Das System starten

4.1.1 Von Diskette booten

Wenn im SETUP des Rechners nichts verändert wurde, versucht das BIOS zuerst vom Laufwerk A: das Betriebssystem zu laden. Dabei ist die Wahl des Betriebssystems im Prinzip frei, weil der eigentliche Ladevorgang nicht vom BIOS sondern von dem sogenannten Loader gesteuert wird, der sich im ersten Sektor der Bootdiskette befindet.

Wenn also beim Einschalten des Rechners eine Linux Bootdisk im Laufwerk A: liegt, erscheint nach den normalen Meldungen des BIOS eine Zeile mit der Meldung "Loading" und darauf eine Folge von Punkten, die anzeigt, wieviele Blöcke vom Betriebssystem schon geladen sind.

Wenn der Kernel auf der Diskette gepackt war (ab Version 0.99.6), dekomprimiert er sich automatisch sobald er in den Arbeitsspeicher geladen ist.

Sobald der Kernel die Kontrolle übernommen hat, werden die Geräte und Komponenten initialisiert.

Wenn der Kernel eine Ramdisk anlegt, versucht er diese Ramdisk von der Bootdiskette zu laden. Diese Ramdisk wird dann als Rootfilesystem benutzt, auf das alle weiteren Dateisysteme aufgesetzt werden. Eine Ramdisk kann nur von der Bootdiskette geladen werden.

4.1.2 Von Festplatte booten

Wie bei anderen Betriebssystemen üblich kann auch Linux von der Festplatte gebootet werden. Die Einzelheiten des Bootvorgangs und alle verschiedenen Möglichkeiten Linux von Festplatte zu laden werden in dem Dokument zu Werner Almesbergers "LILO boot loader for Linux" erläutert. Die Lektüre dieses Textes ist jedem Anwender von lilo anzuraten, wenn die erste Festplatte keine standard AT Bus Platte ist wird sie unumgänglich.

Hier soll nur die Verwendung von LILO zum Booten von Linux auf einer primären Partition der ersten Festplatte beschrieben werden.

In dieser Konstellation kann LILO auf der Linuxpartition installiert werden, ohne das Master Boot Record zu verändern. Linux wird dann geladen, wenn die entsprechende Partition aktiviert ist, oder wenn es durch einen bereits installierten interaktiven Bootselector ausgewählt wird.

Das fertig übersetzte LILO Paket befindet sich im Verzeichnis `/etc/lilo`. Die Konfigurationsdatei `/etc/lilo/config` enthält die systemspezifischen Angaben, die vom Systemverwalter angepaßt werden müssen. Mit dem Shellscript `/etc/lilo/install` wird der Bootblock entsprechend den Angaben in der Konfigurationsdatei auf der Festplatte installiert.

Die Konfigurationsdatei enthält Schalter und Variable. Variablen wir durch eine Gleichung ein Wert zugewiesen. Die Konfigurationsdatei besteht aus einem allgemeinen Teil mit allgemeingültigen Angaben und

und beliebig vielen speziellen Teilen in denen jeweils eine Kerneldatei mit einer Konfiguration beschrieben werden.

Unter anderem werden die folgenden Einstellungen in der Konfigurationsdatei vorgenommen:

boot = *Partition* Diese Variable bestimmt die Partition auf die der Bootsektor geschrieben werden soll. Wenn diese Angabe in der Konfigurationsdatei fehlt, wird der Bootsektor auf die aktuelle Rootpartition geschrieben. Die Partition muß eine Linuxpartition sein. Um mit dem normalen Masterbootrecord von MS-DOS davon booten zu können, muß es sich um eine primäre Partition der ersten Festplatte handeln.

install = *Bootsektor* Der *Bootsektor* wird aus der angegebenen Datei gelesen. Falls diese Angabe fehlt, wird der bestehende Bootsektor verändert.

delay = *Zehntelsekunden* Mit der *delay* Variablen kann die Zeit eingestellt werden, die LILO auf eine der unten beschriebenen Tasten wartet, bevor es das erste Kernelimage lädt.

prompt Mit diesem Schalter bringt LILO immer den Bootprompt auf den Bildschirm, auch wenn keine der unten beschriebenen Tasten gedrückt wurde.

timeout = *Zehntelsekunden* Mit dieser Variablen wird die Zeit eingestellt, die LILO nach dem Bootprompt auf eine Eingabe des Systemverwalters wartet, bevor es zum automatischen Laden der ersten konfigurierten Kerneldatei zurückkehrt. Wenn die Variable *timeout* nicht gesetzt ist, wartet LILO endlos auf eine Eingabe.

image = *Kerneldatei* Mit dem *image* Eintrag wird eine *Kerneldatei* (mit absolutem Pfad auf der Rootpartition) angegeben. Dieser Eintrag leitet einen speziellen Teil in der Konfigurationsdatei ein. Alle Einträge bis zum nächsten *image* oder bis zum Dateiende gelten nur für diese Kerneldatei. Die folgenden Kernelspezifischen Einstellungen können vorgenommen werden:

label = *Name* Zur Auswahl einer von mehreren konfigurierten Kerneldateien wird nach dem Bootprompt (siehe unten) entweder die Kerneldatei oder der unter *label* angegebene *Name* angegeben. Mit Hilfe vom *label* kann ein *image* auch mehr als einmal Benutzt werden. LILO speichert die Konfiguration im Bootsektor, nicht in der Kerneldatei.

vga = *Videomodus* Der Kernel wird in dem angegebenen Textmodus gestartet. Die Modi sind bei dem *rdev* Kommando auf Seite 158 beschrieben. Wenn dieser Eintrag in der Konfigurationsdatei fehlt, wird der in der Kerneldatei gespeicherte Videomodus beibehalten.

ramdisk = *Kilobytes* Es wird eine Ramdisk in der angegebenen Größe eingerichtet, unabhängig ob dies beim Übersetzen des Kernels so bestimmt wurde. Siehe auch beim *rdev* Kommando auf Seite 158.

root = *Rootfilesystem* Die Variable *root* enthält den Namen der Gerätedatei für die Festplattenpartition mit dem Rootfilesystem. Wenn anstelle einer Gerätedatei das Wort *current* angegeben ist, wird die aktuelle Rootpartition angenommen. Wenn dieser Eintrag ganz fehlt, wird die beim Übersetzen des Kernels bestimmte Rootpartition beibehalten.

Wenn die speziellen Parameter für alle Konfigurationen gleich sind, können die entsprechenden Einträge auch im allgemeinen Teil vorgenommen werden.

Wenn die Systemspezifikationen in der Konfigurationsdatei eingetragen sind, kann das Installationsscript aufgerufen werden, und so der Bootsektor der Linuxpartition geschrieben werden.

Zum Aktivieren einer primären Festplattenpartition gibt es das **activate** Kommando, das in der Form

activate *Festplatte Partitionsnummer*

aufgerufen wird (zum Beispiel **activate_/_dev/hda_2**)

Wenn beim Booten eine der Tasten ALT, CONTROL, SHIFT gedrückt ist, oder die CAPS LOCK oder SCROLL LOCK Schalter gesetzt sind, bringt LILO einen Prompt auf den Bildschirm, nach dem der Systemverwalter die unter *timeout* bestimmte Zeitspanne zur Verfügung hat, einen bei der Installation des Loaders Konfigurierten alternativen Kernel zum Laden zu bestimmen, mit der Anweisung *root = Rootfilesystem* eine andere Partition zur Wurzel des Dateisystems bestimmen oder mit dem Schlüsselwort *single* den Systemstart im Einbenutzermodus zu erzwingen.

4.1.3 Die Vorgänge bei der Systeminitialisierung

Wenn das Betriebssystem fertig geladen ist, muß der Textmodus für die virtuellen Konsolen eingestellt werden. Wenn eine andere als die RETURN Taste gedrückt wird, bleibt der normale Textmodus mit 25 Zeilen und 80 Zeichen pro Zeile eingestellt. Dieser Modus funktioniert mit allen Grafikkarten problemlos. Wenn die RETURN Taste gedrückt wird, können auch andere Textmodi gesetzt werden, wenn die Grafikkarte diese Modi unterstützt.

Diese Abfrage kann durch einen (im Makefile) fest eingestellten Modus mit einem neu kompilierten Kernel und durch entsprechende Konfigurierung des LILO übersprungen werden.

Nachdem der gewählte Textmodus eingeschaltet, und der Bildschirm gelöscht worden ist, werden die von Linux automatisch erkannten Systemkomponenten angezeigt und initialisiert. Dieser Vorgang wird auf dem Bildschirm dokumentiert, kann aber nicht direkt beeinflußt werden. Der mit einer Distribution ausgelieferte Kernel erkennt ein sehr breites Spektrum an Hardware ohne Probleme; auch in ungewöhnlichen Kombinationen. Trotzdem können natürlich im Einzelfall Probleme auftauchen, die aber in der Regel nicht zum Totalausfall des Systems führen, sondern nur einzelne Teile betreffen. In so einem Fall kann das Problem meist dadurch behoben werden, daß der Kernel mit den konkreten Parametern der Konfiguration neu übersetzt wird. Wenn das System überhaupt nicht hochfährt, oder die Sourcen zum Kernel nicht vorhanden sind, muß der Kernel auf einem anderen System mit den notwendigen Anpassungen neu übersetzt werden.

Die Initialisierung des Kernels läuft in den folgenden Schritten ab:

In der ersten Phase des Hochfahrens werden die internen Funktionen und Tabellen des Kernels initialisiert. Dazu gehören die Tabelle zur Speicherseitenverwaltung, die Belegung der Fehlerinterrupts und der IRQ, die Einrichtung der Prozeßtable und der Start der Schedulers. Dabei findet noch keine Bildschirmausgabe statt.

Danach werden die zeichenorientierten Geräte initialisiert. Zuerst werden die seriellen Schnittstellen gesucht. Normale Schnittstellenkarten mit zwei Ports werden von jedem Linux Kernel ohne Probleme erkannt. Die 16550A Schnittstellenbausteine werden unterstützt. Nur bei Systemen mit mehr als zwei Ports kann es unter Umständen Probleme mit der automatischen IRQ Erkennung geben. In diesem Fall können die entsprechenden Parameter aber ohne großen Aufwand fest in den Kernel einkompiliert werden.

Im nächsten Schritt sucht und initialisiert der Kernel die parallele(n) Druckerschnittstelle(n).

Dann wird der Treiber für die 'normalen' (IDE, RLL, MFM) Festplatten mit den Parametern aus dem BIOS initialisiert. Wenn (wie beim Kernel auf der Installationsbootdisk) eine Ramdisk eingerichtet wird, findet deren Initialisierung auch in diesem Schritt statt.

Danach sucht der Kernel nach SCSI Hostadaptoren, und wenn er welche gefunden hat, wird nach SCSI Geräten gesucht. Auch hier werden einigermaßen unkomplizierte Konfigurationen automatisch erkannt. Wenn es Probleme mit einem Hostadapter oder einer Festplatte gibt, die prinzipiell unterstützt wird, aber in der Konfiguration nicht korrekt erkannt wird, kann auch hier der Kernel mit den entsprechenden Parametern neu übersetzt werden, um das Problem zu lösen.

In den folgenden Schritten wird der Arbeitsspeicher und eine Reihe von internen Tabellen und Parametern initialisiert, ohne daß es auf dem Bildschirm dokumentiert wird.

Dann werden die Diskettenlaufwerke untersucht und initialisiert.

Im nächsten Schritt werden die Sockets für den Kernel und das TCP/IP Netzprotokoll initialisiert. Dabei ist eine Ethernetkarte nicht notwendig. In jedem Fall wird ein Loopback Device eingerichtet, mit dem die verschiedenen Netzwerkprogramme auch auf einem Rechner arbeiten können.

Schließlich wechselt der Kernel selbst in den Benutzermodus und startet die ersten Tasks.

Der "nullte" Task des Kernels ist der idle Prozess. Seine einzige Bestimmung ist es, die überflüssige Rechenzeit zu verbrauchen.

Das erste Programm, das der Kernel ausführt ist /etc/init, oder /bin/init, wenn es /etc/init nicht gibt.

Nur wenn es weder in /etc noch in /bin ein ausführbares 'init' Programm gibt, startet der Kernel selbst eine Shell, die die in /etc/rc aufgeführten Kommandos ausführt, und anschließend eine Shell mit Rootrechten, in der ein Benutzer interaktiv arbeiten kann.

4.1.4 init

Auch bei einem multiuser multitasking Betriebssystem muß irgendein Task, irgendein User den Anfang machen. Dieser Anfang ist der init Prozeß, der vom Kernel selbst gestartet wird. init wird auch oft als der Vater aller Prozesse im System bezeichnet. Es ist so eine Art Metab Benutzer. Der init Prozeß startet eine Reihe weiterer "Benutzerprozesse" ohne direkte Benutzer, und er sorgt dafür das bestimmte Prozesse oder Dienste immer zur Verfügung stehen.

Der Name 'init' deutet auf eine Hauptaufgabe des Programms hin: Es initialisiert das System. Diese Initialisierung kann vollständig vom Systemverwalter konfiguriert werden.

Die zentrale Konfigurationsdatei für init ist `/etc/inittab`.

Für Linux sind zwei sehr verschiedene Versionen von init verbreitet:

Das einfache init (simpleinit) von `poe@daimi.aau.dk` mit Erweiterung von Werner Almesberger läßt nur den Einbenutzermodus (single user) und einen Mehrbenutzermodus zu. Beim Übergang in den Mehrbenutzermodus bearbeitet es die Systeminitialisierungsdatei `/etc/rc` und anschließend startet es die `getty` Prozesse für die virtuellen Terminals mit den Parametern die in der Datei `/etc/inittab` eingestellt worden sind.

Das vielseitigere und damit natürlich auch kompliziertere System 5 kompatible init Programm von Miquel van Smoorenburg erlaubt eine Vielzahl von Systemzuständen, sogenannte Runlevel, bei denen die unterschiedlichsten Prozesse oder Dienste gestartet und Initialisierungen durchgeführt werden.

Simpleinit

Wie bereits oben gesagt, bietet das simpleinit zwei Modi an. Im Einbenutzermodus wird nur eine einzige Shell auf der Konsole gestartet. Auf den anderen virtuellen Terminals kann nicht gearbeitet werden. Das System startet im Einbenutzermodus, wenn die Datei `/etc/singleboot` existiert. Wenn außerdem die Datei `/etc/securesingle` existiert, muß das korrekte Rootpaßwort eingegeben werden, bevor init die Shell öffnet.

Beim Systemstart im Einbenutzermodus wird die Systeminitialisierungsdatei `/etc/rc` nicht bearbeitet.

Wenn die Einbenutzershell verlassen wird, fährt das System automatisch in den Mehrbenutzermodus hoch. Erst beim Übergang in den Mehrbenutzermodus wird die Systeminitialisierungsdatei `/etc/rc` abgearbeitet. Diese Datei wird als Shellsript an die Standardshell `/bin/sh` übergeben. Außer den `getty` Prozessen für die Terminals werden vom simpleinit nach der Ausführung des Shellscripts keine weiteren Programme ausgeführt. Das bedeutet, daß alle darüberhinaus notwendigen Systeminitialisierungen in diesem Shellsript durchgeführt werden müssen.

Um ein korrekt lauffähiges System zu erhalten, müssen eine Reihe von Prozessen und Diensten "im Hintergrund" laufen.

Ein wichtiges Kommando dieser Art ist `'/etc/update &'`, das im Hintergrund während der gesamten Lebenszeit des Systems in regelmäßigen Abständen eine Synchronisierung des Dateisystems auslöst. Dieses Programm muß von init gestartet werden.

Eine zweite unbedingt notwendige Aufgabe des `/etc/rc` Scripts ist das Zusammensetzen des Dateisystems. Dazu muß das `mount` Kommando aufgerufen werden. Wenn die Datei `/etc/fstab` genau der Systemkonfiguration entspricht, reicht es, wenn das Kommando `'/etc/mount -a'` in der Scriptdatei steht. Sonst kann natürlich auch jede Partition einzeln aufgesetzt werden.

Außerdem sollte im `/etc/rc` Shellsript eine vorhandene Swappartition mit dem `swapon` Kommando aktiviert werden.

Einige Dienste werden von Dämonen ausgeführt. Diese Programme bleiben im Hintergrund und warten auf bestimmte Ereignisse, um bei deren Eintreten bestimmte Aktionen auszuführen. Beispiele für häufig verwendete Dämonen sind der `cron` Dämon, der in regelmäßigen abständen aufwacht und in einer Datei (z.B. `/usr/spool/cron/crontab/*` für `vixie-cron`) in einem Zeitplan nach Kommandos sucht, die zur aktuellen Zeit ausgeführt werden sollen; oder der Druckerdämon `lpd`, der die Druckjobs aller Anwender verwaltet und auf die im System vorhandenen Drucker verteilt.

Auch diese Dämonen sollten von init gestartet werden.

Schließlich ist es noch eine der Standardaufgaben der `/etc/rc` Datei, alle möglichen Reste aus vergangenen Laufzeiten aufzuräumen. Dazu gehören die Lockfiles z.B. in `/usr/spool/uucp`, mit denen ein Modemport

blockiert wird, oder die Datei `/etc/nologin`. Die Datei `/etc/utmp`, die eine Liste der aktiven Benutzer enthält, sollte nicht gelöscht werden, aber mit dem Kommando `'cat /dev/null > /etc/utmp'` geleert werden.

Nachdem das `/etc/rc` Shellsript abgearbeitet ist, liest das einfache `init` die `/etc/inittab` Datei, in der ausschließlich die `getty` Prozesse gestartet werden, mit denen das Einloggen auf den verschiedenen Terminals erst möglich wird. Das Format dieser Datei ist bei der Reise durchs Dateisystem beschrieben worden.

Jeder dieser `getty` Prozesse wird in einer Art Endlosschleife immer wieder erzeugt, wenn die Loginshell auf einem der Terminals verlassen wird. Das bedeutet auch, daß der `init` Prozeß niemals wirklich beendet wird. Er läßt sich auch mit einem `'kill -9 1'` Signal nicht abschießen.

Es besteht aber die Möglichkeit, mit einem `'kill -1 1'` den `init`-Prozeß zu veranlassen, die `/etc/inittab` neu zu lesen und entsprechend den dann gefundenen Zeilen die `getty`-Prozesse neu zu starten. Noch vorhandene alte `getty` werden vom einfachen `'init'` nicht automatisch beendet. Diese Prozesse müssen einzeln direkt vom Superuser (`root`) beendet werden.

