# LinuX

## Anwenderhandbuch

und

## Leitfaden für die Systemverwaltung

3. erweiterte und aktualisierte Auflage

**Hetze, Sebastian; Hohndel, Dirk; Müller, Martin; Kirch Olaf u.a.:**
LinuX Anwenderhandbuch und Leitfaden für die Systemverwaltung
3. erweiterte und aktualisierte Auflage
LunetIX Softfair 1993 1994
ISBN 3-929764-02-4

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.

Auch wenn wir eine Menge Fehler beseitigen konnten, möchten wir den Eindruck vermeiden, es handele sich jetzt um ein fertiges Produkt. Die dicken Fehler sind nur besser versteckt ;-)

Vielleicht haben wir ja die Chance es beim nächsten Mal besser zu machen. Für Hinweise sind wir immer noch 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.



ISBN 3-929764-02-4

## Foreword by Linus Torvalds

I could start this off with something like "When I started writing Linux, I could only dream of the success that the system has enjoyed and that somebody would write a book about it", but even that would be lying — I couldn't even have dreamed of it. Oh, every programmer hopes to write something that people find useful and will enjoy, but having an actual *real book* about the system is something that makes me very happy.

What makes me even happier is that I didn't even have to write the book myself: I have enjoyed the fun parts of the system (coding and watching it all come to life) without having to deal with the bothersome details of documenting it for users and developers alike. So when Dirk Hohndel asked me to write a little foreword for this book, I was delighted. Not least because we have been communicating electronically since the very early Linux days, and even had a chance to meet in person at the 1993 NLUUG conference. So this is it:

I hope you'll find the book useful and that you'll enjoy it as much as I do,

Linus Torvalds
Helsinki, 28.12.93

---

## Vorwort zur dritten Auflage

Die Entwicklung von Linux ist im vergangenen halben Jahr seit dem Erscheinen der zweiten Auflage des Linux–Anwenderhandbuches nicht weniger stürmisch verlaufen als in den Monaten davor. Die vielfältigen Veränderungen haben eine Neuauflage des Handbuches dringend erforderlich gemacht.

Wir freuen uns ganz besonders, daß diese Auflage zusammen mit der ersten "offiziellen" Linux–Version 1.0 erscheinen kann. Die außerordentlich schnelle Herstellung und unser direkter Kontakt zu Linus Torvalds erlauben es uns, Ihnen in diesem Buch bereits alle Features von Linux 1.0 vorzustellen.

Mit wachsender Verbreitung von Linux ist das Linux–Anwenderhandbuch auch ein finanzieller Erfolg. Obwohl es als FreeWare–Produkt verteilt und ausgedruckt werden darf, sind von der zweiten Auflage ca. 7000 Exemplare verkauft worden. Vom Verkaufserlös haben wir einen Teil zur Finanzierung weiterer FreeWare–Projekte abgegeben, wir haben aber auch die Gelegenheit genutzt, das Autoren–Team zu verstärken und das Handbuch erheblich zu erweitern.

Besonders hervorzuheben ist hier das neue Kapitel über XFree86 von Dirk Hohndel, in dem das wichtige und interessante Thema der Konfiguration und Benutzung des X Window Systems mit einzigartiger Kompetenz beschrieben wird. Die Kapitel über UUCP–Vernetzung und das smail–Programm, die von Olaf Kirch, dem Autor des "Network Administation Guide", beigesteuert wurden, führen in das aktuelle Thema der Vernetzung über USENET und Internet ein.

Mit der Überarbeitung und den Erweiterungen haben wir viele Fehler und Schwachstellen früherer Auflagen beheben können. Der Einstieg in die Linux-Welt sollte Ihnen mit diesem Buch keine Probleme bereiten.

Wenn Sie inhaltliche Fehler finden oder Kritik und Anregungen zu unserem Buch haben, würden wir uns sehr freuen von Ihnen zu hören. Diese Rückkopplung ist wohl Wunsch eines jeden Autors, für die weitere Arbeit an dem Linux-Anwenderhandbuch ist sie dringend notwendig.

# Danksagung

Die vorliegende dritte Auflage des Linux Anwenderhandbuches ist nicht das Produkt der Autoren allein — an dem Gelingen des Werkes haben viele Menschen direkt oder indirekt mitgewirkt, denen wir hiermit herzlich danken wollen.

Zuerst danken wir allen Entwicklerinnen und Entwicklern Freier Software für Ihre großartigen Produkte. Die Faszination der "GNU Generation“ und der Wunsch, selbst aktiver Teil davon zu sein, ist dauernde Motivation für unsere Arbeit. Insbesondere gilt unser Dank Linus Torvalds, der uns das unerschöpfliche Thema unseres Buches liefert.

Den Mitarbeiterinnen und Mitarbeitern der Firma J.F. Lehmanns, besonders Bernd Sommerfeld, danken wir für ihre Unterstützung bei der Vermarktung des Buches und für die dauernde Ermutigung zu unserer Arbeit. Ohne Sie hätte es das Handbuch vielleicht nie gegeben.

Allen Leserinnen und Lesern, die uns geschrieben haben, danken wir für Lob, Anregungen und Kritik. Wir wünschen uns viel mehr davon :-) Denjenigen, die unser Buch gekauft haben oder es jetzt kaufen werden danken wir für die finanzielle Untersützung unseres Projektes.

Wir danken Sven Schüle, Olaf Titz, Ralf Flaxa und Christian Wildenhain für ihre Anregungen und für ihre Unterstützung auf dem Weg vom Manuskript zum fertigen Buch.

Und — last not least — danken wir Ruth dafür, daß sie die Rechner unseres Netzes (fast) immer am Laufen hält.

Nur durch die Summe der Einzelleistungen ist es möglich, daß dieses Handbuch weiterhin im Selbstverlag herausgegeben wird.

Berlin, Würzburg, den 30.12.1993

Sebastian Hetze (she@lunetix.de)
Dirk Hohndel (hohndel@lunetix.de)
Olaf Kirch (okir@lunetix.de)
Martin Müller (mm@lunetix.de)

# Vorwort zur ersten Auflage

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 herumkommen.

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 Menschen und Völkern der "anderen" Länder und Kontinente entwickelt werden, nie gegen sie.
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 entschuldigen diese Taten nicht.

Für uns gilt weiterhin:

**Schaut nicht weg!** **Greift ein!**

Berlin, den 30.12.1993
Sebastian Hetze, Dirk Hohndel, Olaf Kirch und Martin Müller
LunetIX Softfair

# Inhaltsverzeichnis

# Kapitel 1

# Grundlagen

## 1.1 Geschichte

### 1.1.1 Steinzeit

In grauer Vorzeit wurden Elektronenrechner mit Lochkarten gefüttert. Die Programme und die Daten wurden in die Karten gestanzt und als Pakete in den Kartenleser geschoben, die Ergebnisse auf dem Drucker ausgegeben. Klar, daß solche Rechner nur ein einziges Programm für einen einzigen Anwender bearbeiten konnten. Das Betriebssystem, wenn man es überhaupt so nennen kann, bestand aus den fest verdrahteten Funktionen des Rechenwerkes.