Sysvinit

Wie bereits weiter oben gesagt, bietet das System 5 kompatible `init` durch das System der Runlevel eine sehr flexible Methode zur Systemkonfiguration. Die Runlevel werden vom Systemverwalter mit dem `telinit` Kommando geschaltet. In jedem Runlevel werden bestimmte Dienstprogramme und Dämonen gestartet, andere werden beim Übergang in einen neuen Modus beendet, und es können für jeden Runlevel eigene Konfigurationsscripts aufgerufen werden.

In welchem Runlevel welche Programme laufen, und welche Initialisierungsdateien abgearbeitet werden sollen, das wird in der Datei `/etc/inittab` (eventuell auch `sysvinitab`) festgelegt. Diese Datei enthält Datensätze, deren Bedeutung bei der Reise durch das Dateisystem auf der Seite 165 beschrieben wurde.

Zur Änderung des Runlevel gibt es das `telinit` Kommando, das als einzigen Kommandozeilenparameter den Namen oder das Zeichen (Buchstabe oder Ziffer) des gewünschten Runlevel erwartet. Dieses Kommando kann nur vom Superuser (`root`) ausgeführt werden.

4.2 Das System abschalten

Der komplexen Einschaltprozedur entspricht die Prozedur des Abschaltens, oder Herunterfahrens des Systems. Im Mehrbenutzerbetrieb ist dabei nicht nur auf die Integrität des Dateisystems zu achten, sondern selbstverständlich auch auf eventuell gerade aktive Anwender Rücksicht zu nehmen. Es ist also definitiv nicht damit getan, den Rechner abzuschalten.

Um dem Systemverwalter das Herunterfahren des Systems zu erleichtern, gibt es das `shutdown` Programm mit den Varianten `reboot`, `halt`, `fastboot` und `fasthalt`, die alle Links auf `shutdown` sind. Eine Beschreibung ist auf Seite 159 des Handbuchs zu finden.

Wenn aus irgendwelchen Gründen das System nicht mit `shutdown` angehalten werden kann, ist auf jeden Fall eine Synchronisation des Dateisystems mit dem `sync` Kommando vor dem Abschalten notwendig! Um als Superuser (`root`) die übrigen aktiven Benutzer vor einer bevorstehenden Abschaltung zu warnen steht das `wall` Kommando zur Verfügung. Um andere Prozesse, zum Beispiel Dämonen vor dem Abschalten zu beenden kann das `kill` Kommando verwendet werden, das auf Seite 101 des Handbuchs beschrieben ist.

Mit dem `sysvinit` hat der Superuser die Möglichkeit, das System vor dem Abschalten in den Einbenutzermodus herunterzufahren, indem er dem `telinit` Kommando den Parameter `'1'` übergibt.

Dateisystem abbauen

Mit dem `'umount -a'` Kommando kann das komplette Dateisystem in der umgekehrten Reihenfolge abgebaut werden, in der es beim Systemstart von `'mount -a'` aufgebaut worden ist. Das `umount` Kommando führt dabei selbst den `sync` Systemaufruf aus, der das Dateisystem mit dem Blockdepot synchronisiert.

Dabei dürfen aber die abzusetzenden Dateisysteme nicht "aktiv" sein. Das bedeutet, daß kein Dateisystem auf einem Verzeichnis dieses Systems aufgesetzt sein darf, daß sich kein Benutzer in einem Verzeichnis dieses Systems aufhalten darf. Das bedeutet auch, daß Programme deren Arbeitsverzeichnis auf der betreffenden Partition liegt vor dem Absetzen des Dateisystems beendet werden müssen.

4.3 Prozeßordnung

Alle Benutzeraktivitäten unter Linux finden in irgendwelchen Prozessen statt. Ein solcher Prozess besteht nicht nur aus einem Programm im Speicher. Als Prozess wird die Gesamtheit aller Systemdateidaten zu einem Programm gezählt. Dazu gehört die Prozessumgebung, die von dem Programm geöffneten Dateien (auch Terminals), das Arbeitsverzeichnis, ein Eigentümer mit samt einer Gruppe und schließlich kann ein Prozess noch Kinder haben.

Ein Prozess entsteht, indem ein anderer Prozess sich mit dem `fork` Systemaufruf teilt. Dabei wird der aufrufende Prozesses kopiert und dabei ein neuer Eintrag in der Prozesstabelle erzeugt. Der Scheduler wählt aus dieser Prozesstabelle jeweils den Prozess mit der höchsten Priorität aus und teilt ihm Rechenzeit zu. Wenn der neue Eintrag das erste mal vom Scheduler aufgerufen wird, also Rechenzeit zugeteilt bekommt, wird das aufrufende Programm durch das neu aufgerufene verdrängt, das in der Umgebung des aufrufenden weiterläuft. Das aufrufende Programm wird als Vater bezeichnet, das daraus entstehende Programm Kind.

Der erste Prozess und damit Vater aller weiteren Prozesse ist das bereits beschriebene `init` Programm. Die daraus entstandenen `getty` Programme erzeugen `login` die ihrerseits eine Benutzershell starten. Wenn ein Anwender in dieser Shell ein Kommando aufruft wird die Shell wieder zum Vater und das aufgerufene Kommando zum Kind. Häufig wartet ein Vater auf die Beendigung des Kindprozesses; das ist zum Beispiel der Fall wenn die Shell ein Kommando nicht im Hintergrund ausführt.

Die Prozeßtabelle kann mit dem `ps` Kommando angesehen werden. Daraus läßt sich unter anderm die Prozeßnummer (PID) und die Nummer des Vaterprozesses (PPID) ermitteln.

In der Prozeßtabelle werden außerdem die Benutzer- und Gruppennummer (UID, GID) des aufrufenden Benutzers sowie gegebenenfalls davon abweichende effektive Benutzer- und Gruppenkennung (EUID, EGID) festgehalten. Die effektiven Kennungen können durch Änderung der entsprechenden Zugriffsrechte auf die ausführbare Programmdatei geändert werden.

Prozesse können Signale senden und empfangen, und auf diese Weise miteinander kommunizieren. Der Benutzer hat die Möglichkeit mit dem `kill` Kommando an dieser Kommunikation aktiv Teilzunehmen.

4.3.1 Abstürzende Programme und hängende Prozesse

Es gibt kein fehlerfreies Programm, und überall Benutzer, die einem sonst sehr zuverlässigen Programm ganz erstaunliches und unerklärliches Verhalten entlocken, und es gibt das Naturgesetz von Murphy...

Deshalb kommt es mit Sicherheit bei jedem Rechner irgendwann einmal zu einem Programmabsturz.

Aber nicht jedes fehlerhafte Programm führt gleich zu einem kompletten Systemabsturz. Durch die Architektur des Betriebssystems hat jedes Programm seinen eigenen, geschützten Speicherbereich, in den kein anderes Programm schreiben oder hineinsehen darf. Beim Versuch, auf den Speicherbereich eines anderen Programms zuzugreifen, wird jedes Programm sofort mit dem `SIGSEGV` Signal abgebrochen. Dieses Programm wird dadurch sofort aus dem Arbeitsspeicher gelöscht und kann keinen Schaden mehr anrichten.

Wenn ein Programm wegen eines solchen Fehlers vom Betriebssystem abgebrochen werden muß, wird automatisch ein Speicherabzug (`core`) des gesamten Speicherbereiches des Prozesses auf der Festplatte im aktuellen Verzeichnis abgelegt. Die maximale Größe dieses Speicherabzuges kann mit dem `ulimit` Kommando begrenzt werden.

Viel öfter passiert es, daß ein Prozeß sich "aufhängt". So ein Prozeß arbeitet noch irgendwie, ist aber über die Tastatur nicht mehr ansprechbar. Oder die Ausgabe des Programms findet den Weg zum Bildschirm nicht mehr. Wenn nichteinmal ein `^C` mehr hilft, können solche Prozesse bei einigen anderen Betriebssystemen nur durch einen Reset des Rechners beendet werden. Bei Linux (oder Unix) kann im Prinzip jeder Prozeß von einem anderen Terminal aus beendet werden, indem ihm mit dem `kill` Kommando ein Signal zum Aufhören gesendet wird. Das `kill` Kommando kann jeder Anwender nur für seine eigenen Prozesse benutzen. Nur der Superuser (`root`) kann alle Prozesse killen.

Der drastische Ausdruck trifft nicht genau den Kern von `kill`, denn es gibt eine ganze Reihe von Signalen, die dem Prozeß der sie erhält nicht gleich das ganze Leben entzieht. Vielmehr können viele dieser Signale von den Programmen, die sie erhalten, 'abgefangen' und sinnvoll bearbeitet werden. Um dem Benutzer (und

dem Programm) die Möglichkeit einer differenzierten Reaktion zu geben, stehen für das `kill` Kommando 23 verschiedene Signale zur Verfügung.

Das Standardsignal ist `SIGTERM` (15), die Aufforderung zu terminieren. Andere häufig verwendete Signale sind `SIGHUP` (1) das die Unterbrechung einer Terminalverbindung signalisiert oder `SIGINT` (2) das auch durch die Tastenkombination `^C` erzeugt wird, und in vielen Programmen von einer Routine abgefangen wird.

In der Regel werden die Programme "freiwillig" die Bühne verlassen, wenn sie von `kill` ein solches Signal erhalten haben. Erst wenn das nichts mehr hilft, kann mit dem Signal `SIGKILL` (9) jeder (eigene) Prozeß ohne die Möglichkeit einer Reaktion beendet werden.

Das `kill` Kommando benötigt natürlich in der Kommandozeile eine Identifikation des Prozesses, an den das Signal geschickt werden soll. Dazu kann jedes `kill`-Kommando die Prozeßnummer dieses Prozesses verarbeiten. Um diese Prozeßnummer (PID) herauszubekommen, kann das `ps` Kommando benutzt werden. Mit der Option `'-a'` zeigt es alle Prozesse mit ihren Prozessnummern, den Terminals auf denen sie laufen dem Status und dem Namen.

4.4 Systemabsturz

Was für die Anwenderprogramme gilt, ist leider auch für den Kernel selbst nicht verkehrt. Auch das Betriebssystem ist nicht auf alle möglichen Ausnahmen vorbereitet, und es enthält auch Fehler. Ein Fehler des Betriebssystems ist nicht so leicht abzufangen wie der eines Anwenderprogramms. In einem solchen Fall kommt es in der Regel zu einem Systemabsturz; manchmal in Form eines "Kernel Panic" Systemhalts, manchmal zu einem Reset, manchmal zu einem kompletten Systemstillstand.

Wenn sich ein Systemabsturz irgendwie ankündigt, indem beispielsweise das System immer langsamer wird, obwohl kein rechenzeitintensives Programm läuft, können einige Maßnahmen helfen, den Schaden zu begrenzen.

Natürlich sollten alle Anwender ihre Arbeit mit dem Rechner sofort beenden, und vor allem alle geöffneten Dateien schließen. Danach sollte unbedingt ein `sync` Kommando ausgeführt werden, weil nur auf diese Weise die Daten tatsächlich auf der Festplatte gesichert werden.

Wenn das noch möglich ist, sollte der Rechner durch den `reboot` oder `halt` Befehl vom Superuser (`root`) angehalten werden. Weil dieses Kommando sofort wirkt, sind die in diesem Moment z. B. in einem Editor geöffneten Dateien meist verloren. Trotzdem ist diese Methode dem unkontrollierten Systemabsturz vorzuziehen.

Wenn der Rechner unkontrolliert ausgeschaltet wurde, oder wenn es zu einem Absturz wegen eines internen Fehlers gekommen ist, muß damit gerechnet werden, daß das Dateisystem Schaden genommen hat, und Daten verloren gegangen sind. Aus diesem Grund muß unbedingt nach einem solchen Vorfall das Dateisystem mit dem (e)`fsck` Programm überprüft und wenn nötig repariert werden.

4.5 Die Konsistenz des Dateisystems prüfen

In gewissen Abständen, jedenfalls nach einem Absturz oder einer unkontrollierten Abschaltung des Rechners, muß die Konsistenz des Dateisystems überprüft, und im Fehlerfall repariert werden. Zu diesem Zweck existieren die Hilfsprogramme `fsck` und `efsck`, die bestimmte Komponenten des Dateisystems prüfen, und nach Möglichkeit auch reparieren.

Während der Arbeit am Dateisystem dürfen keine Benutzer im System sein. Die (e)`fsck` Programme müssen im Rootfilesystem liegen, und alle aufgesetzten Partitionen müssen abgesetzt werden.

Es wird nicht das gesamte Dateisystem auf einmal geprüft, sondern jede Partition einzeln. Je nach Typ des Dateisystems muß das Programm `fsck` (für Minix Partitionen) oder `efsck` (für "Erweiterte" Partitionen) mit der der Partition entsprechenden Gerätedatei aus `/dev` aufgerufen werden.

Zuerst werden die einzelnen Partitionen nur getestet, indem das entsprechende (e)`fsck` Kommando mit der `'-v'` Option aufgerufen wird. Wenn keine Fehler festgestellt werden, meldet das Programm die Anzahl der benutzten Inodes und Datenblöcke, sowie die Anzahl der Dateien, Verzeichnisse, Spezialdateien und Links.

Wenn aber ein Fehler festgestellt wird, zeigt das Programm die Art des Fehlers an.

Mit der ‘-r’ Option können die Fehler einzeln interaktiv Behoben werden.

Die ‘-a’ Option veranlaßt das (e)fsck Programm, die gefundenen Fehler ohne Nachfrage automatisch zu beheben.

4.5.1 Das Rootfilesystem reparieren

Das Rootfilesystem kann selbstverständlich nicht abgesetzt werden. Eine einfache Überprüfung mit der ‘-v’ Option kann auch für das aufgesetzte Rootfilesystem durchgeföhrt werden. Erst wenn tatsächlich ein Fehler gefunden wurde, der repariert werden muß, ist ein Griff in die Trickkiste nötig.

Das Problem besteht darin, daß bei einem aufgesetzten Dateisystem immer eine Kopie des Superblockes und einer ganzen Reihe von Datenblöcken und Inodes im Cachespeicher (dem Blockdepot) gehalten werden. Wenn das Dateisystem auf der Festplatte verändert wird, darf keine Synchronisation mit dem Blockdepot mehr stattfinden, weil ja sonst die fehlerhaften Daten wieder auf die Festplatte geschrieben würden. Um dieses Problem zu lösen, gibt es zwei Möglichkeiten.

Eine Möglichkeit, das Rootfilesystem zu reparieren, besteht darin, das System mit einer “bootable Rootdisk” zu starten, die eine Ramdisk als Rootfilesystem anlegt. Von dieser Ramdisk aus kann dann das Rootfilesystem auf der Festplatte praktisch von außen repariert werden.

Eine zweite Möglichkeit besteht darin, den `update` Prozeß mit einem `SIGKILL` zu beenden, und dann unmittelbar nach Beendigung des (e)fsck Programms ohne einen `sync` Befehl das System durch Betätigen des RESET Knopfes neu zu booten.

4.6 Benutzer eintragen

Die Notwendigkeit einer Benutzerverwaltung liegt bei Mehrbenutzersystemen auf der Hand. Aber auch wenn Linux nur von einer einzigen Person benutzt wird, ist unbedingt anzuraten, mindestens einen ‘normalen’ Benutzer einzutragen. Beim alltäglichen Arbeiten unter Rootrechten sind die meisten Sicherheitsvorkehrungen des Kernels ganz abgeschaltet, oder sie lassen sich leicht umgehen, eventuell auch unbeabsichtigt. Die auf diese Weise entstehenden Fehler können leicht das gesamte System betreffen.

Es gibt zur Unterstützung des Systemverwalters Dienstprogramme, die das Ein- und Austragen von Benutzern automatisieren. Besonders bei Installationen mit dem Shadowpaßwortsystem sind Programme wie `useradd`, `userdel`, `groupadd`, `groupdel` u. a. sicher eine große Hilfe. Weil aber diese Programme die grundlegenden Zusammenhänge verdecken soll hier der Vorgang für das normale Paßwortsystem im Einzelnen beschrieben werden. Die Verwendung der Dienstprogramme kann in den englischen Manualpages nachgelesen werden.

Zum Eintragen eines neuen Benutzers gehören:

1. der Eintrag in der Paßwortdatei
2. wenn nötig der Eintrag in die Gruppendatei
3. das Erstellen des Heimatverzeichnis mit den Initialisierungsdateien

4.6.1 Eintrag in `/etc/passwd`

Die zentrale Benutzerdatenbank ist die Datei `/etc/passwd`. In dieser Datei muß jeder Anwender aufgeföhrt sein damit er sich einloggen kann. Diese Datei enthält normal lesbaren Text, und kann deshalb mit jedem Editor bearbeitet werden.

Für jeden Anwender muß der Systemverwalter hier eine Zeile mit sieben Feldern anlegen. Die Felder werden durch einen Doppelpunkt ‘:’ voneinander getrennt. Die Felder haben folgende Bedeutung:

Benutzername:Paßwort:Benutzernummer:Gruppennummer:Name:Heimat:Shell

- Das erste Feld enthält den Benutzernamen. Dieser Name darf aus beliebigen druckbaren Zeichen bestehen. Allerdings ist es sinnvoll und üblich, nur Kleinbuchstaben für die Benutzernamen zu verwenden.

- Das zweite Feld enthält das verschlüsselte Paßwort. Wenn dieses Feld leer bleibt, ist der Account durch kein Paßwort gesichert, also frei zugänglich. Wenn anstelle eines korrekt verschlüsselten Paßwortes ein Klartextwort in die Datenbank geschrieben wird, ist unter dem Benutzernamen kein Login möglich.

Jedem Benutzer steht das `passwd` Kommando zur Verfügung, um sein Paßwort selbst zu verändern. Dieses Programm läuft “SUID root”, das heißt während der Ausführung dieses Programms erhält auch der einfache Anwender Rootrechte.

Beim Neuentrag eines Benutzers wird dieses Feld freigelassen, und unmittelbar nach der vollständigen Eintragung muß der Benutzer oder der Systemverwalter unter dem neuen Benutzernamen das `passwd` Kommando aufrufen und ein Paßwort eingeben.