Auch als die Daten und Programme von Magnetspeichern gelesen und die Ergebnisse auf Bildschirmen statt auf Druckern ausgegeben wurden, blieb es bei der Batchverarbeitung durch den Großrechner. Zwar konnten bereits mehrere Jobs in einem Stapel (Batch) geladen werden, es blieb aber bei der sequentiellen Bearbeitung der Jobs.

Durch die wachsende Rechenleistung und vor allem durch die bessere Ausstattung mit Arbeitsspeicher können seit der dritten Rechnergeneration mehrere Jobs gleichzeitig in den Arbeitsspeicher geladen werden. Auf diese Weise kann die wertvolle Rechenzeit (CPU–Zeit) auch dann ausgenutzt werden, wenn ein Job beispielsweise auf neue Daten vom Magnetband warten muß (indem das Betriebssystem einen anderen Job im Speicher laufen läßt, bis die Daten vom Magnetband geladen sind).

Die Bearbeitung kompletter Jobs wurde in den sechziger Jahren durch das quasiparallele Bearbeiten mehrerer Programme im Timesharing–Verfahren abgelöst. Dabei werden mehrere Programme geladen und allen Programmen wird nach einem festgelegten Verfahren Rechenzeit in Zeitscheiben zugeteilt. Mit dieser Methode können interaktive Programme zur Datenerfassung gleichzeitig mit den rechenzeitintensiven Programmen zur Datenauswertung laufen. Durch diese Mischung wurde eine weitaus bessere Auslastung der teuren Hardware möglich.

Für die interaktive Arbeit am Großrechner werden Bildschirm–/Tastatur–Kombinationen, sogenannte Terminals, angeschlossen. Solche Terminals schicken die auf der Tastatur eingegebenen Zeichen unverändert an den Rechner und stellen die vom Rechner gesendeten Zeichen auf dem Bildschirm dar. Die meisten Terminals, zum Beispiel das vt100 Terminal von DEC, erkennen neben den normalen Buchstaben verschiedene Sonderzeichen zum Löschen eines Buchstabens oder des gesamten Bildschirms, zum Positionieren der Einfügemarke, für invertierte Darstellung und so weiter.

### 1.1.2 Unix

Das erste Unics[1] wurde 1969 von Ken Thompson und Dennis Ritchie bei den Bell Laboratories (AT&T und Western Electric) geschrieben (um auf einer wenig benutzten DEC PDP–7 in einer Ecke Space Travel zu spielen ...). Bis 1971 war es als Version 1 auf eine PDP–11 portiert; Version 4 wurde 1973 von dem

[1] Der ursprüngliche Name ist abgeleitet von MULTICS, einem Betriebssystemprojekt, an dem auch die Bell Laboratories beteiligt waren.

ursprünglich in Assembler geschriebenen Quelltext fast vollständig in die eigens dafür entwickelte Hochsprache C umgeschrieben.

Weil AT&T durch Verträge mit der US–Bundesregierung (per Gerichtsbeschluß?) daran gehindert war, Unix zu vermarkten, gab sie für Lehr– und Forschungszwecke Sourcelizenzen zu sehr günstigen Konditionen (ein paar hundert Dollar) an Universitäten weiter. Dieser möglicherweise gut gemeinte, jedenfalls außerordentlich geschickte Schritt führte zu einer sehr dynamischen Entwicklung von Unix. Es verbreitete sich schnell, und aus den Universitäten flossen viele Ideen in die Entwicklung des Systems ein.

Seit Version 6 (1975) hat AT&T aber auch kommerzielle Lizenzen für Unix verkauft. Spätestens seit dem Jahr 1984, als ein weiteres Gerichtsurteil AT&T die Vermarktung von Software erlaubte, wurde Unix als System V unter rein kommerziellen Gesichtspunkten verbreitet.

Gerade wegen seiner Verbreitung an den Universitäten hat es sich im kommerziellen Sektor schnell durchgesetzt.[2] Da fast komplett in C geschrieben, ist es auf praktisch alle Großrechnerarchitekturen portiert. Für Softwareanbieter ist Unix ein (zur Hardwareseite) offenes System. Aus eifersüchtiger Sorge um die Quelltexte hat AT&T die tiefgreifendere Öffnung auf Systemebene abgeblockt.

### 1.1.3 Minix

Im Jahr 1987 hat Andrew Tanenbaum, Professor an der Freien Universität von Amsterdam, ein Lehrbetriebssystem für PC veröffentlicht, das ohne jeden AT&T Code die Funktionalität von Unix Version 7 hat und als Quelltext für wenig Geld zu kaufen ist.

Minix ist keine echte Basis für Anwenderprogramme; aber es ist ein sehr lehrreiches Spielzeug. Ein Lebensnerv von Minix ist das USENET, wo in der Gruppe comp.os.minix alle Neuigkeiten, Fragen und Antworten zu Minix ausgetauscht werden. Hier werden auch Veränderungen am Betriebssystem (dem **Kernel**)[3] veröffentlicht und gelegentlich ganze Programme verschickt. Hier tummelt sich eine weltweit aktive Minix–Gemeinde und entwickelt das Betriebssystem und die Anwendungen drumherum.

Minix wurde auf den Atari ST, den Amiga und den Apple Macintosh portiert; und es wurde an die erweiterten Möglichkeiten des Intel–386–Prozessors angepaßt. Auf diese Weise konnte die lästige Beschränkung auf 64 Kilobytes Speicher je Prozeß überwunden werden.

Es sollten und sollen aber andere Beschränkungen für Minix bestehen bleiben, auf denen Andrew Tanenbaum als Autor besteht. So ist Minix preiswert, aber nicht frei kopierbar. Und es werden keine grundlegenden Veränderungen am Kernel zugunsten von Anwendungen vorgenommen, was zum Beispiel die Portierung der unter Unix weit verbreiteten grafischen Benutzeroberfläche, des X Window Systems, ausschließt.

Diese Einschränkungen sind im Sinne eines Lehrbetriebssystems sinnvoll, das Bedürfnis nach einem vollwertigen und freien Betriebssystem für die modernen 386–PC's bleibt aber unerfüllt.

### 1.1.4 Linux

Im März 1991 fing Linus Benedict Torvalds in Helsinki damit an, die Möglichkeiten des Intel–386–Prozessors in seinem neuen PC zu studieren. Er hatte das 386er Minix installiert — und damit das C–Entwicklungssystem der Free Software Foundation.

Nur ein halbes Jahr später war aus den Assemblerstudien ein kleines, lauffähiges Betriebssystem entstanden. Als Linus im September 1991 die erste Version (0.01) von Linux an interessierte Minixer verschickte, mußte es noch unter Minix übersetzt und installiert werden. Diese Verbindung von Minix und Linux hat sich aber nicht im Quelltext niedergeschlagen.[4]

---

[2] Allein die Tatsache, daß praktisch jeder Informatiker unter Unix Programmieren gelernt hat, ist ein guter Grund, dieses Betriebssystem auch später für die Entwicklung kommerzieller Produkte einzusetzen.

[3] Der Begriff "Betriebssystem" wird in zwei verschiedenen Weisen benutzt. Das Betriebssystem im engeren Sinne wird auch als Kernel bezeichnet; sonst wird mit Betriebssystem die komplette Installation eines Basissystems mit Kernel, Dateisystem, Shell und Utilities gemeint.

[4] Gerade die ersten Versionen von Linux waren so weitgehend für 386–Assembler geschrieben, daß von einer Verwandschaft der Betriebssysteme auf dieser Ebene nicht die Rede sein kann. Selbst in den architektonischen Grundkonzepten unterscheiden sich Minix und Linux erheblich. (Der "Linux is obsolete" Thread in comp.os.minix hat die unterschiedlichen Konzepte seinerzeit beleuchtet.) Linux ist ein monolithisches System, in dem der Kernel im Adreßbereich des Benutzerprozesses liegt und alle Gerätetreiber Teil dieses Kernels sind. Minix ist ein prozeßstrukturiertes, modulares System, in dem die Gerätetreiber eigene Prozesse sind.

Linus Torvalds hat seine eigene Entwicklung von Anfang an frei angeboten. Jeder kann die Quelltexte bekommen und daran mitarbeiten.

Die im Januar 1992 herausgegebene Version 0.12 war bereits ein stabil laufender Kernel. Es gab den GNU C–Compiler, die bash, uemacs und viele der GNU Utilities. Dieses Betriebssystem wurde in comp.os.minix angekündigt und per anonymous FTP weltweit verteilt.

Die Zahl der Programmierer, Tester und Unterstützer wuchs in diesen Tagen so schnell, daß die Kommunikation per eMail nicht mehr ausreichte und nach dem Beispiel von comp.os.minix die Rubrik alt.os.linux im USENET eingerichtet wurde. Dieses Medium und der anonyme FTP Service im Internet ermöglichten eine Programmentwicklung, wie sie in den Vorstandsetagen der mächtigen Softwareschmieden erträumt wird. Innerhalb weniger Monate wurde aus dem weitgehend in Assembler geschriebenen Minikernel ein ausgewachsenes Betriebssystem mit vollständiger Unix Funktionalität.

Die weitgehende POSIX–Konformität von Linux und die umfangreichen C–Bibliotheken des GNU–C–Compilers erleichtern die Portierung von Unix– oder BSD–Software auf ein Maß, das diesen Namen eigentlich nicht verdient. Praktisch das gesamte Angebot an freier Software läuft auch unter Linux. Dank der enormen Leistungen, die bereits seit vielen Jahren von der Free Software Foundation, dem X Consortium, den Universitäten und ungezählten Organisationen und Einzelpersonen in diesem Bereich erbracht wurden, haben die Linux–Distributionen heute einen Umfang erreicht, der die Angebote kommerzieller Unixe leicht in den Schatten stellt.

## 1.2 Das Betriebssystem

Jeder Computer besteht aus mehreren Komponenten und Geräten, der sogenannten Hardware. Unter anderem sind das die Zentrale–Prozessor–Einheit (CPU), der Arbeitsspeicher, die Festplatte und die Diskettenlaufwerke, der Bildschirm und die Tastatur, der Drucker und das Modem. Die Komponenten sind nur lose miteinander verknüpft. Philosophisch betrachtet, stellen sie eine universelle Maschine dar, die erst durch ein konkretes Anwenderprogramm zu einer simulierten Schreibmaschine, einer Lohnbuchhaltung, einem Schachspiel oder einer Mondlandefähre wird.

Um den Programmierer eines Anwenderprogramms von den Einzelheiten der Hardwareprogrammierung zu entlasten, werden die Komponenten durch das sogenannte Betriebssystem verwaltet. Im Idealfall stellt das Betriebssystem alle Dienste der Hardware in einer abstrakteren Form zur Verfügung, ist also eine Art Hardwareerweiterung.

Das Betriebssystem muß, wie jedes andere Programm auch, zur Laufzeit im Arbeitsspeicher geladen sein. Bei einigen Computern steht das Betriebssystem in dauerhaften Speicherbausteinen, sogenannten ROM's. Die IBM–kompatiblen PC's laden den größten Teil des Betriebssystems von Diskette oder Festplatte.[5] Das Betriebssystem ist das erste Programm, das nach dem Einschalten des Rechners automatisch geladen und gestartet wird. Wenn es einmal geladen ist, bleibt das Betriebssystem im Arbeitsspeicher, bis der Rechner ausgeschaltet wird.

Gemeinsam mit dem Betriebssystem stehen normalerweise noch eine Reihe von Systemprogrammen zur Verfügung, mit denen Sie beispielsweise eine Diskette formatieren oder eine Datei ausdrucken können. Ein besonders wichtiges Systemprogramm ist der Kommandozeileninterpreter, die Benutzeroberfläche des Betriebssystems, der Ihre Eingaben liest und die darin formulierten Befehle ausführt. Bei MS–DOS wird diese Aufgabe von COMMAND.COM erfüllt, unter Linux gibt es zu diesem Zweck sogenannte Shells.

Zur optimalen Ausnutzung der teuren Hardware wurden für Großrechner schon sehr früh Betriebssysteme entwickelt, die mehreren Anwendern gleichzeitig die Systembenutzung ermöglichen. Diese als *Multiuser/Multitasking* bezeichnete Eigenschaft war für das Betriebssystem der ersten PC's (1981) überflüssig. Ein moderner 386er Rechner wird dagegen von einem normalen Anwenderprogramm nicht ausgelastet. Die meiste Zeit verbringt das Betriebssystem damit, auf den nächsten Tastendruck des Benutzers zu warten. Aus diesem Grund werden jetzt auch Mehrbenutzerbetriebssysteme für PC angeboten.

Die Anforderungen an ein Mehrbenutzerbetriebssystem unterscheiden sich grundlegend von einem Einbenutzersystem:

[5] Um das eigentliche Betriebssystem zu laden, haben auch die PC's ein Minimalsystem, das BIOS (Basic Input Output System), im ROM.

- Es müssen konkurrierende Hardwarezugriffe verhindert beziehungsweise verwaltet werden.

- Es müssen die privaten Daten der Benutzer geschützt werden.

- Die Speicherbereiche der Anwenderprogramme müssen vor ungewollten Veränderungen durch andere Programme geschützt werden.

Als echtes Mehrbenutzerbetriebssystem verwaltet Linux die Systemkomponenten sehr restriktiv. Es erlaubt den Anwenderprogrammen prinzipiell keinen direkten Zugriff auf die Hardware.