- Das dritte Feld enthält die Benutzernummer (UID). Die Nummer ist eine beliebige nichtnegative Zahl (bis 64000). Um genügend Benutzernummern für Verwaltungsaccounts offen zu halten sollten die eigentlichen Anwender Benutzernummern größer als 99 erhalten. Jede Benutzernummer darf nur einmal verwendet werden.
- Das vierte Feld enthält die Gruppennummer (GID) einer Benutzergruppe. Die hier angegebene Gruppe ist die Standardgruppe dieses Anwenders. Durch einen entsprechenden Eintrag in der Datei `/etc/group` kann jeder Benutzer auch Mitglied anderer Gruppen werden.
- Das fünfte Feld enthält einen Kommentar, in der Regel den vollständigen Namen des Benutzers. Dieser Kommentar wird von vielen News&Mail Programmen benutzt, die daraus den Namen des Absenders lesen.
- Das sechste Feld enthält den absoluten Pfadnamen des Heimatverzeichnisses des Anwenders.
- Das siebente Feld enthält den Namen des Programms, das von `login` nach erfolgreicher Anmeldung gestartet werden soll. Der Benutzer kann mit dem Programm `chsh` diesen Eintrag selbst ändern. Die möglichen Programme (Shells) sind in der Datei `/etc/shells` aufgelistet.

Wenn ein Benutzer keine interaktive Shell haben soll, kann auch ein beliebiges anderes Programm mit allen Argumenten in dieses Feld eingetragen werden. Zum Beispiel kann hier für UUCP Accounts der `uucico` Daemon aufgerufen werden. Der Account “sync” ist ein anderes Beispiel, mit dem Sinn auch von außerhalb die Synchronisation des Dateisystems mit dem Blockdepot auslösen zu können.

Das letzte Feld kann auch leer bleiben. Dann wird automatisch die Standardshell `/bin/sh` gestartet. Das siebente Feld wird nicht durch einen Doppelpunkt, sondern durch ein Zeilenende abgeschlossen.

4.6.2 Gruppenzwang

In der Datei `/etc/group` werden die Benutzergruppen für das System festgelegt. Für jede Benutzergruppe existiert eine Zeile in dieser Datei. Jede Zeile besteht aus vier Feldern, die durch einen Doppelpunkt (‘:’) voneinander getrennt sind.

- Das erste Feld enthält den Namen der Gruppe.
- Das zweite Feld kann das Verschlüsselte Paßwort für den Gruppenzugang enthalten. Das einfache `newgrp` Kommando für Linux benutzt diesen Eintrag aber nicht, sondern erlaubt nur den eingetragenen Benutzern das Wechseln der Gruppenrechte.
- Das dritte Feld enthält die Gruppennummer (GID). Die Nummer ist eine beliebige nichtnegative Zahl (bis 64000). Jede Gruppennummer darf nur ein mal verwendet werden.
- Das vierte Feld schließlich enthält eine durch Kommata getrennte Liste der Namen aller Gruppenmitglieder.

Die in dieser Datei eingetragenen Gruppenmitglieder werden nicht nach einem Paßwort gefragt, wenn sie mit dem `newgrp` Kommando die Gruppenidentität wechseln wollen. Anwender ohne Eintrag müssen sich mit

dem Paßwort legitimieren, bevor sie die Rechte dieser Gruppe erhalten (wenn das `newgrp` Kommando diese Möglichkeit bietet).

Aus Gründen der Systemsicherheit ist es üblich, anstelle eines verschlüsselten Paßwortes einen Klartexteintrag in das zweite Feld zu schreiben (z. B. `VOID` oder `*`). Dadurch ist die Benutzung der Gruppenrechte für nicht eingetragene Benutzer gesperrt.

4.6.3 Das Heimatverzeichnis anlegen

Jeder Anwender braucht sein eigenes Heimatverzeichnis. Dieses Verzeichnis muß vom Superuser eingerichtet und in die `/etc/passwd` Datei eingetragen werden. Der Name des Verzeichnisses kann dem Benutzernamen entsprechen, muß es aber nicht.

Das Verzeichnis muß dem Benutzer und seiner Standardgruppe gehören; dazu dient das Kommando `chown -R Benutzer.Gruppe Heimatverzeichnis`. Die Zugriffsrechte sollten beim Einrichten mit dem Modus `1700` initialisiert werden, um dem Anwender die größtmögliche Datensicherheit zu gewährleisten. Dazu wird das Kommando `chmod 1700 Heimatverzeichnis` aufgerufen. Es steht jedem Benutzer frei, die Zugriffsbeschränkungen auf sein eigenes Verzeichnis zu lockern.

In dem Verzeichnis sollten Standardinitialisierungsdateien für die wichtigsten Shells und unter Umständen auch für weitere Programme, wie z. B. einen X11 Windowmanager angelegt werden.

Wenn ein Mailsystem installiert ist, kann es erforderlich sein, die in der Shellvariablen `MAIL` (`/usr/spool/mail/name`) angegebene Datei mit den Eigentumsrechten des Benutzers zu erzeugen.

4.7 Die “Sicherheit” des Systems

4.7.1 Eigentum und Zugriffsrechte

In einem Betriebssystem, das mehrere Anwender zuläßt, taucht die Frage nach der Datensicherheit schneller auf als bei einem Einbenutzersystem. Es sollte möglich sein, Daten vor unberechtigt Zugriff zu schützen. Dabei sind zum Einen die privaten Daten vor fremden Benutzern zu verbergen; zum Anderen ist es auch sehr sinnvoll, den Systembereich prinzipiell vor Veränderungen zu schützen. Um ein versehentliches Löschen oder Überschreiben von Dateien zu verhindern, bieten die meisten Betriebssysteme die Möglichkeit des nur-lesen Status. Dieser Mechanismus reicht im Mehrbenutzersystem nicht aus. Linux bietet deshalb ein dreistufiges System der Zugriffsberechtigung an. Es wird unterschieden zwischen:

- dem Eigentümer einer Datei
- einer Benutzergruppe der die Datei zugeordnet wird
- allen anderen Benutzern

Der Begriff des Eigentümers macht nur Sinn in Verbindung mit der Identifikation jedes Benutzers beim Login. Auf diese Weise kann jede Datei eindeutig ihrem Erzeuger zugeordnet werden.

Neben seinem eindeutigen Benutzernamen ist jeder Anwender auch mindestens einer Gruppe zugeordnet. Die Gruppenkennung wird vom Systemverwalter in der Paßwortdatei zusammen mit dem Benutzernamen abgespeichert. Dieser Gruppe wird die Datei bei ihrer Erzeugung ebenfalls zugeordnet.

Nur dem Eigentümer einer Datei ist es möglich, die Zugriffsrechte zum Lesen, Schreiben und zum Ausführen der Datei beliebig für die drei Benutzerkategorien zu setzen. Wenn der Eigentümer sich selbst das Schreiben in seine Datei verbietet, ist sie vor versehentlicher Änderung geschützt. Die Zugriffsrechte auf die Dateien werden nicht in der Datei selbst gespeichert, sondern in den Inodes. Deshalb kann der Eigentümer auch die Zugriffsrechte für eine Datei ändern, die er nicht beschreiben kann.

Die Ausführbarkeit macht nur bei binären Programmdateien oder bei Shellsripten Sinn. Shellscripte sind Textdateien mit Befehlen, die Zeile für Zeile von der Shell abgearbeitet werden. Shellscripte müssen zusätzlich lesbar sein, damit sie von der Shell bearbeitet werden können. Beim Versuch, irgendeine andere, nicht ausführbare Datei zur Ausführung zu bringen, wird das Kommando mit einer Fehlermeldung abgebrochen.

Eine Spezialität der *NIX Betriebssysteme ist die Möglichkeit, ein Programm mit den Rechten und Privilegien des Eigentümers oder einer Gruppe ausführen zu können. Auf diese Weise kann beispielsweise jeder Benutzer mit dem `passwd` Kommando sein eigenes Paßwort verändern, ohne selbst Schreibberechtigung für die `/etc/passwd` Datei zu haben. Das `passwd` Kommando gehört dem Superuser (root) und hat das SUID Bit im Berechtigungsfeld gesetzt. Dadurch hat jeder Benutzer die effektive User ID von root, solange er das `passwd` Kommando ausführt. Das `passwd` Kommando sorgt selbstverständlich dafür, daß kein Anwender ein anderes als sein eigenes Paßwort ändern kann.

Ein weiteres Bit, das im Berechtigungsfeld einer ausführbaren Datei gesetzt werden kann, ist das 'Stickybit'. Dieses Bit hat nicht direkt mit den Zugriffsrechten zu tun. Es veranlaßt den Kernel, das Programm dauernd im Arbeitsspeicher zu halten, auch wenn es nicht mehr ausgeführt wird. Dadurch wird die Zeit zum laden dieses Programms von der Festplatte gespart. Die Verwendung dieses Bits macht aber nur bei sehr häufig benutzten Programmen wie z.B. `ls` Sinn, und setzt natürlich einen entsprechend großen Arbeitsspeicher voraus.

Wie bei den Dateien können auch für die Verzeichnisse Zugriffsrechte gesetzt werden. Die Benutzer werden dabei in die gleichen Kategorien eingeteilt wie bei den Dateien. Die Zugriffsrechte werden der Einfachheit halber genauso bezeichnet wie bei Dateien, bedeuten aber:

- lesbar erlaubt das Lesen des Inhaltsverzeichnisses selbst. Die Rechte zum Lesen von Dateien in diesem Verzeichnis wird dadurch nicht berührt.
- beschreibbar erlaubt das Erstellen und Löschen von Dateien in diesem Verzeichnis. Die Rechte zum Verändern der Dateien in diesem Verzeichnis werden davon nicht berührt.
- ausführbar erlaubt es, dieses Verzeichnis als aktuelles Verzeichnis zu wählen und auf die (lesbaren bzw. ausführbaren) Dateien darin zuzugreifen (auch wenn die Namen der Dateien nicht aufgelistet werden können).
- das Stickybit bedeutet für ein Verzeichnis, daß nur der Eigentümer einer Datei diese auch löschen darf.

Kapitel 5

Recompilieren des Kernels

5.1 Einführung

Da die meisten Routinen der auf Systemerweiterungen vorhandenen BIOSe nicht multitaskingfähig sind, ist der Linux Kernel darauf angewiesen die Steuerung dieser Peripherie selbst zu übernehmen. Um dieser Aufgabe gerechtzuwerden, müssen im Kernel die entsprechenden Treiber vorhanden sein. Da in Linux noch keine Unterstützung für zur Laufzeit ladbare Treiber implementiert ist, muß der Quellcode entsprechend konfiguriert und anschließend übersetzt werden. In dem Quellcode des Kernels sind bereits Treiber für viele verschiedenen Peripheriegeräte enthalten, allerdings ist es nicht sinnvoll alle Treiber immer zu aktivieren. Auf einem Rechner ohne SCSI-Controller ist es beispielsweise unsinnig die Treiber für Streamer, CD-ROM Laufwerke oder SCSI-Festplatten zu integrieren, da unnötigerweise Arbeitsspeicher belegt wird. Ebenso können auf einem Rechner, der als Eingabegerät eine serielle Maus verwendet, alle Busmaustreiber weggelassen werden.

5.2 Entpacken der Quelltexte

Der Quellcode des Kernels sollte sich in dem Verzeichnis `/usr/src/linux` befinden. Sind die Sourcen noch nicht entpackt, so müssen sie in dem Verzeichnis `/usr/src` mit dem Kommando `tar xzf Dateiname.tar.Z` entpackt werden. Damit die zu dem jeweiligen Kernel gehörenden Headerdateien beim Compilieren verwendet werden, müssen sich in dem Verzeichnis `/usr/include` Symlinks auf die Verzeichnisse `/usr/src/linux/include/linux` und `/usr/src/linux/include/asm` befinden. Sie werden mit den Kommandos

```
rm -f /usr/include/linux /usr/include/asm
ln -s /usr/src/linux/include/asm /usr/include/asm
ln -s /usr/src/linux/include/linux /usr/include/linux
```

erzeugt. Da die Headerdateien nicht nur zum Übersetzen des Kernels benötigt werden, sind diese Symlinks für alle Programme, die Datenstrukturen des Kernels verwenden, notwendig. Sind mehrere Kernelquelltexte auf der Platte verfügbar, so ist darauf zu achten, daß die Symlinks auf die richtigen Headerdateien zeigen.

5.3 Konfigurieren des Kernels

5.3.1 Das Konfigurationsscript

Da, wie schon oben gennant, nicht alle Treiber immer im Kernel benötigt werden, sollte der Kernel um Speicher zu sparen auf das jeweilige System konfiguriert werden. Dazu wechselt man in das Verzeichnis `/usr/src/linux` und führt das Kommando `make config` aus. Dadurch wird ein Shellscript gestartet, das die interaktive Konfiguration der meisten Kernelparameter zuläßt. Folgende Fragen werden dem Benutzer von dem Konfigurationsprogramm gestellt:

- Kernel math emulation (y/n, default=n)?
Befindet sich ein mathematischer Koprozessor im System, oder wird der Kernel für einen 80486DX übersetzt, so kann auf diese Frage mit n geantwortet werden, da der Kernel keinen Koprozessor zu emulieren braucht.
 - Normal harddisk support (y/n, default=y)?
Hier wird der Treiber für IDE, RLL, ESDI und MFM Festplatten konfiguriert. Befindet sich eine der Festplattentypen in ihrem System sollten sie auf diese Frage mit y antworten.
 - TCP/IP (y/n, default=y)?
Soll der Rechner im Netz betrieben werden, sollte diese Frage mit y beantwortet werden.
 - Kernel profiling support (y/n, default=n)?
Wenn Programme mit Hilfe eines Profilers optimiert werden sollen, kann mit dieser Option Unterstützung für den Profiler in den Kernel eincompiliert werden.
 - Limit memory to low 16MB (y/n, default=y)?
Da über den ISA-Bus nur 16 MB Speicher adressiert werden können, entsteht das Problem, daß bei DMA-Zugriffen vom Bus aus nur die ersten 16 MB angesprochen werden können. Dadurch können bei Busmaster Kontrollern insbesondere bei SCSI-Kontrollern Fehler beim Ablegen von Datenblöcken im RAM auftreten. Um dies zu verhindern, kann der Kernel gezwungen werden nur die ersten 16 MB Speicher zu verwenden.
 - Use -m486 flag for 486-specific optimizations (y/n, default=y)?
Der 80486 verfügt über eine Reihe Befehle, die gegenüber dem 80386 stark optimiert wurden. Wird diese Option angegeben, so werden verstärkt diese Befehle verwendet. Der erzeugte Maschinencode ist aber auch auf 80386 Prozessoren ausführbar.
 - SCSI support? (y/n, default=n)?
Wird hier n angegeben, so überspringt das Konfigurationsscript alle weiteren SCSI Konfigurationsmöglichkeiten.
 - Scsi disk support (y/n, default=y)?
Bestimmt, ob SCSI-Festplatten unterstützt werden. Die Angaben zu den entsprechenden Kontrollern werden später erfragt.
 - Scsi tape support (y/n, default=y)?
Bislang unterstützt Linux nur SCSI-Bandlaufwerke. An einem QIC-80 Treiber wird gearbeitet.
 - Scsi CDROM support (y/n, default=y)?
Hier kann die Unterstützung für SCSI-CDROM-Laufwerke eincompiliert werden. Diese Unterstützung ist nur für SCSI-CDROMs, die über einen der nachfolgend konfigurierten Hostadapter angesteuert werden. Sie arbeitet nicht mit den 'pseudo' SCSI-Schnittstellen auf Sound- oder Videokarten.
- Die folgenden Fragen beziehen sich auf den Hostadapter, der in ihrem System installiert ist. Die Fragen nach den Adaptern, die in ihrem System installiert sind sollten mit y beantwortet werden, alle anderen mit n.
- Adaptec AHA1542 support (y/n, default=y)?
 - Adaptec AHA1740 support (y/n, default=y)?
 - Future Domain SCSI support (y/n, default=y)?
 - Seagate ST-02 SCSI support (y/n, default=y)?
 - UltraStor SCSI support (y/n, default=y)?
 - 7000FASST SCSI support (y/n, default=y)?

- Standard (minix) fs support (y/n, default=y)?

Da sich noch alle anderen Dateisysteme im Teststadium befinden oder nur eine vorübergehende Form haben, sollten Linuxinstallationen in erster Linie auf diesem Dateisystem basieren.

- Extended fs support (y/n, default=n)?

Wenn Partitionen > 64 MB benötigt werden, bietet sich dieses Dateisystem an. Es ist ziemlich stabil, wenn auch etwas langsam. Allerdings sei daraufhingewiesen, daß dieses Dateisystem wahrscheinlich durch das 'Extended Filesystem 2' ersetzt werden wird. Es ist noch unklar ob es möglich sein wird, das Dateisystem zu konvertieren, d.h. die Partition muß evt. neu formatiert werden um auf das neue Dateisystem umzusteigen.

- msdos fs support (y/n, default=n)?

Wenn MS-Dos Disketten oder Partitionen gemounted werden sollen, kann das Dos-Dateisystem eingekompiliert werden. Wird diese Frage mit n beantwortet ist es trotzdem möglich MS-Dos Disketten mit den `mtools` zu bearbeiten.

- /proc filesystem support (y/n, default=y)?

Das sog. Prozeßdateisystem bildet die interne Prozeßstruktur des Kernels auf das Dateisystem ab. Die neuen ps, free, xsysinfo, u. a. Programme verwenden dieses Dateisystem um die entsprechenden Informationen zu lesen.

- NFS filesystem support (y/n, default=n)?

Wird das NFS-Dateisystem mit eingebunden, so können im Netz Verzeichnisse anderer Rechner gemounted werden.

- ISO9660 cdrom filesystem support (y/n, default=n)?

Das ISO9660 Dateisystem ist das Standarddateisystem für CDROMs. Linux unterstützt sowohl das 'reine' ISO9660 Dateisystem, als auch die Rock Ridge Erweiterungen.

- Keyboard meta-key sends ESC-prefix (y/n, default=y)?

Hier kann konfiguriert werden, ob vor einer Taste, die zusammen mit der ALT-Taste gedrückt wird, ein ESC gesendet wird.

- Autoconfigure serial IRQ lines at bootup (y/n, default=n)?

Hiermit ist es möglich den Kernel die Interrupts, auf denen die seriellen Schnittstellen arbeiten, selbst bestimmen zu lassen. Werden beim Booten falsche Interrupts angezeigt, müssen sie manuell in der Datei `serial.c` im Verzeichnis `kernel/chr_drv` geändert werden.

Die nächsten beiden Treiber stellen die Möglichkeit zur Verfügung unter Linux AST Fourport und Accent Async 4 Karten auf Basis des 16550AFN zu betreiben. Diese Erweiterungskarten bieten die Möglichkeit über die von der PC-Hardware unterstützten vier seriellen Schnittstellen, von denen jede einen eigenen Interrupt benötigt, hinaus, weitere vier serielle Schnittstellen auf einem einzigen Interrupt zu betreiben.