Um diese Einschränkung durchzusetzen, werden die Anwenderprogramme durch das Betriebssystem (den Kernel) kontrolliert. Wenn ein Programm vom Benutzer aufgerufen wird, lädt der Kernel die ausführbare Datei[6] in den Arbeitsspeicher und macht daraus einen Prozeß. Dieser Prozeß erhält bei seiner Entstehung einen logischen Adreßraum[7], in dem zuerst der Programmtext und die initialisierten Daten des Programms dargestellt werden, in dem das Programm aber auch seine variablen Daten ablegen kann. Die logischen Adressen werden vom Betriebssystem auf die physikalischen Adressen des Arbeitsspeichers abgebildet. Wenn das Programm auf eine (logische) Speicheradresse zugreift, muß diese Adresse erst in die physikalische Adresse umgewandelt werden. Dadurch kann das Betriebssystem unberechtigte Zugriffe auf den Adreßraum anderer Prozesse oder auf die Hardwarekomponenten feststellen und unterbinden (durch das Signal SIGSEGV).

Die einzige Möglichkeit, auf die Systembereiche außerhalb des eigenen Adreßraums zuzugreifen, bietet der Kernel den Programmen auf Benutzerebene durch die sogenannten Systemaufrufe (system calls). Linux bietet ca. 126[8] solcher Systemaufrufe an. Dieses von Unix übernommene Prinzip erscheint auf den ersten Blick vielleicht als Hindernis und Einengung. Bei genauerer Betrachtung stellt man aber die enormen Vorteile fest. So ist es der Übereinstimmung mit den Systemaufrufen des Unix System V zu verdanken, daß praktisch alle Unix–Programme sofort unter Linux übersetzt werden können. Die unterschiedlichen Hardwarevoraussetzungen all der unterschiedlichen Systeme werden allein vom Kernel aufgefangen.

Wie bereits gesagt, arbeitet jedes Anwenderprogramm in einem logischen Speichersegment. Dieses Speichersegment ist in Speicherseiten zu je 4 Kilobyte unterteilt und wird vom Betriebssystem seitenweise auf den physikalischen Arbeitsspeicher abgebildet (mapping). Wenn mehr Programme gestartet werden, als auf einmal in den Arbeitsspeicher passen, kann der Kernel einzelne Speicherseiten aus dem physikalischen Adreßraum auf Festplatte auslagern (swapping). Wenn das dadurch unvollständige Programm wieder auf eine Adresse der ausgelagerten Speicherseite zugreifen will, wird sie automatisch zurückgeladen. Dank der MMU (Memory Management Unit) des 386–Prozessors werden die grundlegenden Funktionen dieser aufwendigen Speicherverwaltung bereits durch die CPU erledigt. Die intensive Ausnutzung spezieller Prozessoreigenschaften macht Linux zu einem außerordentlich schnellen Betriebssystem.

---

[6] Tatsächlich wird nach dem Programmstart nur ein erster Teil der ausführbaren Datei geladen. Weitere Teile werden in Speicherseiten zu je 4 Kilobyte hinzugeladen, sobald das Programm darauf zugreift (demand loading).

[7] Der logische Adreßraum jedes Prozesses ist 4 Gigabyte groß. Neben dem maschinenlesbaren Code (Text) und den Daten werden die von allen Programmen gemeinsam benutzten *shared libraries* und der Kernel in dem Adreßraum abgebildet. Natürlich kann nur ein kleiner Teil des Adreßraumes tatsächlich sinnvoll angesprochen werden; der Zugriff auf eine falsche Adresse führt in der Regel zum sofortigen Programmabbruch.

[8] Linux 1.0 kennt genau 126 direkte Systemcalls. Manche dieser Systemaufrufe fassen eine ganze Gruppe sinnverwandter Funktionen zusammen, beispielsweise sys_socketcall(9) alles was mit Sockets gemacht werden kann. In der C–Bibliothek werden alle Systemaufrufe durch mindestens eine C–Funktion bereitgestellt. Systemcalls wie sys_socketcall(9) werden auf der Ebene der C–Bibliothek nur indirekt durch viele Einzelfunktionen repräsentiert (accept(2), bind(2), connect(2), getpeername(2), getsockname(2), getsockopt(2), listen(2), recv(2), recvfrom(2), send(2), sendto(2), setsockopt(2), shutdown(2), socket(2), socketpair(2)). Wegen ihrer unterschiedlichen Funktion können alle diese Aufrufe jeweils als eigener Systemcall gezählt werden. Auf diese Weise erhöht sich die Zahl der Systemaufrufe über 150.

## 1.3 Die ersten Schritte

Vor die interaktive Benutzung von Linux haben die Wizards die Loginprozedur gesetzt.[9] Jeder Benutzer muß sich beim Betriebssystem mit seinem Benutzernamen anmelden. Diese Anmeldung ist notwendig, damit eine eindeutige Zuordnung von Dateien zu ihren Eigentümern hergestellt werden kann und um den unberechtigten Zugriff auf Dateien anderer Eigenümer zu verhindern. Nach der Meldung

```
login: _
```

müssen Sie Ihren Benutzernamen eingeben und mit der RETURN Taste abschließen. Die daraufhin erscheinende Frage nach dem

```
Password:
```

muß korrekt mit dem nur Ihnen bekannten Paßwort beantwortet werden. Die eingegebenen Zeichen erscheinen nicht auf dem Bildschirm. Wenn das System die Anmeldung akzeptiert, wird eine sogenannte *Session* eröffnet. Das bedeutet für Sie als interaktiver Benutzer, daß der Kommandozeileninterpreter (die Shell) gestartet wird, so daß Sie interaktiv über die Tastatur Kommandos eingeben können.

Die Shell meldet ihre Bereitschaft durch den sogenannten Prompt. Der Prompt (Eingabeaufforderung) kann unterschiedliche Formen annehmen. Die Prompts der Bourne Again Shell (`bash`) sind auf der Seite 83 beschrieben.

Die Kommandos werden zeilenweise von der Tastatur gelesen und anschließend von der Shell interpretiert, die die entsprechenden Befehle ausführt. Jede Kommandozeile wird durch die RETURN Taste abgeschlossen. Jedes Kommando besteht aus einem oder mehr Wörtern, die durch Leerzeichen voneinander getrennt werden. Das erste Wort ist immer der Kommandoname. Der Kommandoname entspricht exakt dem Namen der ausführbaren Datei.[10] Eine genauere, allerdings auch formalistischere Erklärung der Kommandozeile finden Sie in der Einleitung zu den Kommandobeschreibungen (→ Seite 53) und in der genauen Beschreibung der Standardshell `bash` (→ Seite 54ff).

### 1.3.1 Verzeichnisse und Dateien

Die im Arbeitsspeicher des Rechners gehaltenen Daten, seien es Programme, Texte, Bilder oder Zahlen, gehen alle mit dem Ausschalten der Stromversorgung verloren. Aus diesem Grund werden alle wichtigen Daten auf einem dauerhaften Speichermedium gehalten. Dieses Medium ist normalerweise die Festplatte, es kann aber auch ein Diskettenlaufwerk oder eine CD–ROM sein.