- AST Fourport serial driver support (y/n, default=n)?

- Accent Async 4 serial support (y/n, default=n)?

Linux ist in der Lage mit verschiedenen Busmaustypen zusammenzuarbeiten. Im folgenden ist es möglich Treiber für die vier gebräuchlichsten Busmäuse einzubinden. Für serielle Mäuse ist kein Treiber erforderlich.

- Logitech busmouse support (y/n, default=n)?

- PS/2 mouse (aka 'auxiliary device') support (y/n, default=n)?

- MicroSoft busmouse support (y/n, default=n)?

- ATIXL busmouse support (y/n, default=n)?
- Soundcard support (not really there yet) (y/n, default=n)?

Diese Frage ist eher rhetorisch zu verstehen. Bislang ist im Linux Kernel keine nennenswerte Unterstützung für Soundkarten vorhanden.

Grundsätzlich ist es möglich alle Treiber gleichzeitig einzubinden um einen Kernel auf möglichst vielen Rechnern verwenden zu können. Dies ist bei den Bootdisketten der verschiedenen Distributionen der Fall. Diese Vielfalt führt aber zu einem wesentlich höheren Speicherbedarf des Kernels.

5.3.2 Änderungen im Makefile

Ist die Konfiguration abgeschlossen, muß noch das Makefile geändert werden, da das Shellscript nicht alle Konfigurationsmöglichkeiten berücksichtigt. Zu Konfigurieren bleibt noch das Rootdevice, der Tastaturtreiber, der Videomodus und die Ramdisk. Dazu werden im Makefile die entsprechenden Angaben den Variablen zugewiesen. Das Rootdevice wird über die Variable `ROOT_DEV = /dev/Partition` festgelegt (Zeile 29). Um den Tastaturtreiber zu konfigurieren, muß nur das `#` (Kommentarzeichen) vor der entsprechenden Zeile entfernt werden. Es ist jedoch darauf zu achten, daß nur eine Zeile unkommentiert ist. Der Videomodus wird über die Variable `SVGA_MODE = Modus` eingestellt (Zeile 69). Soll Linux im normalen 80×25 Zeichen Modus arbeiten, so ist hier `NORMAL_VGA` anzugeben, ansonsten wird die Nummer des Videomodus eingetragen, wie er beim Booten angegeben wird. Die Ramdisk wird in erzeugt, indem das Kommentarzeichen vor der Variable `RAMDISK` entfernt wird und in der dahinter stehenden Präprozessordirektive `-DRAMDISK=xxx` die gewünschte Größe angegeben wird.

5.3.3 Konfiguration der Ethernetkarte

Die Standardkonfiguration ist auf eine WD8003 8-Bit Ethernetkarte eingestellt. In dieser Konfiguration arbeitet der Kernel auch mit einer WD8013 kompatiblen Ethernetkarte, verwendet aber nur 8KB des Puffers auf der WD8013. Die Basis I/O Adresse ist 280 und als Interrupt wird IRQ 5 verwendet. Die Speicheradresse ist 0xd0000. Diese Einstellungen werden in der Datei `net/tcp/Space.c` bei der Initialisierung der Struktur "wd8003" festgelegt (ca. Zeile 41). Für eine WD8013 Ethernetkarte ist das erste und dritte Element (jeweils 0xd2000) auf 0xd4000 zu ändern. Wenn gewünscht, kann hier auch der IRQ und die I/O Adresse geändert werden.

5.4 Das Übersetzen der Quellcodes

Sind die Konfigurationen abgeschlossen, so müssen die Abhängigkeiten (Dependencies) der Quelldateien untereinander im Makefile eingetragen werden. Dies geschieht automatisch mit dem Kommando `make dep`. Sind die Dependencies erstellt, kann mit dem Kommando `make image` der Kernel übersetzt werden. Mit `make disk` wird eine neue Bootdiskette erzeugt. Dazu muß eine leere, formatierte Diskette in Laufwerk A eingelegt werden, bevor das Kommando ausgeführt wird. Ist der neue Kernel erstellt, so können mit dem Kommando `make clean` die beim Übersetzen des Kernels erstellten Objektdateien wieder gelöscht werden.

Kapitel 6

Die Installation von Linux

Linux wird auf unterschiedlichen Wegen in verschieden gestalteten Paketen, sogenannten Distributionen angeboten. Wenn eine solche Distribution per Modem auf die Festplatte geladen wurde, muß wenigstens die Bootdiskette mit einem Programm wie `rawrite.exe` (unter MS-DOS) oder `dd` (unter Unix) erstellt werden. Nähere Angaben zur Vorbereitung und Durchführung einer Linuxinstallation finden sich normalerweise in Dateien mit dem Namen README.

Eine Aufgabe, die jeder Linuxinstallation vorangeht, ist die Partitionierung der Festplatte und die Einrichtung eines Linuxdateisystems auf den dafür vorgesehenen Partitionen. Weil dieser Schritt einen tiefen Eingriff in ein bestehendes System bedeutet ist er im folgenden Abschnitt sehr ausführlich beschrieben.

6.1 Die Festplatte partitionieren

GANZ WICHTIG!!

Jede Veränderung an einer Festplatte, die schon Daten enthält (z. B. eine MS-DOS Installation) kann zum Verlust der Daten führen!!

Deshalb muß vor jeder Veränderung an den Partitionstabellen ein Backup aller Daten gemacht werden.

Für Schäden kann keinerlei Haftung übernommen werden, auch wenn sie auf Fehler im Handbuch zurückzuführen sind. Hierzu sei nochmal auf die GNU General Public License verwiesen.

Hintergrundinformation

Eine Festplatte besteht aus mehreren speziell beschichteten Aluminiumscheiben. Die Beschichtung kann magnetisiert werden und dadurch Daten speichern, ganz ähnlich wie ein Tonband Musik speichert. Zum Schreiben und Lesen gibt es die Schreib/Leseköpfe (oder einfach **Köpfe**), für jede beschichtete Scheibenseite einen. Diese Köpfe sind alle zusammen an einem Arm befestigt, der alle gemeinsam (in radialer Richtung) auf den Scheiben bewegen kann. Die Bewegung erfolgt in sehr genau bestimmten Schritten. Bei jedem Schritt erreicht jeder Kopf eine **Spur** der Scheibe. Die Gesamtheit aller Spuren, die von den parallelen Köpfen überstrichen wird, heißt **Zylinder**.

Um einen möglichst schnellen und direkten Zugriff auf die Daten zu ermöglichen, werden die Spuren nicht kontinuierlich beschrieben. Der Festplattencontroller richtet beim Formatieren **Sektoren** ein, in denen jeweils 512 Bytes Platz haben. Diese Sektoren können einzeln "adressiert" werden, das bedeutet, daß jeder Sektor einzeln gelesen und beschrieben werden kann. Die Betriebssysteme verwalten aber in der Regel immer zwei Sektoren gemeinam als einen **Block** von 1024 Bytes.

Die konkrete "Plattengeometrie" ist herstellerabhängig. Eine Platte mit 120Mbyte kann beispielsweise 1024 Zylinder, 14 Köpfe und 17 Sektoren pro Spur haben.

Aus verschiedenen Gründen ist es sinnvoll, die gesamte Kapazität einer Festplatte in verschiedene **Partitionen** zu unterteilen:

1. können manche Betriebssysteme nicht 120MByte auf einmal verwalten
2. können sich verschiedene Betriebssysteme eine Festplatte teilen, indem jedes eine eigene Partition erhält
3. läßt sich der Datenbestand auf der Festplatte besser strukturieren
4. erhöht sich die Datensicherheit, weil von einem Plattenfehler oft nur eine Partition betroffen wird

Die Einteilung der Festplatte in Partitionen ist nicht an irgendwelche physikalischen Gegebenheiten gebunden, sondern wird allein durch die Vereinbarung in der **Partitionstabelle** festgelegt. Diese Partitionstabelle wird von allen Betriebssystemen in gleicher Weise benutzt.

6.1.1 Die Partitionstabelle

Der erste Sektor der Festplatte ist der **primäre Bootsektor**. In diesem Sektor kann eine Partitionstabelle für höchstens vier Partitionen angelegt werden. Diese Partitionen heißen deshalb **primäre Partitionen**. Anstelle einer primären Partition kann in dem primären Bootsektor auch (genau) eine **erweiterte Partition** definiert werden, die den gesamten Plattenplatz enthält, der keiner der primären Partitionen zugeordnet ist. In der erweiterten Partition können weitere **logische Partitionen** eingerichtet werden, die im Prinzip genauso aufgebaut sind wie die primären Partitionen, mit dem Unterschied, daß nur von den primären Partitionen direkt gebootet werden kann. Mit einem Hilfsprogramm wie dem Linux Loader (LILO) kann auch von allen anderen Partitionen gebootet werden. Weitere Hinweise dazu finden sich in der Dokumentation die mit dem Programm verteilt wird.

6.1.2 Planung

Die Partitionen werden von den verschiedenen Betriebssystemen auf die eine (format) oder andere (mkfs) Weise mit spezifischen Strukturen zur Dateiverwaltung belegt. Weil diese Strukturen sich stets auf die ganze Partition beziehen, ist es enorm kompliziert – wenn nicht unmöglich – eine bestehende Partitionstabelle neu einzuteilen, ohne alle Daten auf der Partition oder gar der ganzen Festplatte zu verlieren. Deshalb ist es sinnvoll, die Partitionierung sorgfältig zu planen. Folgende Kriterien sind dabei in Betracht zu ziehen:

- Linux bietet die Möglichkeit, eine Partition als **Swapdevice** zu benutzen. Dieser Speicher wird dann zum kurzfristigen Auslagern schlafender Prozesse aus dem Arbeitsspeicher benutzt. Damit fungiert diese Partition als eine Art Speichererweiterung. Alternativ zu einem eigenen Swapdevice kann Linux auch eine Swapdatei in einer normalen Linuxpartition verwenden. Diese Datei braucht aber den gleichen Speicherplatz und arbeitet weniger effektiv. Die Swappartition sollte etwa so groß sein wie der Arbeitsspeicher, also 5 bis 10 MB. Für Rechner mit weniger als 4 Megabyte Arbeitsspeicher muß unbedingt eine Swappartition eingerichtet werden. Die Swappartition muß mit dem **mkswap** Kommando initialisiert und dann bei jedem Systemstart mit dem Kommando **swapon** aktiviert werden.
- Das Standarddateisystem von Linux kann höchstens 64 MByte Partitionen verwalten. Die Dateinamen können maximal 14 Zeichen lang sein. Dieses Dateisystem basiert auf dem ausgereiften Minix-Dateisystem. Es gibt aber auch ein neues '**extended filesystem**', das mehrere Gigabyte Partitionen verwalten kann, und das Dateinamen bis zu einer Länge von 255 Zeichen zuläßt. Dieses Dateisystem ist aber noch in der Entwicklung. Deshalb ist die Benutzung dieses Dateisystems nur zu Testzwecken vorgesehen.
- Die Minimalkonfiguration von Linux braucht ca. 5 bis 10 Megabyte Festplattenplatz. Ein komplettes Basissystem mit allen Werkzeugen und dem Entwicklungssystem belegt ca 25 Megabyte, für das **T_EX** Satzsystem oder den X11 Server mit Clients kommen jeweils nochmal mindestens 20 Megabyte hinzu. Dazu kommen dann natürlich noch die Daten. C-Quelltexte, Nachrichten aus dem Usenet, Postscript und Druckdateien, alle diese Daten belegen weiteren Platz auf der Festplatte.

- Die Verteilung des Dateisystems auf mehrere Partitionen bringt Vorteile in Hinsicht auf die Betriebssicherheit; bei Beschädigung des Dateisystems ist in der Regel nur eine Partition betroffen. Andererseits wird aber in einem mehrteiligen Dateisystem immer Platz verschenkt. Deshalb ist es wichtig, die Trennlinien zwischen den Teilsystemen vorausschauend zu ziehen.
 - Wenn zum Beispiel Nachrichten aus dem Usenet bezogen werden sollen, ist dafür eine eigene Partition sinnvoll, die auf dem Verzeichnis `/usr/spool` aufgesetzt wird.
 - Wenn das System in größerem Umfang zum Erstellen oder Portieren von C-Programmen benutzt werden soll, kann eine Partition auf das Verzeichnis `/usr/src` aufgesesetzt werden.
 - Wenn das System von vielen verschiedenen Benutzern genutzt wird, erhöht sich die Betriebssicherheit durch eine eigene Partition für die Heimatverzeichnisse auf `/home` oder `/usr/home` beträchtlich.
 - Die Betriebssicherheit des Systems erhöht sich auch, wenn das Rootfilesystem auf einer möglichst kleinen Partition angelegt ist, auf der keine temporären Dateien angelegt werden, und auf die ein Anwender nicht schreibend zugreifen kann.
- Um Linux von der Festplatte booten zu können, muß das Rootfilesystem auf einer primären Partition liegen.

Wenn auf einer Partition der Platz knapp wird, auf einer anderen Partition aber noch reichlich freier Speicher vorhanden ist, kann zur Not auch ein Verzeichnis durch einen symbolischen Link auf die freie Partition verlegt werden.

6.1.3 Benutzung von fdisk

Das fdisk-Programm von Linux dient zur Erstellung von Linux-Partitionen auf IDE Festplatten. Um DOS Partitionen einzurichten, sollte das gleichnamige Programm für DOS verwendet werden.

Um eine SCSI Festplatte für den gemischten Betrieb von Linux und MS-DOS einzurichten, sollte die Festplatte unter MS-DOS partitioniert werden. (Dann können die vom DOS verwendeten Plattenparameter [Köpfe, Zylinder, Sektoren] in das Linux fdisk Programm übertragen werden, und damit die Partitionskennung der Linuxpartitionen geändert werden.)

Wenn fdisk ohne Parameter aufgerufen wird, bearbeitet es die Partitionstabelle der ersten Festplatte, das ist `/dev/hda`. Um die zweite Festplatte, `/dev/hdb`, zu partitionieren muß das Programm als `'fdisk /dev/hdb'` aufgerufen werden. Die erste SCSI Platte wird entsprechend als `'dev/sda'` angesprochen.

Wenn es korrekt aufgerufen wurde meldet sich fdisk mit dem Prompt:

```
Command (m for help): _
```

fdisk erwartet einen einzelnen Buchstaben gefolgt von einem RETURN als Kommando. Das Kommando 'm' zeigt folgendes Menü:

```
Command action
a   toggle a bootable flag
d   delete a partition
l   list known partition types
m   print this menu
n   add a new partition
p   print the partition table
q   quit without saving changes
t   change a partition's system id
v   verify the partition table
w   write table to disk and exit
x   extra functionality
```

Das **print** Kommando zeigt die aktuelle Partitionstabelle an. Eine typische Anzeige sieht beispielsweise so aus:

```
Disk /dev/hda: 5 heads, 17 sectors, 977 cylinders
   Device Boot   Begin    Start      End  Blocks   Id  System
/dev/hda1    *         1        17    20059   10021+    1  DOS 12-bit
/dev/hda2                71060   71060   83044    5992+    5  Extended
/dev/hda3    *    20060   20060   71059   25500    81  Linux/MINIX
/dev/hda5                71061   71061   79559    4249+    82  Linux swap
/dev/hda6                80002   80018   83044    1734     1  DOS 12-bit
```

In diesem Beispiel werden 5 Partitionen angezeigt. Es sind zwei Rootpartitionen auf hda1 und hda3 angelegt, und eine erweiterte Partition auf hda2. In der erweiterten Partition sind zwei logische Partitionen hda5 und hda6 angelegt. Von den Partitionen hda1 und hda3 kann gebootet werden. Der Beginn einer Partition muß nicht mit dem Start des Datenbereichs übereinstimmen, wie das Beispiel der MS-DOS Partitionen zeigt, bei denen immer die ersten Sektoren für Verwaltungsdaten freigehalten werden. Das Linux-fdisk setzt den Start des Datenbereichs normalerweise auf den Partitionsbeginn.

Die auf die Partitionsgröße folgenden '+' Zeichen markieren Partitionen mit ungerader Sektorzahl. Weil Linux immer Blocks zu zwei Sektoren, also immer mindestens 1024 Bytes belegt, bleibt auf diesen Partitionen ein Sektor ungenutzt.

Das **verify** Kommando überprüft die Partitionstabelle auf Fehler und gibt die Anzahl der nichtbelegten Sektoren aus.

Das **quit** Kommando beendet fdisk. Die Veränderungen an der Partitionstabelle werden nicht automatisch geschrieben. Eine Veränderung an der Partitionstabelle wird erst durch ein write Kommando auf die Festplatte gesichert. Es ist möglich, fdisk zu jedem Zeitpunkt mit CTRL-C zu unterbrechen.

Das **delete** Kommando löscht eine Partition aus der Partitionstabelle. Die von dieser Partition belegten Sektoren werden freigegeben. Alle auf diesen Sektoren gespeicherten Daten gehen dabei normalerweise verloren. Das delete Kommando fragt nach der Partitionsnummer, die es löschen soll. Soll doch lieber keine Partition gelöscht werden, kann fdisk mit CONTROL-C beendet werden.

Wenn eine nichtexistierende Partition angegeben wird, erscheint eine Fehlermeldung. Wenn die erweiterte Partition gelöscht wird, verschwinden automatisch alle logischen Partitionen auf der erweiterten Partition mit.

Wenn eine logische Partition gelöscht wird, werden alle darüberliegenden logischen Partitionen sofort neu nummeriert, sodaß alle logischen Partitionen immer fortlaufende Nummern haben.

Das **add** Kommando 'n' erlaubt das anlegen einer neuen Partition. Voraussetzung zum Anlegen einer neuen Partition ist natürlich, daß noch freie Sektoren existieren.

- Wenn ein Platz in der primären Partitionstabelle frei ist, und Sektoren außerhalb einer eventuell existierenden erweiterten Partition frei sind, kann eine Rootpartition angelegt werden
- Wenn ein Platz in der primären Partitionstabelle frei ist, und noch keine erweiterte Partition existiert, kann eine erweiterte Partition angelegt werden
- Wenn innerhalb der erweiterten Partition Sektoren frei sind, kann eine logische Partition angelegt werden

Wenn mehr als eine Möglichkeit zum Anlegen einer neuen Partition existiert, wird gefragt ob eine primäre, erweiterte oder logische Partition angelegt werden soll. Wenn eine primäre oder die erweiterte Partition angelegt werden soll, muß eine freie Partitionsnummer im Bereich 1-4 gewählt werden.

Nun muß der Beginn der neuen Partition angegeben werden. Dazu erscheint Beispielsweise folgende Meldung:

```
First sector (20060-71059): _
```

Die Zahlen bezeichnen den ersten und den letzten freien Sektor. Es müssen nicht alle Sektoren in dem angegebenen Bereich frei sein, weil innerhalb noch eine komplette andere Partition liegt. Die Angabe eines

bereits belegten Sektors wird einfach abgelehnt, und es wird erneut nach dem ersten Sektor der neuen Partition gefragt.

Wenn ein gültiger Sektor für den Beginn der Partition bestimmt ist, wird nach dem letzten Sektor gefragt:

```
Last sector (20060-71059): _
```

Hierbei sind alle Sektoren in dem angegebenen Bereich erlaubt, außer dem ersten, der ja der Beginnsektor ist.

Wenn nicht alle freien Sektoren belegt worden sind, kann das Kommando erneut aufgerufen werden.

Bevor die veränderte Partitionstabelle gesichert wird, sollt auf jeden Fall das `verify` Kommando aufgerufen werden.

Die veränderte Partitionstabelle wird in jedem Fall nur durch ein ausdrückliches **write** Kommando auf die Festplatte geschrieben. In diesem Fall erscheint die Aufforderung, den Rechner neu zu starten:

```
The partition table has been altered.
Please reboot before doing anything else.
```

Es ist ohne Problem möglich, `fdisk` sofort wieder zu starten, um weitere Veränderungen an der Partitionierung vorzunehmen.

ACHTUNG! Es darf auf keinen Fall eines der Programme `mkfs`, `mkswap`, `mount`, oder `swapon` aufgerufen werden, bevor der Rechner neu gestartet wurde!

6.1.4 Die Bedeutung des Boot-Flags

Die meisten modernen Rechner haben ihr Betriebssystem nicht mehr in Permanenten Speicherbausteinen auf der Hauptplatine, sondern laden erst nach dem Einschalten ein Betriebssystem in den Arbeitsspeicher. Und selbst dieser Ladevorgang findet nicht auf eine ganz bestimmte Art und Weise statt, sondern wird von einem sogenannten Bootprogramm ausgeführt. Einzig der Aufruf dieses Bootprogramms ist durch das BIOS festgelegt:

Wenn es nicht ausdrücklich im BIOS Setup anders bestimmt ist, wird im ersten Diskettenlaufwerk nach einer Diskette gesucht. Auf dieser Diskette wird aus dem ersten Sektor (in dem sich auch die Partitionstabelle befindet) das Bootprogramm geladen und ausgeführt.

Wenn keine Diskette im Laufwerk ist, wird das Bootprogramm aus dem Bootsektor der ersten Festplatte ausgeführt. Es gibt verschiedene solcher Bootprogramme. Einige dieser Programme werten die Boot-Flags aus der Partitionstabelle aus, um zu bestimmen von welcher Partition das Betriebssystem geladen werden soll.

Das Boot-Flag ist ein einzelnes Bit für jede Partition. Ist dieses Bit gesetzt, kann von dieser Partition ein Betriebssystem geladen werden.

Das MS-DOS Bootprogramm erwartet genau ein gesetztes Boot-Flag. Das shoelace Bootprogramm ignoriert alle Boot-Flags.

6.1.5 Die Bedeutung des Partitionstyps

Um den verschiedenen Betriebssystemen die Möglichkeit zu geben, Partitionen anderer Formate zu erkennen, werden die Partitionen mit Kennzahlen (ID) versehen. `fdisk` kann z.Zt. 30 Kennzahlen bestimmten Betriebssystemen zuordnen. Es ist aber auch – mit ein paar Ausnahmen – möglich beliebige andere ID's zu setzen.

Es ist nicht möglich eine Partition vom oder nach dem Typ "extended" (5) zu ändern. Es ist nicht möglich einer leeren Partiton (Typ 0) irgendeinen andern Typ zu geben. Einer Partition den Typ 0 zuzuordnen, heißt sie zu löschen.

Die Typ-ID wird als hexadezimale Zahl angegeben. Die folgenden Typen werden erkannt:

0	Empty	a	OPUS	93	Amoeba
1	DOS 12-bit FAT	40	Venix	94	Amoeba BBT
2	XENIX root	51	Novell?	b7	BSDI fs
3	XENIX user	52	Microport	b8	BSDI swap
4	DOS 16-bit <32M	63	GNU HURD	c7	Syrinx
5	Extended	64	Novell	db	CP/M
6	DOS 16-bit >=32	75	PC/IX	e1	DOS access
7	OS/2 HPFS	80	Old MINIX	e3	DOS R/O
8	AIX	81	Linux/MINIX	f2	DOS secondary
9	AIX bootable	82	Linux swap	ff	BBT

Voreinstellung für neue Partitionen ist '81' für das Linux-Dateisystem.

Diese ID führt beim DR-DOS zu Problemen. Wenn eine Festplatte von Linux und DR-DOS gemeinsam genutzt werden soll, ist es sinnvoll anstelle der 81 die 41 als Linux-ID zu verwenden.

6.1.6 SCSI Festplatten einrichten

Um eine SCSI Festplatte für Linux vorzubereiten, muß dem `fdisk` Kommando die Anzahl der Köpfe, Zylinder und Sektoren mitgeteilt werden. Wenn auf der gleichen Festplatte ein MS-DOS betrieben werden soll, müssen hier unbedingt die gleichen Werte eingesetzt werden, die das DOS verwendet.

Um die Werte im Linux `fdisk` einzutragen, stehen im Expertenmenü die Punkte 'h', 'c' und 's' zur Verfügung.

6.2 Eine Distribution einspielen

Die minimale Voraussetzung zur Installation von Linux ist ein Rechner mit 386 Prozessor, mindestens 2 Megabyte RAM und eine freie Festplattenpartition von 10 Megabyte.

Allerdings ist ein sinnvolles Arbeiten mit so einer Konfiguration nicht möglich. Um z. B. das Entwicklungssystem (gcc) oder das \TeX Satzsystem zu benutzen, sind mindestens 4 MB RAM nötig. Das X-Windowssystem braucht sogar 8 Megabyte Arbeitsspeicher. Es ist zwar prinzipiell möglich, fehlenden Arbeitsspeicher durch virtuellen Speicher in einer Swappartition zu ersetzen, der Rechner wird aber mit so einer Konfiguration sehr langsam.

Je nach Umfang der geplanten Installation reichen natürlich 10 Megabyte Festplattenplatz nicht aus. Ein etwas umfangreicheres System (wie die komplette SLS Distribution) kann leicht 80 Megabyte Plattenplatz belegen.

Linux und die dafür verfügbaren Werkzeuge und Programme werden auf unterschiedlichen Wegen angeboten. Eine wichtige Quelle sind die an das Internet angeschlossenen Universitätsrechner und FTP-Server, die Quelltexte für die verschiedensten Programme aber auch komplette Pakete mit ausführbaren Programmen anbieten.

Linux wird aber auch von verschiedenen Mailboxen angeboten und es kann auf Disketten gekauft werden.

Hier soll exemplarisch die Installation für zwei Distributionen beschrieben werden:

Die SLS Distribution von Softlanding System ist wohl das umfangreichste Linuxpaket auf dem Markt. Es besteht zur Zeit aus 30 gepackten HD Disketten und enthält so ziemlich alles, was an freier Software von allgemeinem Interesse sein könnte. Das komplette Pakete belegt die oben genannten 80 Megabyte Festplattenplatz. Darin ist der Platz den verschiedene Programmpakete zusätzlich bei der Benutzung verbrauchen nicht enthalten. Beispielsweise der Compiler braucht zusätzlichen Platz für die Quelltexte, und das Newssystem braucht einige Megabyte Plattenplatz, wenn es an das Usenet angeschlossen wird.

Dank einer weitgehend automatisierten Prozedur kann die Distribution sehr unkompliziert auf der Festplatte installiert werden. Um alle Teile des Systems wirklich in Betrieb zu nehmen bedarf es aber noch einer Vielzahl von Konfigurationen, die der Systemverwalter beziehungsweise der Benutzer selbst vornehmen muß. Die SLS Distribution kommt in der Regel mit amerikanischem Tastaturtreiber, kann aber für die deutsche Tastatur angepaßt werden.

Die LSD Distribution von lunetIX Softfair ist ein wesentlich kleineres Paket, das in mancher Hinsicht weniger "professionell" daherkommt. Für den *NIX Anfänger hat es aber den Vorteil, überschaubar und leichter konfigurierbar zu sein. Diese Distribution kann schrittweise in jeder Richtung ausgebaut werden. Die Erweiterungspakete sind nicht nur bei lunetIX Softfair erhältlich. Sie sind als Quelltexte und Binärdistributionen auf den FTP Servern zu finden, die auch Linux anbieten. Die LSD Distribution arbeitet von vornherein mit der deutschen Tastatur.

6.2.1 Die SLS Distribution

Übersicht

Um das Linux Paket von Softlanding System zu installieren, sind die folgenden Schritte nötig:

- Die Installationsdisketten sollten schreibgeschützt sein.
- Es muß eine leere, formatierte Diskette im Format der Bootdiskette bereitliegen.
- Der Rechner muß mit der Diskette a1 aus der SLS Distribution gebootet werden, die eine Ramdisk als Rootfilesystem anlegt; anschließend wird die Diskette a2 mit weiteren für die Installation wichtigen Programmen auf das Rootfilesystem aufgesetzt.
- Auf einer Festplatte muß mit dem `fdisk` Kommando eine Linuxpartition eingerichtet werden; anschließend muß der Rechner wieder mit der Diskette a1 gebootet werden.
- Auf der Linuxpartition muß mit einem der Kommandos `mkfs` oder `mkefs` ein Dateisystem eingerichtet werden.
- Mit dem Kommando `doinsta|| Rootpartition` beginnt die interaktive Installation der Distributionsdisketten auf der Linux Rootpartition. Wenn die Distribution auf mehrere Partitionen verteilt werden soll (das ist bei Verwendung des Minix Dateisystems und einer Installation von mehr als 64 Megabyte notwendig), können weitere Partitionen gemeinsam mit den Verzeichnissen auf die sie aufgesetzt werden sollen in folgender Form angegeben werden:
`doinsta|| Rootpartition Partition1 Verzeichnis1 Partition2 Verzeichnis2`
 (zum Beispiel "`doinsta|| /dev/hda2 /dev/hdb1 /usr /dev/hdb2 /usr/X386`")
- Zum Schluß wird auf der leeren, formatierten Diskette der neue Kernel installiert, von dem das System danach gebootet werden kann.

Nochmal im Einzelnen

Die SLS Distribution besteht aus 30 HD Disketten mit mindestens 1,2 Megabyte Kapazität. Außer den ersten beiden sind alle Disketten im MS-DOS Format. Die erste Diskette (a1) ist die Bootdiskette, die für die Installation beim Einschalten des Rechners im Startlaufwerk eingelegt sein sollte. Wenn der Kernel geladen ist, muß ein Textmodus ausgewählt werden. Durch Eingabe eines Leerzeichens wird in den Standardmodus mit 80x25 Zeichen geschaltet; RETURN führt zu einem Menü in dem weitere Textmodi angeboten werden, von denen der Kernel meint, daß die Grafikkarte sie unterstütze. Allerdings kann sich der Kernel auch mal irren.

Der Kernel der SLS Bootdiskette ist in der Regel für die amerikanische Tastatur konfiguriert. Der auffälligste Unterschied zwischen dem amerikanischen und dem deutschen Tastaturlayout besteht in der Vertauschung von Z und Y. Aber auch die Sonderzeichen sind unterschiedlich belegt, was vor allem bei den häufig verwendeten Zeichen '/', ' ' und '*' zu Problemen bei der Installation führen kann. Bei den folgenden Paaren ist das erste Zeichen das gewünschte Sonderzeichen, und das zweite Zeichen das Zeichen auf der deutschen Tastatur, das dieses Sonderzeichen mit dem amerikanischen Tastaturtreiber erzeugt.

/, - ,ß *, (, ;, Ö , , ?

Von der a1 Bootdiskette wird eine Ramdisk als Rootfilesystem in den Arbeitsspeicher geladen. Wenn dieser Ladevorgang abgeschlossen ist, muß die Diskette a2 in das Laufwerk gesteckt werden, von dem aus die

weitere Installation vorgenommen werden soll. Dieses Laufwerk muß danach aus einer Liste mit den vier verschiedenen Möglichkeiten durch Eingabe einer Ziffer ausgewählt werden.

Die Diskette a2 enthält ein Minix Dateisystem, das mit dem im Arbeitsspeicher befindlichen Rootfilesystem auf der Ramdisk zusammengebunden wird. Wenn das System soweit erfolgreich gestartet wurde, erscheint ein '#' als Standardprompt der **bash** für den Superuser (root). Bereits dieses Minimalsystem bietet die für die weitere Installation notwendigen Programme.

Für die erste Begegnung mit dem neuen Betriebssystem bietet die SLS eine einfache Menüoberfläche, die durch den Befehl **menu** eingeschaltet wird. Hier können beispielsweise die englischen Hilfstexte angezeigt und ausgedruckt werden. Die Installation selbst ist aber von der Menüoberfläche aus nicht möglich.

Mit dem **fdisk** Programm muß die Festplatte wie auf den Seiten 152 und 191 beschrieben partitioniert und mit entsprechenden PartitionsKennungen für Linux versehen werden. Bei einem so umfangreichen Paket wie der SLS Distribution ist eine sorgfältige Planung der Partitionsaufteilung angesagt. Wenn das Minix Dateisystem benutzt werden soll, dürfen die Partitionen nicht größer als 64 Megabyte gemacht werden. Für Rechner mit weniger als 16 Megabyte Arbeitsspeicher ist eine Swappartition von 5 bis 10 Megabyte einzuplanen, vor allem wenn mit X-Windows gearbeitet werden soll. Weitere Hinweise für die Planung des Dateisystems sind auf den Seiten 192ff. zu finden.

Wenn Festplatte partitioniert ist, muß der Rechner unbedingt neu gebootet werden! Danach muß die Swappartition mit dem **mkswap** Kommando vorbereitet und mit dem **swapon** Kommando in Betrieb genommen werden. Wenn beispielsweise die Partition /dev/hda6 mit 5 Megabyte als Swappartition benutzt werden soll, müssen die Befehle

```
mkswap /dev/hda6 5120; swapon /dev/hda6
```

einggegeben werden.

Die Partitionen für das Linux Dateisystem müssen mit **mkfs** oder **mkefs** (siehe Seite 153) eingerichtet werden. Das erweiterte Dateisystem (**mkefs**) ist noch im alpha Teststadium. Deshalb ist von der Benutzung dieses Dateisystems abzuraten.

Bevor die eigentliche Installation losgehen kann, muß noch eine formatierte Diskette im Format der Bootdiskette bereitliegen. Eine Diskette kann mit dem auf Seite 83 des Handbuchs beschriebenen Kommando **fdformat** formatiert werden.

Mit dem Kommando **doinstall**, hinter dem sich ein Shellsript verbirgt, wird die Installation durchgeführt. Als Parameter muß mindestens die Bezeichnung der Rootpartition angegeben werden, auf der Linux installiert werden soll. Wenn Linux zum Beispiel auf der zweiten Partition der ersten (normalen) Festplatte installiert werden soll, lautet das Kommando:

```
doinstall /dev/hda2
```

Bei einer Installation auf mehreren Partitionen müssen die weiteren Partitionen mit den Verzeichnissen, auf die sie aufgesetzt werden sollen angegeben werden. Wenn beispielsweise zusätzlich zur zweiten Partition noch die erste erweiterte Partition (/dev/hda5) auf das Verzeichnis /usr/spool und die Partition /dev/hda6 auf das Verzeichnis /usr/X386 aufgesetzt werden sollen, lautet das Kommando

```
doinstall /dev/hda2 /dev/hda5 /usr/spool /dev/hda6 /usr/X386
```

Das Installationsprogramm kann jederzeit mit der Tastenkombination **CONTROL-C** abgebrochen werden.

Es werden vom Installationsprogramm nacheinander die verschiedenen Disketten der Serien a, b, c und x aus der Distribution angefordert. Jede Diskette enthält mehrere Pakete mit bestimmten Programmkomplexen. Wenn es zum Beginn des Installationsprogramms so gewählt wurde, kann die Installation jedes dieser Pakete einzeln bestätigt werden.

Nach dem Wechseln der Diskette erscheinen zwei Zeilen mit Daten, in denen der Kerneltreiber für das MS-DOS Dateisystem die Parameter der Diskette anzeigt. Diese Zeilen können ignoriert werden.