Um die Vielzahl der anfallenden Daten rationell und übersichtlich speichern zu können, bietet Linux, wie alle modernen Betriebssysteme, die Möglichkeit, die Daten in Verzeichnissen zu organisieren.

#### Das aktuelle Verzeichnis

Als Benutzer eines interaktiven Computers "befinden" Sie sich ständig in einem Verzeichnis. Dieses Verzeichnis, das sogenannte **aktuelle Verzeichnis**, ist der relative Bezugspunkt für all Ihre Aktionen. Jedes Kommando, das Sie aufrufen, ist mit diesem Verzeichnis verbunden.

Jedes Verzeichnis kann Dateien und weitere Verzeichnisse enthalten. Der Name des aktuellen Verzeichnisses wird mit dem Kommando `pwd` (print working directory) angezeigt:

```
$ pwd
/usr/src/linux
$ _
```

[9] Die allererste Begegnung mit Linux findet normalerweise vor der Installation auf Festplatte mit einem minimalen System auf Diskette bzw. auf einer RAM–Disk statt.

Das auf den Installationsdisketten der meisten Distributionen enthaltene System gibt einen normalen Login–Prompt aus. Unter dem Benutzernamen `root` können Sie sich ohne Passwort einloggen und haben dann die Benutzerkennung (ID) 0, die mit besonderen Privilegien verbunden ist.

Das Dateisystem auf dem Diskettensystem sieht zwar etwas anders aus als auf dem fertig installierten System, die in diesem Kapitel vorgestellten Kommandos können aber trotzdem ausgeführt werden.

[10] Hiervon ausgenommen sind die internen Kommandos der Shell, die nicht separat als ausführbare Datei existieren.

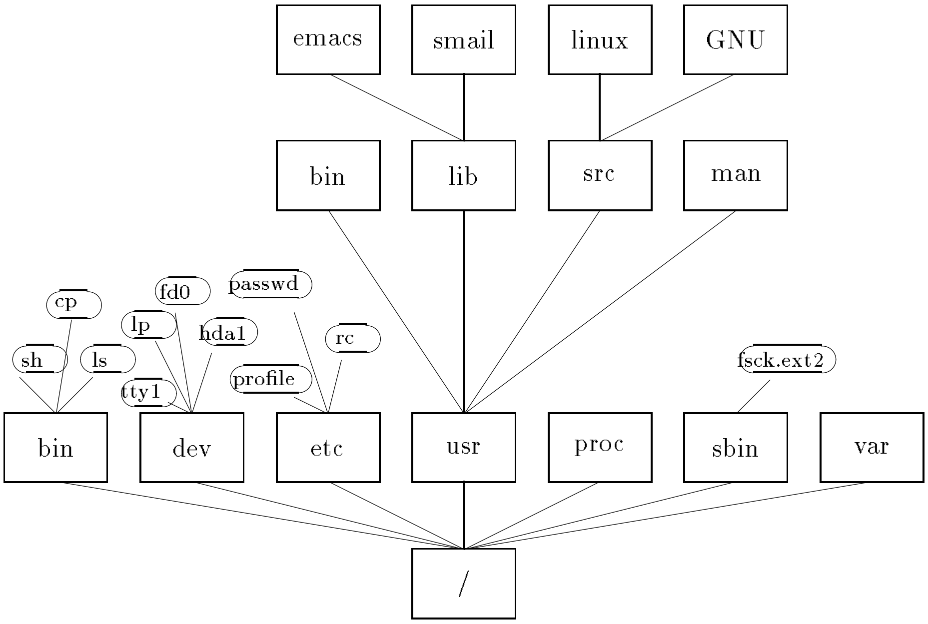


Abbildung 1.1: Dateibaum (Ausschnitt)

Das aktuelle Verzeichnis ist eines unter vielen Verzeichnissen im System. Alle Verzeichnisse und die darin enthaltenen Dateien zusammen werden als Dateisystem bezeichnet. Das Dateisystem ist mit einem Baum zu vergleichen, dessen Äste und Zweige die Verzeichnisse sind und dessen Blätter die Dateien (siehe Abbildung 1.1). Die Stellen, an denen zwei Verzeichnisse oder ein Verzeichnis und eine Datei zusammenstoßen werden durch einen einfachen Schrägstrich '/' (Slash) bezeichnet.

Das Verzeichnis, in dem alle Äste zusammenlaufen, ist das Wurzelverzeichnis oder Rootdirectory $\boxed{/}$.

### Namen und Pfade

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 als **Pfad** bezeichnet. Die einzelnen Verzeichnisnamen werden alle durch '/' (Slash) Zeichen getrennt.

Um 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 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. Damit wird im Prinzip die gleiche Ordnung erzeugt, aber in genau umgekehrter Richtung. Wenn ein Verzeichnis beim Baum näher an der Wurzel, also tiefer ist, ist es in der Hierarchie höher.

### Das Verzeichnis wechseln

Sie können das aktuelle Verzeichnis mit dem `cd` (change directory) Kommando wechseln. Zum Beispiel bringt Sie das Kommando

```
$ cd /
$ _
```

in das Wurzelverzeichnis. Das Kommando

```
$ cd /dev
$ _
```

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

```
$ cd /usr/lib/emacs
$ _
```

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

```
$ cd /usr/lib
$ _
```

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

```
$ cd ..
$ _
```

Die beiden Punkte bezeichnen das übergeordnete Verzeichnis, ein einzelner Punkt steht für das aktuelle Verzeichnis selbst.

### Den Inhalt eines Verzeichnisses anzeigen

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

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

zeigt beispielsweise alle Dateien des bin Verzeichnisses an. Das sind alles ausführbare Programmdateien.[11] Durch Zusätze in der Kommandozeile können Sie das Ausgabeformat des ls Kommandos ändern. Solche Zusätze werden Optionen genannt. Die Optionen werden von einem Minuszeichen eingeleitet und bestehen (fast) immer aus einzelnen Buchstaben, die als Schalter bestimmte Eigenschaften des Programms verändern. Alle Optionen des ls Kommandos sind ab Seite 156 beschrieben. Für den Anfang sind die Optionen `-l` (long) für ein ausführliches Listing,[12] `−a` (all) für die Anzeige aller Dateien sowie `−F` (File) für zusätzliche Symbolisierung bestimmter Dateitypen besonders interessant. Sie können mehrere Optionen hinter einem einzigen Minuszeichen zusammenfassen.

[11] Der Name bin ist eine Abkürzung für binary, die amerikanische Bezeichnung für ausführbare Dateien. Weil die Shell nur in einer begrenzten Anzahl Verzeichnissen nach ausführbaren Programmen sucht, ist diese Zusammenfassung sinnvoll.

[12] Bei einem langen Listing stellt das erste Feld die Zugriffsrechte (permissions) dar. Die Zahl im zweiten Feld ist die Anzahl der Links auf diese Datei bzw. auf dieses Verzeichnis. In den beiden folgenden Feldern ist angegeben, welchem Benutzer und welcher Gruppe die Datei gehört (owner, group). Die Zahl im fünften Feld gibt die Dateigröße in Bytes an. Die auf die Größe folgenden Felder geben das Datum der letzten Veränderung an und das letzte Feld ist der Dateiname.