Um das gezielte Auswählen einzelner Teile der Gesamtdistribution zu ermöglichen, folgt auf den nächsten Seiten eine Inhaltsangabe für die einzelnen Pakete. (Änderungen im Sinne des technischen Fortschritts vorbehalten)

- Diskette a2 ist beim Start von **doinstall** bereits im Installationslaufwerk. Sie enthält den ersten Teil der Minimalkonfiguration.

image enthält den Kernel von Linux, der im Anschluß an die Installationsprozedur auf die neue Bootdiskette geschrieben wird; dieser Kernel kann später mit dem **fixkbd** Kommando auf das deutsche Tastaturlayout angepaßt und dann mit dem **dd** oder dem **cp** Kommando auf eine leere Diskette geschrieben werden

bin4 enthält die Werkzeuge (tools) für das Verzeichnis /bin (mit **Jumtable.4** gelinkt)

faq enthält eine Textdatei mit häufig gestellten Fragen zu Linux mit ihren Antworten

sc enthält eine einfache Tabellenkalkulation mit **Manualpage** und dem Programm **psc** zum Konvertieren von ASCII Texten in **sc**-Dateien

shlibs enthält die “shared librarys” die von allen Programmen benutzt werden, die nicht statisch gelinkt wurden; die Librarys enthalten Sprungtabellen (jumpables)

zoneinfo enthält verschiedene Dateien für die Zeitzonen unseres Planeten, die von dem **clock** und **date** Kommando ausgewertet werden; die Dateien werden im Verzeichnis /usr/lib/zoneinfo installiert, wo auch der Hilfstext **time.doc** zu finden ist

- Die Diskette a3 enthält weitere Bestandteile der Minimalkonfiguration und Programme zur Datenfernübertragung. Die Fehlermeldung “MINIX-fs magic match failed” beim Wechseln der Diskette kann ignoriert werden.

base enthält weitere Werkzeuge, die in /bin und /usr/bin installiert werden; außerdem werden die zu den Programmen gehörenden Dateien in die Verzeichnisse /usr/lib und /usr/local/lib installiert

comms enthält das **minicom** Terminalprogramm, **kermit** sowie die X- Y- und Z-Modem Datenübertragungsprotokolle (**rx**, **rb**, **rz**, **sx**, **sb**, **sz**)

dosemu enthält den “ausführbaren” “DOS-Emulator”, und die dazugehörenden Quelltexte, Manualpages und Hilfstexte; das zum Starten des “Emulators” notwendige **hdimage** wird im Verzeichnis /usr/src/dosemu installiert

efspgrog enthält die Programme zum Einrichten und Reparieren des erweiterten Dateisystems, die in /usr/bin installiert werden (**mkfs** **efsck**)

etc enthält die Dateien und Programme aus dem Verzeichnis /etc, sowie das Heimatverzeichnis für den Superuser (/root)

getty enthält die Programme **getty** und **ugetty**, die für die logins auf verschiedenen Terminals zuständig sind, und mit je einem Link in den Verzeichnissen /bin und /etc installiert werden; dazu gehören die Konfigurationsdateien /etc/gettydefs und die Dateien im Verzeichnis /etc/default

joe ist ein einfacher bildschirmorientierter Editor; er wird in /usr/bin installiert und die dazugehörige Initialisierungsdatei **joerc** im Verzeichnis /usr/lib

select14 enthält das **selection** Werkzeug zum Ausschneiden und Einfügen von Bildschirminhalten mit der Maus auf virtuellen Terminals, sowie die zugehörigen Quelltexte und Manualpages

term099 enthält einen (Singel-User) Multiplexer für eine serielle Leitung, sowie die dazugehörenden Hilfsprogramme und Manualpages (**term**, **trsh**, **tupload**, **txconn**, **tredir**, **tmon**)

vgalib enthält die Quelltexte zu einem Programm, das die Grafikmodi einer VGA Karte umschaltet

- Diskette a4

bin enthält Werkzeuge und Programme, die in /usr/bin installiert werden, sowie die Programme für das Heimatverzeichnis der anonymen FTP-Benutzer

devs enthält die Gerätedateien (Devices) die im Verzeichnis /dev installiert werden

gzip enthält den für die SLS Distribution verwendeten Packer und die zugehörigen Manualpages (kein zip Archiver)

lilo enthält den “linux loader” zum Laden der Kernels von Festplatte

menus enthält die Dateien für das **menu** Menüsystem

- ps** enthält die Programme zur Anzeige der Prozeßtabelle, der Speichernutzung der Systemauslastung etc. (**ps**, **free**, **top**, **tload**, **fstat**, ...)
- shadow** enthält die Programme, Text- und Konfigurationsdateien für das Shadow Paßwortsystem und zur Benutzerverwaltung; dieses Paßwortsystem ist besonders sicher
- syslogd** enthält einen Dämon, der die Systemmeldungen in Dateien sichert und/oder an Benutzer verschickt (**syslogd**) und die dazugehörigen Quelltexte und Manualpages
- syssetup** enthält zwei Shellscripts zum Setzen der Zugriffsrechte und zum Einrichten des Benutzersystems, die am Ende der Installationsprozedur automatisch ausgeführt werden

Die Disketten der b Serie enthalten die Erweiterungen des Minimalsystems zum Basissystem

- Diskette b1

- man** enthält die fertig formatierten Manualpages für die meisten Werkzeuge
- tcip** enthält die Netzwerkprogramme für das tcp/ip Protokoll; unter anderem sind hier **ftp**, **telnet**, **nfs**, **talk**, **rlogin**, ... inclusive Manualpages und Musterkonfigurationsdateien zu finden; zum Testen der Programme ist keine Netzwerkinstallation notwendig, weil der Kernel ein Loopbackdevice anbietet; die Konfigurationsdateien werden im Verzeichnis `/usr/etc/inet` angelegt, das einen Link auf das Verzeichnis `/etc/inet` hat
- uucp** enthält das Taylor UUCP (**uucp**, **uucico**, **uux**, **uuxqt**, ...) und Manualpages; die Konfigurationsdateien in `/usr/lib/uucp` müssen dem System angepaßt werden

- Diskette b2

- emacsbin** enthält die Programmdatei für den **emacs** Editor
- emacssexp** enthält die Konfigurations- und Hilfsdateien, sowie die elisp Makros für den **emacs** Editor
- usrbin4** enthält die Programme und Manualpages für die englische Rechtschreibkorrektur **ispell**

- Diskette b3

- deliver** enthält ein Programm zur lokalen Zustellung von elektronischer Post
- english** enthält das englische Wörterbuch für die Rechtschreibhilfe
- groff** enthält den **groff** Textformatierer mit allen Hilfsprogrammen, Manualpages Makros und Fonts
- smail** enthält die Programme für die Zustellung elektronischer Post per UUCP; die Konfigurationsdateien in `/usr/local/lib/smail` müssen dem System angepaßt werden

- Diskette b4

- bc** enthält das **bc** Werkzeug zum Berechnen von Zahlenausdrücken und die dazugehörigen Manualpages
- cp22a** enthält das **cpio** Programm zum Kopieren von und nach rohen Datenträgern
- diff20a** enthält das **diff** Programm das die Differenz mehrerer Dateien ermittelt
- elm2417** enthält das **elm** Programm zum Lesen und Schreiben elektronischer Post
- fixkbd** enthält ein Programm zum Verändern des Tastaturtreibers im Kernel (Image) inclusive Source und Manualpage
- ipcbeta** enthält die Quelltexte für die alpha Testversion der System 5 kompatibelen Interprozeßkommunikationsroutinen für den Kernel (Semaphoren, Messages und Shared Memory)
- lpr** enthält den Druckerdämon zum Verwalten mehrerer Drucker und Druckjobs im System; inclusive Manualpages und Konfigurationsdateien
- lx99.4** enthält die Linux Quelltexte; die Sourcen werden im Verzeichnis `/usr/src/linux` installiert

man2 enthält die Manualpages für Sektion 2 (Systemaufrufe)

- Diskette b5

ghostscr enthält die Programme, Makros und Manualpages für den GNU Postscriptinterpreter; mit (dieser Version von) **ghostscript** können Postscriptdateien auf Epson 24Nadeldruckern oder unter X-Windows auf dem Bildschirm ausgegeben werden.

gsfonts1 enthält den ersten Teil der Postscriptzeichensätze

trn22 enthält den "threaded rn" Newsreader zum Lesen und Schreiben von Nachrichten im Usenet

- Diskette b6

gsfonts2 enthält den zweiten Teil der Postscriptzeichensätze

- Diskette b7

cnews enthält die Programme und Konfigurationsdateien für das cnews Paket, das die Nachrichten aus dem Usenet verwaltet; die Dateien werden im Verzeichnis `/usr/lib/news` installiert

nn6418 enthält die aktuelle Version des altgedienten nn Newsreaders, mit dem die Nachrichten aus dem Usenet gelesen, beantwortet und geschrieben werden können; Onlinehilfen, Konfigurationsdateien und der nnmaster Dämon werden im Verzeichnis `/usr/local/lib/nn` installiert

perl ist eine vielseitige Interpretersprache zur Textverarbeitung; einige Perlscripts werden im Verzeichnis `/usr/local/lib/perl` installiert

rscs56a ist ein "revision control system" mit dessen Hilfe verschiedene Versionen einer (Text-)Datei verwaltet werden können

tin11p4 ist ein weiterer Newsrader, der im Unterschied zu nn die Antworten auf einen Artikles in sogenannten threads zusammenfaßt

nn6418m enthält die Manualpages für den nn Newsreader

Die Disketten der c Serie enthalten das Entwicklungssystem.

- Diskette c1

gcc233 enthält den C-Compiler, den Compilertreiber und den Präprozessor

gxx233 enthält den C++ Compiler und die dazugehörenden Headerdateien

- Diskette c2

binutils enthält einige Werkzeuge für den Compiler, wie z. B. **as**, **ld**, **nm** **gprof** und **ar**

f2c enthält das Programm zur Umwandlung von Fortran77 Quelltexten in C Quellcode, sowie die dazu notwendigen Bibliotheken und eine Manualpage

gdb enthält den GNU Debugger

lib42 enthält die C Funktionsbibliotheken

make enthält das **make** Programm mit dem die Übersetzung der größeren Programme verwaltet wird

p2c enthält das Programm zur Übersetzung von Pascal Quelltexten nach C mit den notwendigen Bibliotheken, sowie einen Basic Interpreter; Beispielpprogramme in Pascal werden im Verzeichnis `/usr/src/p2c/examples` installiert

- Diskette c3

clisp enthält den c(ommon)lisp Interpreter/Compiler mit Manualpage; die Lisp Module und die Quelltexte werden im Verzeichnis `/usr/local/lib/lisp` installiert

inc42 enthält die Headerdateien für den C-Compiler

libm enthält die Funktionsbibliothek für die Fließkommaoperationen

smaltalk enthält den **smalltalk** Interpreter; die Klassen werden im Verzeichnis `/usr/local/smalltalk` installiert

Auf den Disketten der x Serie befinden sich das X-Windows System

- Diskette x1

xbina enthält den ersten Teil der X-Clients

xlibs enthält die shared librarys für die X-Programme

- Diskette x2

xbina enthält den zweiten Teil der X-Clients und die Konfigurationsdateien

xmono enthält den Monochromserver

- Diskette x3

bitstrma enthält den ersten Teil der Bitstream Fonts

xmisc enthält verschiedene Fonts

xprog enthält die Funktionsbibliotheken, Headerdateien und Bitmaps zum Compilieren von X-Anwendungen

- Diskette x4

modegen enthält ein Spreadsheet zum Berechnen der Moduseinträge in der Xconfig Datei

type1 enthält die Type1 Fonts

x75dpi enthält die 75dpi Fonts

xman enthält die Manualpages für die X-Clients und den Server

- Diskette x5

afm1 enthält den ersten Teil der Adobe Font Metric

bitstrmb enthält den zweiten Teil der Bitstream Fonts

speedo enthält die Speedo Fonts

xkit enthält Objektdaten aus denen der Server neu zusammengesetzt werden kann

- Diskette x6

afm2 enthält den zweiten Teil der Adobe Font Metric

doc301 enthält den **doc** WYSWIG Editor für \LaTeX (ohne Coprozessor nicht empfehlenswert)

idraw enthält das **idraw** Zeichenprogramm (ohne Coprozessor nicht empfehlenswert)

Nach der Installation der X-Serie wird die neue Bootdiskette erzeugt, und ein paar Konfigurationen vorgenommen. Die weiteren Serien müssen nach dem ersten Booten des neuen Systems mit dem **sysinstall** Kommando vorgenommen werden.

Das **sysinstall** Kommando

Syntax:

```

sysinstall [-view] [-all] [-base] [-mini] [-rest] [-X11] [-serie Serie] [-install Paket] [-remove Paket]
[-extract Paket] [-extracttar Paket] [-disk Diskette] [-mount] [-umount] [-instdev Gerätedatei] [-instroot
Verzeichnis]
[-instsrc Verzeichnis]
[-doprompt] [-compress]

```

Beschreibung:

Das **sysinstall** Kommando ist ein Shellsript, das die Kommandos **tar**, **sed**, **basename**, **compress/zcat/gzip**, **mv**, **mount** und **umount** aufruft um SLS Distribution auf der Festplatte zu installieren. Es können einzelne Serien, Disketten oder Pakete von Verschiedenen Geräten aus installiert werden.

Optionen:

- view** zeigt die bisher installierten Pakete
- all** installiert die Serien a, b, c und x
- base** installiert die Serien a, b und c
- mini** installiert die Serie a
- rest** installiert die Serien a und b
- X11** installiert die Serie x
- series *Serie*** installiert die angegebene *Serie*
- install *Paket*** installiert das angegebene *Paket*
- remove *Paket*** entfernt das angegebene *Paket*
- extract *Paket*** packt das *Paket* wieder zu einem .tpz zusammen
- extracttar *Paket*** wie extract, nur mit .tar wenn das kleiner ist
- disk [*Diskette*]** installiert die Angegebene Diskette (z. B. diskd1 oder diskd2); wenn keine Diskette angegeben ist, wird die im Installationslaufwerk befindliche Diskette installiert
- umount** setzt das "aktuelle Installationsgerät" ab
- mount** fügt das Installationsgerät (Festplattenpartition) in das Dateisystem ein
- instdev *Gerätedatei*** veranlaßt sysinstall die Installationsdateien von dem angegebenen Diskettenlaufwerk zu lesen
- instroot *Verzeichnis*** setzt das angegebene *Verzeichnis* als relatives Wurzelverzeichnis für die Installation
- instsrc *Verzeichnis*** benutzt das *Verzeichnis* (auf der Festplatte) anstelle von Disketten als Quelle der Installationsdateien
- doprompt** fragt vor der Installation jedes Paketes nach
- compress** benutzt das **compress** Programm anstelle von **gzip**

Die Erweiterungsserien

Die Disketten der d Serie enthalten Dokumentationstexte zu einigen Programmen.

- Diskette d1

dnewspak enthält die Hilfstexte für die Programme **cnews**, **elm**, **nn**, **smail**, **tin** und **trn**, die im Verzeichnis **/usr/doc** installiert werden

info1 enthält die texinfo Dateien für die Programme **bison**, **gawk**, **termcap**, **tar**, **make**, **gdb**, **cpp** und **gcc**

- Diskette d2

info2 enthält die texinfo Dateien für die Programme **emacs**, **elisp**, **vip** und **texinfo**

Die s Serie ist für die Quelltexte zu den Programmen vorgesehen.

- Diskette s1

sefsprog enthält die Quelltexte für die Programme zum Erstellen und Reparieren des erweiterten Dateisystems

sgettyps enthält die Sourcen für **getty** und **ugetty**

smount enthält die Quelltexte für **mount** und **umount**

snetwork enthält die Quelltexte für die Netzwerkprogramme (ohne **nfs**)

snfsserv enthält die Quelltexte für den **nfs** Server

sprocps enthält die Quelltexte für **ps**, **free**, **tload** und **uptime** mit Unterstützung des Prozeßdateisystems

sselecti enthält die Quelltexte für das **selection** Programm für die Mausunterstützung auf virtuellen Terminals (mit zusätzlichem Kernelpatch)

sshadow enthält die Quelltexte für die Programme des shadow Paßwortsystems

ssource enthält eine kleine Sammlung weiterer Quelltexte, die im Verzeichnis **/usr/src/src** installiert werden

Auf den Disketten der t Serie sind der **T_EX** und **L^AT_EX** Textformatierer untergebracht

- Diskette t1

texbin enthält **T_EX**, Metafont und verschiedene dazugehörige Programme, wie zum Beispiel **xdvi**

texdoc enthält das Gentle **T_EX**dokument (US Version)

texlib enthält einen Teil der Metafont Programme für die **T_EX** Zeichensätze, einige der **T_EX** Fontmetriken und die Makros

textfm enthält die **T_EX** Fontmetriken

- Diskette t2

cprint enthält die Quelltexte und **T_EX** Makros für das **cprint** Programm, das C-Quelltexte für **L^AT_EX** formatiert

dekmf enthält Metafont Programme für die meisten **T_EX** Fonts

dviljsrc enthält die Quelltexte für die HP Laserdruckertreiber

dvips enthält die Quelltexte für den **dvi** nach Postscript Konvertierer und die dazugehörigen "virtuellen Fonts"

tex-manp enthält die Manualpages für das **T_EX** Programmpaket

texmisc enthält noch weitere Programmdateien im Umfeld des **T_EX** Paketes

xlib21 enthält die neuere Version 2.1 der shared X Librarys

- Diskette t3

infosrc enthält die Quelltexte zum \TeX infoprogramm **info** mit dem die \TeX info-dateien gelesen werden können

latex enthält die \LaTeX Makros und Style Dateien

texpk enthält die gepackten \TeX Fonts

Tastaturtreiber ändern

Mit Hilfe des **fixkbd** Kommandos läßt sich der Tastaturtreiber des Kernels ändern, ohne ihn neu zu übersetzen. Zu dem Kommando existiert eine englische Manualpage. Das Kommando arbeitet nicht mit der Bootdiskette, sondern nur mit der Kerneldatei, die bei der SLS Distribution im Wurzelverzeichnis abgelegt ist und den Namen **image** trägt. Um den deutschen Tastaturtreiber in den Kernel einzufügen muß im Wurzelverzeichnis das Kommando

```
fixkbd -pKBD_GE_LATIN1 image
```

eingegeben werden. Mit dem **rdev** Kommando (siehe Seite 158) können noch weitere Veränderungen an der Kerneldatei vorgenommen werden, bevor sie mit dem **cp** oder dem **dd** Kommando auf Diskette geschrieben wird.

6.2.2 Die LSD Distribution

Die **lunetIX** Softfair Distribution besteht aus 8 Disketten für 3.5 Zoll Laufwerke und einer 5 1/4 Zoll Diskette. Die 5 1/4 Zoll Diskette und eine 3.5 Zolldiskette sind identisch, und dienen zum ersten Booten des Systems. Wenn der Rechner mit einer dieser Disketten im Bootlaufwerk gestartet wird, erscheint nach den normalen BIOS Meldungen eine Zeile mit der Meldung “**Loading**” und einer Reihe von Punkten, die den Fortschritt des Ladevorgangs anzeigt.

Wenn der Kernel geladen ist, Erscheinen verschiedene Meldungen auf dem Bildschirm, deren Bedeutung auf Seite 178 des Handbuches beschrieben ist.

Unter anderem wird die ebenfalls auf der Bootdiskette befindliche Ramdisk in den Arbeitsspeicher geladen, die für die folgende Installation als Rootfilesystem dient. Die Ramdisk belegt 640 Kilobytes. Diese Ramdisk enthält bereits alle für die Installation notwendigen Programme. Diese Diskette kann auch später noch benutzt werden, um beispielsweise das Rootfilesystem auf der Festplatte zu reparieren.

Wenn die Kernelinitialisierung abgeschlossen ist, wird sofort ein Hilfstext für die weitere Installation angezeigt. Diese Anzeige findet mit dem Kommando **more** statt, das im Handbuch auf Seite 115 beschrieben ist.

Anschließend erscheint ein einzelnes Semikolon ‘;’ als Eingabeaufforderung der **rc-Shell**. Diese Shell wird nur wegen ihres geringen Platzbedarfs für die Installation verwendet. In dieser Shell steht weder ein Kommandozeileneditor, noch ein Kommandozeilenspeicher wie in der **Bash** zur Verfügung. Für die wenigen Eingaben, die für die Installation notwendig sind, sollte das aber kein Problem darstellen. Eine Beschreibung der **rc-Shell** bieten wir nicht an.

Wenn nicht schon unter **MS-DOS** die Partition(en) für **Linux** eingerichtet worden sind, muß an dieser Stelle mit dem **fdisk** Kommando eine entsprechende Partitionierung durchgeführt werden. Auch wenn schon unter **MS-DOS** Partitionen erstellt worden sind, die mit **Linux** belegt werden sollen, müssen die Partitionskennungen der **Linuxpartition(en)** noch mit dem **Linux fdisk** Kommando auf 81 gesetzt werden. Das **fdisk** Kommando und die Partitionierung der Festplatte werden im Handbuch auf den Seiten 152 und 191ff. beschrieben.

Nach jeder Änderung an den Partitionstabellen muß der Rechner neu gebootet werden, damit die Änderungen wirksam werden.

Wie im Abschnitt über die Planung der Partitionen (Seite 192) beschrieben, ist es unter bestimmten Umständen sinnvoll mehrere Partitionen für **Linux** einzurichten.

Auf allen Partitionen, die ein **Linux** Dateisystem enthalten sollen, muß das Kommando **mkfs** oder **mkefs** angewandt werden, deren Benutzung auf Seite 153 des Handbuches beschrieben ist.

Wenn eine Partition als Swappartition zum Vergrößern des virtuellen Arbeitsspeichers benutzt werden soll, muß diese Partition mit dem Kommando **mkswap** (siehe Seite 156) ein einziges mal vorbereitet werden,

und dann mit dem Befehl `'swapon Gerätedatei'` in Betrieb genommen werden. Die Gerätedatei muß der bei `mkswap` benutzten Swappartition entsprechen.

Wenigstens bei Rechnern mit weniger als 4 Megabyte Arbeitsspeicher sollte das vor der weiteren Installation geschehen.

Die eigentliche Installation wird mit dem Kommando

install *Rootpartition*

gestartet. *Rootpartition* ist dabei der einfache Name der Gerätedatei für die Partition, auf der Linux eingerichtet werden soll (ohne Pfad). Das ist also beispielsweise `'hda2'` für die zweite Partition der ersten AT-Bus Platte, oder `'sdb1'` für die erste Partition der zweiten SCSI Festplatte.

Wenn die Partition richtig vorbereitet worden ist, und das Kommando korrekt eingegeben wurde, erscheint die Frage, ob die Installation vom Laufwerk A durchgeführt werden soll. Wenn diese Frage verneint wird, findet die Installation von Laufwerk B statt.

Die weitere Installation des Basissystems erfolgt automatisch wenn die nacheinander angeforderten Disketten eingelegt werden. Nachdem die Basisdistribution ohne den X11-Server installiert ist, erscheint die Frage, ob auch das X11 System installiert werden soll.

Zum Abschluß der Installation muß noch angegeben werden, ob ein mathematischer Koprozessor im Rechner vorhanden ist, und es muß eine (unformatierte) Diskette in das Bootlaufwerk eingelegt werden, auf die der Kernel geschrieben wird, mit der das neu eingerichtete Betriebssystem zukünftig gestartet werden kann. Die Diskette wird automatisch und ohne weitere Nachfrage neu formatiert, und die unter dem Namen `'Image'` im Wurzelverzeichnis des neuen Systems abgelegte Kerneldatei mit dem `dd` Kommando darauf geschrieben.

Wenn, abweichend von der hier beschriebenen Installationsprozedur, Linux auf mehreren Partitionen verteilt installiert wurde, müssen die Partitionen des Linux Dateisystems in der Datei `/root/etc/fstab` eingetragen werden, bevor der Rechner mit der neuen Bootdiskette gestartet werden kann. Diese Datei enthält schon einige Beispiele, und die Bedeutung der einzelnen Einträge ist auf Seite 163 des Handbuches beschrieben.

Das erste Login auf dem neuen System findet als Superuser unter dem Benutzernamen **root** statt. Dieser Account sollte durch ein Passwort gesichert werden. Außerdem sollte noch mindestens ein Benutzer ohne Privilegien eingetragen werden, damit die alltägliche Arbeit mit Linux nicht unter Rootrechten durchgeführt werden muß.

Von den Dateien im Verzeichnis `/etc` sollten vor allem `'rc'`, `'rc.local'`, `'fstab'`, `'profile'` und `'motd'` an das neue System angepasst werden. Die Bedeutung dieser Dateien ist im Handbuch beschrieben.

6.3 X-Windows konfigurieren

Wenn das X-Windows System auf der Festplatte installiert ist, müssen vom Superuser eine Reihe von Konfigurationen vorgenommen werden, ohne die der X-Server nicht funktionieren wird.

Alle Dateien, die irgendwie zum X-Windows System gehören, sind im Ast `/usr/X386` des Verzeichnisbaums untergebracht. Die Konfigurationsdateien sind alle im Verzeichnis `/usr/X386/lib/X11` (und den Unterverzeichnissen) zu finden.

Im Unterverzeichnis `'etc'` befinden sich die englischen Hilfstexte zur Installation des Servers, deren Lektüre dem englischkundigen Leser unbedingt angeraten wird.

Um den Server in das System einzufügen sollte natürlich das Verzeichnis `'/usr/X386/bin'` mit den Binärdateien in die `PATH` Umgebungsvariable eingefügt werden. Wenn diese Veränderung für alle Anwender gelten soll, muß sie in der Datei `/etc/profile` (siehe Seite 168) vorgenommen werden. (Das Verzeichnis `/etc` ist bitte nicht mit dem Unterverzeichnis `/usr/X386/lib/X11/etc` zu verwechseln.)

Außerdem erwartet der X-Server in der Umgebungsvariablen `'DISPLAY'` den Wert `:0` (Sie bezeichnet den Socket, auf dem die Clients den Server kontaktieren; normalerweise `:0.`), sodaß die Zeile `'export DISPLAY=:0'` in der passenden Shellinitialisierungsdatei eingetragen werden muß.

Um das Kommando `man` für die X-Manualpages benutzen zu können, muß das Verzeichnis `/usr/X386/man` in der Datei `/usr/lib/manpath.config` eingetragen werden. Dazu muß nur das Kommentarzeichen vor den entsprechenden Zeilen entfernt werden.

Schließlich sollte in der Datei `/etc/termcap` ein Eintrag für `xterm` zu finden sein. Eventuell muß dazu eine entsprechende Datei aus `/usr/X386/lib/X11` an die `termcap` Datei angehängt werden.

Die eigentliche Konfigurationsdatei für den X-Server ist die Datei `'Xconfig'`.

Ähnlich wie bei der Konfiguration einer Programmumgebung mit den Shellvariablen werden hier bestimmten Variablen oder Schlüsselworten Werte oder Zeichenketten zugewiesen, die vom X11-Server ausgewertet werden. Zeichenketten müssen in Anführungszeichen eingeschlossen werden.

Einige dieser Schlüsselworte werden nicht belegt. Sie schalten allein durch ihr Erscheinen in der Datei die zugeordnete Funktion ein.

Zeilen, die ein `'#'` Zeichen enthalten, werden ab diesem Zeichen bis zum Zeilenende als Kommentar bewertet. Auf diese Weise können Einträge in der Musterkonfigurationsdatei einfach "auskommentiert" werden.

In den ersten Zeilen der Konfigurationsdatei werden einige Pfade gesetzt, die der Installation angepaßt werden müssen.

Der `RGBPath` zeigt auf die 'Familie' der Farbdefinitionsdateien. In dieser Datenbank werden die Bildschirmfarben für den Server definiert. Dieser Pfad sollte mit `'/usr/X386/lib/X11/rgb'` richtig vorbelegt sein.

Der `FontPath` enthält alle Verzeichnisse, in denen die Zeichensätze für die Anwendungen (Clients) gespeichert sind. Dieser Pfad kann auch in der Umgebungsvariablen `'FONT_PATH'` in der Shell definiert werden. Hier sollten alle Unterverzeichnisse von `/usr/X386/lib/X11/fonts` angegeben werden. Von besonderer Bedeutung sind in den jeweiligen Verzeichnissen die Dateien `fonts.dir` und `fonts.alias`. Sollte sich der Server beim Starten über das Fehlen bestimmter Zeichensätze beklagen, so hilft es meistens in jedem der Verzeichnisse das Kommando `mkfontdir` auszuführen. Dadurch werden die Zeichensatzverzeichnisse neu erstellt.

Nach den Pfaden wird mit verschiedenen Variablen das Verhalten des Servers eingestellt.

`NoTrapSignals` verändert das Verhalten des Servers bei schweren Fehlern und sollte auskommentiert bleiben. `Xqueue` sollte ebenfalls auskommentiert bleiben; stattdessen muß das Schlüsselwort `'Keyboard'` ohne Kommentarzeichen stehen.

Mit dem `AutoRepeat` Eintrag läßt sich die Verzögerung und die Frequenz der automatischen Tastaturwiederholung einstellen.

`XLeds` macht die entsprechenden LED's auf der Tastatur für X-Clients ansprechbar.

`ServerNumLock` erlaubt den Anwendungen über den Server auf den Nummernblock der Tastatur zuzugreifen.

`DontZap` verbietet die Unterbrechung des Servers mit der `'ALT-control-backspace'` Kombination. Wenigstens bis zur vollständigen Konfigurierung des Servers sollte diese 'Notbremse' erhalten bleiben.

Mit den Einträgen `'busmouse'`, `'logitech'`, `'microsoft'`, `'mousesystems'`, `'mouseman'`, `'mmseries'` oder `'ps/2'` werden die entsprechenden seriellen Mäuse oder Busmäuse ausgewählt, und mit der entsprechenden Gerätedatei verbunden. Die erste serielle Schnittstelle (COM1) wird als `'/dev/ttyS0'` angesprochen. Die Gerätedateien für die Busmäuse heißen `/dev/bm`, `/dev/psaux` oder so ähnlich haben alle die Major Devicenummer 10.

Für eine Maus mit zwei Tasten gibt es die Möglichkeit die dritte Taste durch gemeinsames Drücken der zwei vorhandenen Tasten zu emulieren, wenn `Emulate3Buttons` unkommentiert ist.

Die Distribution enthält zwei verschiedene X-Server: Einen Farbserver, der nur mit bestimmten Super VGA Karten zusammenarbeitet und 256 Farben in einer Auflösung bis zu 1152x910 (oder 1184x885) Bildpunkten bietet, und einen Monochromserver, der mit **allen** VGA Karten zusammenarbeitet, aber nur in der kleinsten aller möglichen Auflösungen mit 640x480 Bildpunkten zwei Farben darstellen kann.

Der Farbserver trägt den Namen `X386` und der Monochromserver heißt `X386mono`. Beide Programme befinden sich im Verzeichnis `/usr/X386/bin`. Der aktive Server muß mit einem (symbolischen) Link auf den Namen **X** versehen werden. Damit wird der eine oder andere Server vom `startx` Programm aufgerufen.

Der Monochromserver

Der Monochromserver kann mit jeder Hercules, EGA, VGA oder SVGA Grafikkarte und mit jedem Bildschirm zusammenarbeiten. Außer bei SVGA Karten kann immer nur der 640x480 Modus dargestellt werden. Bei SVGA Karten treffen für die Modusauswahl die gleichen Kriterien wie beim Farbserver zu (nur mit ET4000 getestet).

Um den Monochromserver zu aktivieren, müssen die Zeile mit dem Schlüsselwort `vga256` und die darauf folgenden Einstellungen mit einem Kommentarzeichen unwirksam gemacht werden. Stattdessen müssen die Zeilen mit dem Schlüsselwort `vga2` und in der `Modes` Zeile nur der `640x480`er Modus eingetragen sein (nur bei Hercules, EGA und VGA Karten).

Außerdem muß im Verzeichnis `/usr/X386/bin` der Link von `X386` auf `X` gelöscht und durch einen Link von `X836mono` auf `X` ersetzt werden.

Der Monochromserver der XFree86 Release 1.2 unterstützt Speicherbankumschaltung auf SVGA Karten. Dadurch können bis zu 256 Kilobyte des Videoram benutzt werden, was eine virtuelle Bildschirmgröße bis zu 1600x1200 Bildpunkten ermöglicht. Ältere Server und solche ohne SVGA benutzen nur 64 Kilobyte und können deshalb höchstens 800x600 virtuelle Bildpunkte bearbeiten.

Der Farbserver

Mit dem Schlüsselwort `vga256` beginnt in der Konfigurationsdatei die Sektion für den Farbserver. Die Zeilen unmittelbar nach dem Schlüsselwort `vga2` sind entsprechend für den Monochromserver vorgesehen, und müssen mit einem Kommentarzeichen versehen sein.

Wie schon oben gesagt, kann der Farbserver nur mit ganz bestimmten Grafikkarten zusammenarbeiten. Es sind dies alle Karten mit den ET3000/ET4000 Chips von Tseng (mit Ausnahme der Karten vom Hersteller Diamond, die nicht unterstützt werden), die Karten mit dem `gvg`a Chip von Genoa und die Karten mit dem `pvga` Chip von Paradise. Die maximale Auflösung kann nur mit Karten erreicht werden, die 1 Megabyte RAM installiert haben. Mit 512 Kilobyte RAM läßt sich eine maximale Auflösung von 832x624 in 256 Farben darstellen.

Mit der Variablen `Virtual` kann ein virtueller Bildschirm bestimmt werden. Unabhängig von den tatsächlich auf dem Bildschirm dargestellten Bildpunkten werden dann alle Grafikausgaben für diese Bildschirmgröße berechnet, und auch in den Speicher der Grafikkarte geschrieben. Wenn der Mauszeiger den Rand des real dargestellten Bildes berührt, wird automatisch der reale Bildschirm im virtuellen Bild in der entsprechenden Richtung verschoben, sofern das virtuelle Bild in dieser Richtung größer ist als das real darstellbare. Das reale Bildfenster, muß allerdings komplett in das virtuelle Fenster passen. Ebenso müssen alle in der `Modes`-Zeile angegebenen Modi in das gleiche virtuelle passen (ein Modus "1152x910" und "1184x885" können nicht gleichzeitig verwendet werden).

Im Gegensatz zu den von verschiedenen Windowmanagern angebotenen virtuellen Bildschirmvergrößerungen müssen zum Verschieben des realen Bildes im virtuellen keine Bilddaten zur Grafikkarte kopiert werden, was deutliche Geschwindigkeitsvorteile bringt.

Der Farbserver enthält sogenannten SpeedUp Code zur Beschleunigung des Servers in bestimmten Situationen. Einige dieser Routinen funktionieren nur bei ET4000 Karten, andere können nur bei einer virtuellen Bildschirmbreite von 1024 Punkten genutzt werden. Wenn diese Voraussetzungen nicht gegeben sind, werden die entsprechenden Routinen automatisch abgeschaltet.

Der virtuelle Bildschirm darf höchstens so viele Bildpunkte belegen, wie die Grafikkarte Speicherplatz bereitstellt.

Der Server erlaubt das Umschalten zwischen mehreren Auflösungen mit der Tastenkombination `ALT+CTRL+NUMPLUS` (`NUMPLUS` ist die Plustaste im Nummernblock) und `ALT+CTRL+NUMMINUS`. Die verschiedenen Modi und ihre Reihenfolge werden in der 'Modes' Zeile festgelegt. Der erste Modus ist nach dem Starten des Servers aktiv, die weiteren werden der Reihe nach vorwärts bzw. rückwärts mit den entsprechenden Tastenkombinationen umgeschaltet.

Außerdem erlaubt der XFree86 Server ab Version 1.2 das Umschalten zwischen den virtuellen Terminals mit der Tastenkombination `ALT CONTROL F1` bis `F8`. Der X-Server benutzt das erste freie virtuelle Terminal; wenn mit der entsprechenden Tastenkombination auf dieses Terminal geschaltet wird, erscheint wieder das X11 Display.

Die nach 'Modes' eingetragenen Zeichenketten müssen mit Einträgen in dem durch das Schlüsselwort `ModeDB` eingeleiteten Teil der Konfigurationsdatei übereinstimmen. Mit diesen Modes wird die Grafikkarte für verschiedene Bildschirmauflösungen konfiguriert.

Der Viewport sollte die Werte '0 0' enthalten, damit der reale Anfangsbildschirm in der linken oberen Ecke des virtuellen Bildschirms liegt.

In einer Zeile, die mit dem Wort ‘Clocks’ beginnt können die verschiedenen Punkttaktfrequenzen (Clocks) der SVGA Karte eingetragen werden, wenn sie bekannt sind. Für eine Reihe verbreiteter Grafikkarten sind diese Werte in der ‘ModeDB.txt’ Datei bei den englischen Hilfstexten zu finden. Diese Zeile wird nur bei SVGA Karten zum Programmieren der Grafikkarte verwendet.

Wenn keine Clocks Zeile eingetragen wird, ermittelt der Server selbst die Punkttaktfrequenzen der Karte.

In dem durch das Schlüsselwort ‘ModesDB’ eingeleiteten Teil der Konfigurationsdatei müssen durch die bei ‘Modes’ bestimmten Namen für jede gewünschte Bildschirmauflösung eine von der Karte unterstützte Punkttaktfrequenz mit einem vom Monitor unterstützten Timing verbunden werden.

Eine Zeile besteht aus sieben Teilen:

Modusname ist eine beliebige Zeichenkette. Wenn dieser Modus aktiv werden soll, muß der Name mit einem entsprechenden Eintrag in der ‘Modes’ Zeile übereinstimmen. Es wird traditionell die Bildschirmauflösung als Name verwendet. Es gibt aber keinen Zusammenhang zwischen dem Namen und der tatsächlich dargestellten Auflösung.