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

### Eigentum und Zugriffsrechte

Um den unautorisierten Zugriff auf Dateien anderer Benutzer zu verhindern, haben alle Dateien und auch alle Verzeichnisse Eigentümer, die die Zugriffsrechte (Permissions) der anderen Benutzer bestimmen können.

Zur flexibleren Gestaltung dieser Regelung existieren neben dem Eigentümer noch zwei Benutzerkategorien: die Gruppe des Eigentümers und die übrigen Anwender. Die Gruppe wird jedem Benutzer durch einen Eintrag in der Paßwortdatei fest zugeordnet.

| Typ | Eigentümer | | | Gruppe | | | Andere | | |
|---|---|---|---|---|---|---|---|---|---|
| d | r | w | x | r | - | x | r | - | x |

Abbildung 1.2: Zugriffsrechte für ein Verzeichnis

| Typ | Eigentümer | | | Gruppe | | | Andere | | |
|---|---|---|---|---|---|---|---|---|---|
| - | r | w | - | r | w | - | r | - | - |

Abbildung 1.3: Zugriffsrechte für eine Datei

Die Rechte werden bei einem langen Listing mit `ls -l` (wie im Beispiel oben) im ersten Feld angezeigt. Dabei bedeuten:

**r** (read) Leseberechtigung (bei Verzeichnissen Auflistbarkeit)

**w** (write) Schreibberechtigung

**x** (execute) Ausführberechtigung (bei Verzeichnissen die Möglichkeit, in dieses Verzeichnis zu wechseln.)

Die Abbildungen 1.2 und 1.3 zeigen typische Einträge für ein Verzeichnis und eine normale Datei.

Ein Verzeichnis ist durch den Typ d gekennzeichnet. In dem Beispiel hat der Eigentümer das Recht, in das Verzeichnis zu schreiben (das heißt Dateien darin anzulegen oder zu löschen), das Verzeichnis aufzulisten und auf die Dateien in dem Verzeichnis zuzugreifen. Die Gruppe und die anderen Systembenutzer können das Verzeichnis anzeigen und mit cd dorthin wechseln, sie können aber keine Datei in dem Verzeichnis anlegen oder eine Datei aus dem Verzeichnis löschen.

Die Datei in dem Beispiel kann von dem Eigentümer und der Gruppe gelesen und beschrieben (verändert) werden, während die anderen Benutzer die Datei nur lesen können.

### Verzeichnisse erstellen und löschen

Mit dem mkdir (make directory) Kommando können Sie ein neues Verzeichnis anlegen, vorausgesetzt, Sie haben Schreibberechtigung für das Verzeichnis, in dem das neue Verzeichnis erstellt werden soll.

Das rmdir (remove directory) Kommandoist dazu da, Verzeichnisse zu löschen. Es können nur leere Verzeichnisse gelöscht werden.

### Dateien erstellen und löschen

Eine Datei wird von einem Programm als Resultat eines bestimmten Prozesses erstellt. Beispielsweise werden die Programme selbst vom C–Compiler gemacht. Textdateien können Sie einfach mit einem Editor wie zum Beispiel dem vi oder dem emacs erzeugen oder erweitern.

Es gibt aber auch andere Methoden, Dateien zu erzeugen. Beispielsweise können Sie die Kopie einer existierenden Datei mit dem cp (copy) Befehl anfertigen.

Vorausgesetzt, im aktuellen Verzeichnis existiert eine Datei namens `Adressen` und Sie haben Leserecht für die Datei und Schreibberechtigung für das Verzeichnis, erzeugt der Befehl

```
$ cp Adressen Telefonliste
$ _
```

eine genaue Kopie der Datei `Adressen` mit dem Namen `Telefonliste`. Die neue Datei `Telefonliste` ist genauso groß wie `Adressen` und enthält exakt die gleichen Zeichen. Sie belegt zusätzlichen Platz auf der Festplatte, und Veränderungen an der einen Datei sind in der anderen nicht zu sehen.

Das ist nicht selbstverständlich. Mit dem ln (link) Kommando können Sie eine andere Art Kopie erzeugen, bei der dieselbe Datei "nur" einen zusätzlichen Namen erhält. Zum Beispiel

```
$ ln Adressen Adressliste
$ _
```

erzeugt einen neuen Eintrag `Adressliste` im aktuellen Verzeichnis. Das Listing des Verzeichnisses zeigt beide Dateien mit der gleichen Größe, mit den gleichen Zugriffsrechten und mit der gleichen Zeitmarke an. Es **ist** dieselbe Datei, die einfach einen zweiten Namen bekommen hat. Wenn Sie auf die Daten zugreifen wollen, sei es zum Lesen oder Schreiben, macht es keinen Unterschied, unter welchem Namen Sie die Datei aufrufen.

Um eine Datei umzubenennen, können Sie den mv (move) Befehl benutzen. Dieser Befehl verändert den Inhalt der Datei nicht. Es wird nur der Name im Verzeichnis geändert.

Zum Löschen von Dateien gibt es das rm (remove) Kommando. Eine einmal gelöschte Datei ist verloren. Es gibt keinen Weg, wieder an die Daten gelöschter Dateien heranzukommen.[13] Deshalb ist beim Umgang mit rm immer größte Sorgfalt geboten.

Für das Löschen einer Datei ist übrigens keine Schreibberechtigung für die Datei selbst notwendig. Die Schreibberechtigung auf das Verzeichnis ist hier das entscheidende Kriterium, denn die Löschfunktion (wie auch die zum Umbenennen oder Linken) greift nicht auf die Datei zu, sondern "nur" auf den Eintrag im Verzeichnis.

### Dateien anzeigen

In der Praxis werden Sie sich häufig Textdateien ansehen wollen. Wenn Sie vorhaben, die Datei zu verändern, werden Sie einen Editor, beispielsweise den vi, aufrufen. Um die Datei einfach nur zu lesen, ist ein Editor nicht besonders geeignet. Erstens sind die Editoren zum bloßen Lesen von Textdateien einfach zu aufwendig und zu langsam und zweitens besteht immer die Gefahr der versehentlichen Veränderung einer Textdatei.

Aus diesem Grund gibt es kleine, schnelle Programme, die allein den Inhalt einer Datei anzeigen. Die wichtigsten Programme dieser Gattung sind more und less, sogenannte Pager, mit denen Sie in einer Textdatei "blättern" können. Das Vorwärtsblättern wird bei beiden Programmen durch ein Leerzeichen (Space) ausgelöst, das Zurückblättern durch den Buchstaben B (back). Das less–Programm muß ausdrücklich durch Drücken der Taste Q beendet werden, während more automatisch beendet, wenn das Dateiende erreicht wird. Eine ausführliche Beschreibung von more finden Sie auf Seite 161.

Um eine kurze Datei auf den Bildschirm zu schreiben, kann auch das cat–Kommando verwendet werden.