Punktfrequenz muß für alle eingetragenen Modi mit einer von der VGA Karte unterstützten Punkttaktfrequenz (Clock) übereinstimmen. Es können für einen Modus auch mehrere Zeilen mit unterschiedlichen Clocks eingetragen werden. Der Server benutzt dann automatisch die erste Zeile mit einer passenden Frequenz.

horizontales Timing besteht aus vier Zahlen, die den Aufbau einer Bildzeile bestimmen. Jede der Zahlen muß durch 8 teilbar sein.

hdisp ist die Anzahl der Punkte, für die tatsächlich Bilddaten übertragen werden. Für eine Bildschirmauflösung von 1024x768 wären das genau die 1024 Bildpunkte für jede Zeile, im 640x480 Modus 640 usw.

hsyncstart ist die Anzahl der Punkte, nach denen der horizontale Synchronimpuls beginnt. Es wird vom Beginn der Übertragung von Bilddaten an gezählt. Die Differenz von hsyncstart und hdisp ist die Länge der vorderen Schwarzscher.

hsyncend ist die Anzahl der Punkte, nach denen der horizontale Synchronimpuls endet. Es wird wieder vom Beginn der Übertragung von Bilddaten an gezählt. Die Differenz von hsyncstart und hsyncend ist die Länge des Synchronimpulses.

htotal ist die Anzahl der Punkte vom Beginn der Übertragung eines Bildes bis zum Beginn der Übertragung des nächsten Bildes, also die gesamte Anzahl der zur Übertragung einer Zeile Bilddaten Übertragenen Punkte. Die Differenz von htotal und hsyncend ist die Länge der hinteren Schwarzscher.

vertikales Timing für das vertikale Timing gibt es wieder vier Zahlen, die entsprechend dem horizontalen Timing aus den Werten für vdisp, vsyncstart, vsyncend und vtotal bestehen. Die Zahlen für das vertikale Timing müssen nicht durch 8 teilbar sein.

interlaced ist ein Schlüsselwort, das die Darstellung mit Zeilensprung einschaltet.

{+/-}**hsync** setzt die entsprechende Polarisierung des horizontalen Synchronisationsimpulses.

{+/-}**vsync** setzt die entsprechende Polarisierung des vertikalen Synchronisationsimpulses.

Für die optimale Anpassung des Servers an die Grafikkhardware sind noch ein paar Zahlen wichtig, die sich aus den Zeilen der ModeDB errechnen lassen:

Zeilenfrequenz ist der Quotient aus Punktfrequenz und der gesamten Zeilenlänge ‘htotal’.

Bildfrequenz ist der Quotient aus Zeilenfrequenz und der gesamten Zeilenanzahl ‘vtotal’.

Wenn in der ModeDB.txt Datei kein passender Eintrag für den Bildschirm gefunden wird, muß die Zeile für den Monitor neu zusammengesetzt werden. Dabei ist vor allem bei Festfrequenzmonitoren größte Sorgfalt nötig. Eine falsche Zeilenfrequenz kann diese Monitore zerstören.

In den Texten CONFIG und XConfig.notes im etc Unterverzeichnis ist die Anpassung ausführlich beschrieben.

Für einen ersten Test des X-Servers kann aber in jedem Fall die Standard VGA Auflösung '640x480' mit der 25 MHz Clock mit den Zeilen:

```
Mode "640x480"
```

```
ModeDB
```

#	name	clock	horizontal timing				vertical timing				flags
	"640x480"	25	640	664	760	800	480	491	493	525	

aktiviert werden. Diese Einstellung entspricht dem VESA- und dem IBM Standard für VGA, und wird von allen VGA Monitoren unterstützt.

Ohne die Berechnung der Zahlen für den Bildaufbau näher zu beschreiben sollen hier doch noch ein paar Hinweise und Tips gegeben werden.

Die Benutzung dieser Information erfolgt wie beim gesamten Handbuch auf eigenes Risiko. Eine Haftung schließen wir und die GNU General Public License aus.

Zuallererst soll auch hier nochmal die dringende Warnung wiederholt werden: Eine falsche Monitoransteuerung kann die Lebensdauer des Bildschirms auf wenige Minuten verkürzen. Im Zweifelsfall ist professionelle Hilfe sicher die billigere Lösung.

—

Bei der selbstständigen Berechnung ist vor allem auf den Wert für die Zeilenfrequenz zu achten. Die Zeilenfrequenz wird in den Monitorhandbüchern auch als Horizontal- oder einfach H-Frequenz bezeichnet. Bei Festfrequenzmonitoren gibt es ganz genaue Vorgaben, mit welchen Frequenzen der Monitor angesteuert werden muß. Aber auch Multisyncmonitore können nur in einem bestimmten Frequenzbereich betrieben werden. Besonders wenn die Zeilenfrequenz bei Multisyncmonitoren hoch ausgereizt wird, muß darauf geachtet werden, daß die Bildwiederholfrequenz nicht über die maximale Vertikalfrequenz steigt.

—

Beim Ausprobieren neuer Konfigurationen kann es unmöglich sein, den X-Server ordentlich mit dem Window-Manager zu beenden. Um diese Situation zu vermeiden, sollte immer auch der oben genannte "640x480" Modus aktiviert sein. Wenn aus diesem Modus heraus mit der 'ALT-ctrl-NUMPLUS' Tastenkombination in den neuen Modus umgeschaltet wird, kann mit der 'ALT-CTRL-NumMinus' immer wieder dahin zurückgeschaltete werden.

Wenn nichts mehr zu sehen ist, Hilft die Tastenkombination 'ALT-CTRL-BACKSP', die den Server sofort beendet.

—

Bei einer Punkttaktfrequenz von xx MHz werden in einer Mikrosekunde (μs) genau xx Punkte auf den Bildschirm geschrieben.

Die Punktzahlen für den Horizontalen Bildaufbau müssen durch 8 Teilbar sein!

Für die vordere Schwarzscherle jeder Bildzeile ist ein Wert von $1 \mu s$ für einen ersten Versuch gut. Der Wert kann weiter verringert werden. Wenn der RECHTE Bildschirmrand Schatten oder Fransen aufweist oder der Mauszeiger hinter dem rechten Bildschirmrand verschwindet, muß die Schwarzscherle verlängert werden.

Für den horizontalen Synchronimpuls ist ein Wert von $2 \mu s$ als erste Annäherung OK. Wenn dieser Wert zu klein ist, macht sich das durch ein horizontales (nach rechts oder links) Durchlaufen des Bildes bemerkbar.

Die hintere Schwarzscherle der Bildzeilen kann im ersten Anlauf mit $1,5 \mu s$ angelegt werden. Analog zur vorderen Schwarzscherle muß die hintere Schwarzscherle verlängert werden, wenn am LINKEN Bildschirmrand Schatten auftauchen oder der Mauszeiger hinter dem linken Bildschirmrand verschwindet. Gewisse Rückkopplungen mit der Länge und Lage des Synchronimpulses sind zu erwarten.

—

Für den vertikalen Bildaufbau brauchen die Zeilenzahlen nicht mehr durch 8 Teilbar zu sein.

Der vertikale Synchronimpuls ist mit zwei Zeilen in der Regel lang genug.

Die vordere und hintere (obere und untere) Schwarzscherle kann im ersten Anlauf mit je 10 Zeilen angenommen werden. Eine zu kurze vordere Schwarzscherle macht sich wieder mit entsprechenden Fehlern am unteren Bildschirmrand bemerkbar; für die hintere Schwarzscherle gilt entsprechendes für den oberen Bildschirmrand.

Anhang A

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE

Terms And Conditions For Copying, Distribution And Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program“, below, refers to any such program or work, and a “work based on the Program“ means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification“.) Each licensee is addressed as “you“.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program’s source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
- (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 - (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
 - (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 - (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version“, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS“WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Index

- ~, 19
- /dev/cua?, 170
- /dev/fd, 170
- /dev/lp, 171
- /dev/null, 171
- /dev/tty, 172
- /dev/ttyS, 172
- /etc/fstab, 163
- /etc/group, 164
- /etc/hosts, 165
- /etc/inittab, 165
- /etc/issue, 167
- /etc/magic, 167
- /etc/motd, 167
- /etc/nologin, 167
- /etc/passwd, 167
- /etc/profile, 168
- /etc/psdatabase, 168
- /etc/rc, 168
- /etc/securetty, 169
- /etc/shells, 169
- /etc/termcap, 169
- ^\\, 139
- ^?, 139
- ^C, 139, 181
- ^D, 139
- ^N, 139
- ^O, 139
- ^Q, 139
- ^R, 139
- ^S, 139
- ^V, 139
- ^W, 139
- ^Z, 24, 139
- absoluter Name, 162
- alias Shellkommando, 32
- Aliases, 23
- ar, 9
- Arbeitsspeicher, 90
- Ausführbarkeit, 184
- Ausgabeumleitung, 22
- basename, 10
- Bash, 11
- Aufruf, 11
- beenden einer Shell, 35
- Benutzer
 - Gruppe wechseln, 119
 - Kennung, 99
 - Name, 108
- bg Shellkommando, 32
- Bildschirmausgabe speichern, 145
- Binärdateien anzeigen, 97, 122
- Binärzahl, 122
- bind Shellkommando, 32
- Blockdepot, 90, 142
- blockorientiertes Gerät, 170
- break Shellkommando, 32
- builtin Shellkommando, 32
- Byte, 122
- Cache, 90, 142
- case, 14
- cat, 44
- cd Shellkommando, 33
- CDPATH Shellvariable, 16, 33
- Character, 122
- chgrp, 44
- chmod, 45
- chown, 151
- chsh, 46
- cksum, 47
- Clocks, 209
- cmp, 47
- comm, 48
- command Shellkommando, 33
- compress, 49
- continue Shellkommando, 33
- cooked Modus, 138
- core, 42, 180
- cp, 50
- csplit, 51
- cut, 51
- date, 52
- Dateien
 - anzeigen, 97, 101, 115, 122, 142
 - drucken, 126, 171
 - durchsuchen, 58, 91

- komprimieren, 49, 95
- kopieren, 50, 54
- löschen, 130
- Namen, 153, 154
- sortieren, 134
- speichern, 54, 143
- suchen, 84
- teilen, 51, 135
- Typ feststellen, 84
- umbenennen, 118
- verdoppeln, 107
- vergleichen, 47, 48
- zusammenfügen, 44, 100, 125
- Dateinamen Länge, 153
- Dateisystem, 161
- Datum, 52
- dd, 54
- declare Shellkommando, 33
- Devices, 162, 170
- df, 55
- dial in, 172
- dial out, 170
- dir, 56
- dirname, 31, 56
- dirs Shellkommando, 33
- Disketten
 - formatieren, 83
 - Inhalt, 113
 - kopieren, 112
 - löschen, 112
 - lesen, 117
 - MS-DOS Dateisystem, 113
 - Operationen, 118
 - Verzeichnis anlegen, 115
 - Verzeichnis löschen, 117
- doshell, 56
- DR-DOS, 196
- drucken, 126, 171
- Drucker, 171
- du, 57
- echo Shellkommando, 34
- Editor, 60
- efsck, 152
- egrep, 58
- Eigentümer
 - ändern, 151
 - einer Datei, 184
- Eingabe
 - für Shellscrip, 37
- Eingabeaufforderung, 25
- Eingabeumleitung, 21
- elvis, 60
- enable Shellkommando, 34
- Ende der Eingabe, 17
- env, 80
- Environment, 35, 80
- EOF, 17
- eval Shellkommando, 34
- ex, 60
- exec Shellkommando, 34
- exit Shellkommando, 35
- expand, 81
- export Shellkommando, 35
- expr, 82
- fc Shellkommando, 35
- fdformat, 83
- fdisk, 152
- Festplatten, 142, 153, 171
 - booten, 175
 - freier Platz, 55
 - partitionieren, 152
 - SCSI, 171
- fg Shellkommando, 35
- file, 84
- Filesystem, 161
- find, 84
- Floppylaufwerke, 170
- fold, 90
- for, 13
- free, 90
- fsck, 152
- fstab Datei, 160, 163
- Funktionen im Shellscrip, 14, 23
- Funktionsbibliothek, 9
- Gerätedateien, 170
- grep, 91
- groff, 93
- group Datei, 164, 183
- groups, 95
- Gruppe
 - ändern, 44
 - wechseln, 119
- gzip, 95
- halt, 159
- Handbuchseiten, 110
 - formatieren, 93
- harter Link, 107
- hash Shellkommando, 36
- head, 97
- Heimatverzeichnis, 16
- help Shellkommando, 36
- Hexadezimalzahl, 123
- hexdump, 97
- Hilfe
 - für Shellkommandos, 36

- zu Kommandos, 110
- Hintergrundprozesse, 24
 - anzeigen, 36
 - beenden, 36
 - starten, 32, 35
- History, 30
- history Shellkommando, 36
- HOME Shellvariable, 16, 19, 33
- hostname, 98
- hosts Datei, 165
- id, 99
- if, 14
- IFS Shellvariable, 16
- inittab Datei, 165
- install, 99
- issue Datei, 167
- Job Control, 24
- jobs Shellkommando, 36
- join, 100
- kill, 101
- kill Shellkommando, 36
- Klammern i.d. Shell, 13
- Kommandos verketten, 13
- Kommandosubstitution, 20
- Kommandozeile, 11
- Kommandozeileneditor, 26
- Kommandozeilenparameter, 15
- Kommandozeilenspeicher, 30
- kopieren v. Dateien, 50
- löschen, 130
- less, 101
- let Shellkommando, 37
- Lilo, 175
- Link, 107
- Listen v. Kommandos, 13
- ln, 107
- local Shellkommando, 37
- login, 108
- Loginshell, 43
 - ändern, 46
- logname, 108
- logout Shellkommando, 37
- loopback device, 177
- ls, 109
- Mülleimer, 171
- magic Datei, 167
- Major Device Number, 170
- man, 110
- mcopy, 112
- mdel, 112
- mdir, 113
- mformat, 113
- Minor Device Number, 170
- mkdir, 114
- mkefs, 154
- mkfifo, 114
- mkfs, 153
- mknod, 154
- mkswap, 156
- mmd, 115
- Modem, 170, 172
- more, 115
- motd Datei, 167
- mount, 156
- mrd, 117
- mread, 117
- MS-DOS
 - Dateisystem, 113, 118
 - Disketten, 118
- mttools, 118
- mv, 118
- Nachricht
 - an einen Benutzer, 150
- named Pipe, 114
- newgrp, 119
- nice, 119
- nl, 120
- nohup, 121
- nologin Datei, 167
- nroff, 93
- od, 122
- Oktalzahl, 122
- Paßwort ändern, 125
- Packer, 49, 95
- Parameter i.d. Shell, 15
- Parametererweiterung, 19
- passwd, 125
- passwd Datei, 182
- paste, 125
- PATH Shellvariable, 16, 19, 25, 29
- Pfad, 162
- Pipelines, 13
- popd Shellkommando, 37
- Positionsparameter, 15
- pr, 126
- Prüfsummen, 47, 141
- printenv, 127
- profile Datei, 168
- Prompt, 17, 25
- Prozeß
 - beenden, 101
 - Nummer, 127

- Status, 127
- Tabelle, 128
- ps, 127
- PS1 Shellvariable, 17
- psdatabase Datei, 168
- pushd Shellkommando, 37
- pwd, 129
- pwd Shellkommando, 37
- Quotierung, 14
- Rückgabewert, 13
- Ramdisk, 158, 171
- raw Modus, 138
- rc Datei, 168
- rdev, 158
- read Shellkommando, 37
- readonly Shellkommando, 38
- reboot, 159
- return Shellkommando, 38
- rm, 130
- rmdir, 130
- Rootfilesystem, 158, 163
- Runlevel, 166
- Schleifen
 - abbrechen, 32, 33
 - im Shellscrip, 13, 14
- Schreibschutz, 184
- Scriptfunktionen, 14
- securetty Datei, 169
- sed, 131
- serielle Schnittstelle, 170, 172
- set Shellkommando, 38
- SGID, 185
- Sheduler, 119
- Shell, 11
 - beenden, 35, 37
- Shellkommando
 - an/abschalten, 34
 - aufrufen, 32
- Shelllevel, 16
- shells Datei, 169
- Shellscrip, 11
- Shellvariable, 16
 - erzeugen, 33
 - löschen, 43
- shift Shellkommando, 39
- shutdown, 159
- Signale
 - abfangen, 41
 - senden, 36, 101, 180
 - SIGHUP, 181
 - SIGINT, 181
 - SIGKILL, 181
 - SIGSEGV, 180
 - SIGTERM, 36, 101
- sleep, 133
- SLS Distribution, 197
- sort, 134
- sortieren, 134
- source Shellkommando, 39
- Speicherplatz auf Platte, 55
- Spezialparameter, 15
- split, 135
- Status, 13
- Sticky Bit, 185
- stty, 136
- su, 140
- suchen
 - Ausdrücke in Dateien, 58, 91
 - einer Datei, 84
- Suchpfad, 16, 25
- SUID, 185
- sum, 141
- suspend Shellkommando, 39
- swapon, 205
- swapping, 156
- symbolischer Link, 107
- sync, 141
- Synonyme, 23
- sysinstall (SLS), 202
- Systemzeit, 52
- Tabulatoren
 - ersetzen, 81
- tac, 142
- tail, 142
- tar, 143
- Tastaturkommandos ändern, 32
- Tastatortreiber ändern, 205
- tcp/ip, 177
- tee, 145
- teilen v. Dateien, 51
- termcap Datei, 169
- Terminal
 - einstellen, 136
 - virtuell, 56, 172
- test Shellkommando, 40
- Texte formatieren, 93
- Tildenerweiterung, 19
- times Shellkommando, 41
- token, 12
- touch, 145
- trap Shellkommando, 41
- tty, 146
- type Shellkommando, 41
- ulimit Shellkommando, 42

- umask Shellkommando, 42
- umbenennen, 118
 - von Kommandos, 23
- Umgebung, 35, 80
- Umleitung, 21
- umount, 159
- unalias Shellkommando, 42
- uname, 147
- uncompress, 49
- uniq, 147
- unset Shellkommando, 43
- until, 14
- US Tastatur, 197

- Verdoppeln einer Datei, 107
- vergleichen
 - Dateien, 47, 48
 - Zeilen, 147
- verketteten v. Dateien, 44
- verschieben, 118
- Verzeichnis
 - aktuelles anzeigen, 37, 129
 - anlegen, 114
 - auf MS-DOS Disketten, 113
 - aufflisten, 56, 109
 - durchsuchen, 84
 - löschen, 130
 - wechseln, 33
- Verzeichnisbaum, 162
- Verzweigung
 - im Shellscrip, 14
- vi, 60
- view, 60
- virtuelles Terminal, 56

- wait Shellkommando, 43
- wall, 150
- wc, 148
- while, 14
- who, 149
- Worte zählen, 148
- write, 150
- Wurzelverzeichnis, 163

- X-Server, 206
- Xconfig, 207

- zählen, 148
- zcat, 49, 96
- zdiff, 49
- zeichenorientiertes Gerät, 170
- Zeichensätze, 139
- Zeilen
 - nummerieren, 120
 - umbrechen, 90
 - vergleichen, 147
- zmore, 49
- Zugriffsrechte, 42, 184
 - ändern, 45