[13] Im neuen erweiterten Dateisystem ist das Retten versehentlich gelöschter Dateien prinzipiell vorgesehen, und die dazu notwendigen Programme werden in Kürze erhältlich sein. Allerdings sind diese Programme nur dann erfolgreich, wenn sie schon kurze Zeit nach dem Löschen eingesetzt werden.

```
$ cat Adressen
LunetIX   Lichtenrader Str. 41   12049 Berlin  030/6227300
FSF       675 Mass Ave           Cambridge,   MA 02139 USA
XeroxParc 3333 Coyote Hill Raod  Palo Alto,   CA 94304 USA
MIT       545 Technology Square  Cambridge,   MA 02139 USA
O'Reilly  632 Petaluma Avenue    Sebastopol,  CA 95472 USA
$ _
```

schreibt beispielsweise die Datei `Adressen` auf den Bildschirm. Wenn die Datei nicht auf einen Bildschirm paßt, werden die Zeilen einfach nach oben aus dem Bild geschoben, bis das Dateiende erreicht ist.

### 1.3.2 Datenströme

Das Verhalten des `cat`–Kommandos zeigt eine Eigenart der Datenverarbeitung unter Linux auf:

Dateien können als eingefrorene "Datenströme" betrachtet werden. Jeder Druck auf die Tastatur erzeugt ein Zeichen im Datenstrom. Normalerweise wird jedes eingegebene Zeichen auf dem Bildschirm dargestellt, der so ein Echo dieses Eingabedatenstroms ist.

Das `cat`–Kommando öffnet im obigen Beispiel die Datei `Adressen` und leitet die darin enthaltenen Daten als kontinuierlichen Strom auf den Bildschirm.

Passend zum Bild der Datenströme können Sie sich verschiedene Kanäle vorstellen, in denen diese Ströme wie Wasser fließen. Linux bietet sehr flexible Möglichkeiten, diese Ströme zu steuern. Damit nicht jeder Strom ausdrücklich gesteuert werden muß, gibt es einige Standardkanäle: die Standardeingabe, die Standardausgabe und die Standardfehlerausgabe.

Diese Datenkanäle sind immer offen. Normalerweise ist der Bildschirm die Standardausgabe und gleichzeitig auch die Standardfehlerausgabe (deren Sinn es ist, die normale Ausgabe eines Programms von den Fehlermeldungen besser unterscheiden zu können). Die Standardeingabe ist, wie zu erwarten war, normalerweise die Tastatur.

Die drei Standardkanäle sind numeriert: die Standardeingabe hat die Nummer Null, die Standardausgabe und die Standardfehlerausgabe haben die Nummern Eins und Zwei.

Das `cat`–Kommando schreibt normalerweise auf die Standardausgabe. Durch die Operatoren $>$ und $<$ im Kommandozeileninterpreter (der Shell) können die Datenströme aber auch umgelenkt werden. Zum Beispiel

```
$ cat Adressen > Telefonliste
$ _
```

leitet den Datenstrom, den `cat` aus der Datei `Adressen` erzeugt, in die Datei `Telefonliste` um. Die `Telefonliste` speichert den Datenstrom wieder; es entsteht also eine Kopie der Datei `Adressen`. Das gleiche Ergebnis hätten Sie natürlich auch mit dem `cp`–Kommando erzielen können.

Umgekehrt kann auch der Eingabekanal umgelenkt werden:

```
$ sort < Adressen
FSF       675 Mass Ave           Cambridge,   MA 02139 USA
LunetIX   Lichtenrader Str. 41   12049 Berlin  030/6227300
MIT       545 Technology Square  Cambridge,   MA 02139 USA
O'Reilly  632 Petaluma Avenue    Sebastopol,  CA 95472 USA
XeroxParc 3333 Coyote Hill Raod  Palo Alto,   CA 94304 USA
$ _
```

gibt beispielsweise die Adressen alphabetisch sortiert auf dem Bildschirm aus.

#### Verknüpfung mehrerer Kommandos: Pipelines

Um in der Bilderwelt von Datenströmen zu bleiben, bietet Linux zu den offenen Kanälen noch Röhren, sogenannte Pipelines. Damit wird der Ausgabekanal eines Kommandos direkt in den Eingabekanal eines anderen Kommandos geleitet.

Der Operator für diese Pipeline ist der senkrechte Strich |.

Auf der deutschen Tastatur wird er durch gemeinsames Drücken der rechten ALT mit der < Taste (links unten neben der Leertaste und der linken ALT Taste) erzeugt.

```
$ sort Adressen | uniq | less
[Anzeige von less]
$ _
```

ist so eine Pipeline aus gleich drei Kommandos. Das sort Kommando liest die Datei Adressen (die Eingabeumleitung aus dem früheren Beispiel ist nicht notwendig) und sortiert deren Inhalt zeilenweise. Die sortierten Zeilen werden von der Standardausgabe in eine Pipeline gespeist und so dem uniq–Kommando zugeführt, das die Zeilen vergleicht und alle Zeilen entfernt, die doppelt vorkommen. Die Ausgabe von uniq wird wiederum in die zweite Pipeline gespeist und schließlich dem less–Programm zugeführt, das die sortierte und von den Doppeln befreite Adreßdatei bildschirmweise anzeigt.

Mit diesen Röhren lassen sich mit den vielseitigen Kommandos des Linux–Basissystems auf einfache Weise auch komplizierte Aufgaben schnell und zweckmäßig lösen.

### 1.3.3 Eingabehilfen von der Shell

Der Kommandozeileninterpreter, die Shell, ist die Benutzeroberfläche von Linux. Es gibt auch eine ausgezeichnete grafische Benutzeroberfläche — das X Window System — aber auch hier hat die normale Shell eine herausragende Bedeutung.

Dem DOS–geschädigten Anwender mag das seltsam anachronistisch vorkommen, gibt es dort doch schon seit Jahren erfolgreiche Tools und Commanders, die die Berührung mit dem "rohen" Kommandozeileninterpreter vermeiden helfen. Da scheint Linux mit seinen Shells reichlich veraltet.

Wenn Sie aber den Umgang mit einer Shell wie der bash, der tcsh oder der zsh einmal gelernt und sich an die vielfältigen Hilfen und Möglichkeiten gewöhnt haben, werden auch Sie diese großen Shells den Kommandeuren und ihren Artgenossen vorziehen.

Zum Beispiel bieten alle modernen Shells einen Kommandozeilenspeicher. Mit den Cursortasten HOCH und RUNTER können Sie in der eigenen Computergeschichte nach Wiederverwendbarem herumstöbern. Hier stehen die letzten hundert (oder so[14]) Kommandozeilen zur Auswahl. Auch die gezielte Suche nach einer bestimmten Kommandozeile ist möglich, wenn Sie ein eindeutiges Wort aus dieser Zeile erinnern: Mit der Tastenkombination CONTROL–R[15] erscheint anstelle des bekannten Promptes die Aufforderung, den Suchbegriff einzugeben. Mit jedem eingegebenen Zeichen wird weiter rückwärts im Kommandozeilenspeicher nach einem passenden Begriff gesucht und die erste passende Zeile sofort angezeigt. Wenn die gewünschte Zeile auf diesem Weg gefunden wurde, können Sie mit den übrigen Editorfunktionen (siehe unten) die Zeile nochmal bearbeiten und schließlich mit einem Zeilenende RETURN abschließen.

Selbstverständlich kann die Kommandozeile auf vielfältige Weise bearbeitet werden. Mit den Cursortasten kann die Einfügemarke an eine beliebige Stelle gebracht werden, die BACKSPACE und DELETE Tasten verhalten sich erwartungsgemäß, und es stehen weitere Editorbefehle aus dem Befehlssatz von vi oder emacs zur Verfügung. Eine der mächtigsten Funktionen ist dabei die automatische Ergänzung einzelner Wörter durch die Shell. Mit der Tabulatortaste wird diese Funktion ausgelöst. Die Shell versucht, das bis dahin eingegebene Wort zu ergänzen, indem sie das erste Wort einer Kommandozeile mit allen möglichen Kommandos, die folgenden Wörter mit Datei– und Verzeichnisnamen vergleicht und die längste eindeutige Lösung in die Kommandozeile schreibt. Häufig reicht es aus, wenn Sie zwei oder drei Buchstaben eines Kommandos eingeben, der Rest wird von der Shell automatisch ergänzt.

Eine weitere Hilfe, die die Shell anbietet, sind die Jokerzeichen, die sogenannten "Wildcards". Beispielsweise

```
$ ls -l /bin/m*
-rwxr-xr-x   1 ruth     other        4104 Sep  4  1992 /bin/mkdir
```

[14] Der genaue Wert kann mit der Shellvariablen HISTSIZE bestimmt werden.

[15] Die CONTROL Taste wird auf der deutschen Tastatur auch oft als STRG (Steuerung) bezeichnet und befindet sich ganz links unten auf der Tastatur.

```
-rwx--x--x   1 ruth     ruth         5212 Sep  5 17:27 /bin/mkfs
-rwxr-xr-x   1 ruth     other        3336 Sep  4  1992 /bin/mknod
-rwx--x--x   1 ruth     ruth        12220 Apr 17  1993 /bin/mount
-rwxr-xr-x   1 ruth     other        6724 Sep  4  1992 /bin/mv
$ _
```

gibt ein ausführliches Listing aller Kommandos im Verzeichnis `/bin` aus, deren Name mit dem Buchstaben `m` beginnt.

### 1.3.4 Virtuelle Terminals und Hintergrundprozesse

Damit Sie die Multitaskingfähigkeit von Linux auch als einzelner Benutzer ausschöpfen können, bietet Linux verschiedene Möglichkeiten, mehrere Programme nebeneinander laufen zu lassen.

Die erste dieser Möglichkeiten wird durch die sogenannten virtuellen Terminals realisiert. Wie bereits gesagt, ist Linux ein Mehrbenutzerbetriebssystem. So kann beispielsweise ein zweiter Benutzer über eine Modemverbindung auf dem Linux–Rechner arbeiten, wenn der entsprechende serielle Port darauf vorbereitet wurde. Das Modem ist nicht notwendig, wenn ein echtes Terminal über die serielle Leitung angeschlossen wird. In vielen Fällen werden diese Varianten allerdings nicht realisiert; das Mehrbenutzersystem wird von einer einzigen Person benutzt. Um Sie trotzdem in den Genuß der Mehrbenutzerfähigkeiten kommen zu lassen, bietet Linux sogenannte virtuelle Terminals, auf denen Sie simultan eingeloggt sein können.

Die reale Console, der Bildschirm und die Tastatur am Rechner, werden zu 8 logischen Terminals, zwischen denen durch die Tastenkombination ALT-F1 bis ALT-F8 umgeschaltet werden kann. Jedes dieser virtuellen Terminals hat einen eigenen Loginprompt. Damit erhalten Sie die Möglichkeit, sich gleichzeitig mehrfach einzuloggen, eventuell sogar unter verschiedenen Benutzernamen (wenn Ihnen mehrere Accounts auf dem Rechner gehören).

Auf jedem virtuellen Terminal können Programme gestartet werden, die auch weiterlaufen, wenn auf ein anderes Terminal umgeschaltet wird. Die Tastatureingabe wird immer nur an das aktuelle Terminal weitergegeben. Die Bildschirmausgabe der einzelnen Terminals wird erst sichtbar, wenn dorthin umgeschaltet wurde. Allein die Statusmeldungen und Fehlerausgaben des Kernels werden nicht auf ein virtuelles Terminal, sondern auf die Console direkt ausgegeben und erscheinen dadurch immer auf dem Bildschirm.

Die zweite Möglichkeit, die Multitaskingfähigkeit von Linux zu nutzen, besteht darin, Prozesse auf einem Terminal in der aktuellen Shell im Hintergrund zu starten. Das geschieht, indem Sie am Ende des Kommandos ein Ampersand '&' eingeben. So ein Kommando wird ganz normal gestartet, die Shell wartet aber nicht auf dessen Beendigung, sondern gibt sofort wieder den Prompt aus und wartet auf Ihre nächste Eingabe. Je nachdem, wie das Terminal eingestellt ist (→ stty, Seite 178), erscheinen die Meldungen des Hintergrundprogramms auf dem Bildschirm, oder das Hintergrundprogramm wird für die Ausgabe angehalten, bis es wieder in den Vordergrund geholt wird. Durch Umleitung des Standardausgabekanals in eine Datei kann die Ausgabe eines Hintergrundprogramms auch für spätere Bearbeitung gespeichert werden.

```
$ sort Adressen | uniq > Adressliste &
[1] 19365
$ _
```

bearbeitet beispielsweise die Adreßdatei wie im Beispiel oben, schreibt die sortierte Liste aber in eine Datei, anstatt sie auf dem Bildschirm auszugeben. Die Zahlen werden von der Shell ausgegeben und teilen Ihnen erstens die Jobnummer und zweitens die Prozeßnummer des gerade im Hintergrund gestarteten Prozesses mit.

Es gibt noch eine dritte Möglichkeit, Prozesse in den Hintergrund zu bringen. Mit der Tastenkombination CONTROL-Z können Sie ein im Vordergrund laufendes Programm anhalten. Die Shell zeigt wie oben die Job– und die Prozeßnummer an und gibt schließlich wieder eine Eingabeaufforderung aus. Mit den sogenannten Jobcontrol Funktionen der Shell können Sie ein angehaltenes Kommando im Hintergrund oder im Vordergrund weiterlaufen lassen.

Wenn beispielsweise die laufende `sort` Pipeline mit CONTROL-Z angehalten wird, kann sie mit dem `bg`–Kommando im Hintergrund (background) fortgesetzt werden...

```
$ sort Adressen  | uniq >Adressliste

[1]+  Stopped                 sort Adressen | uniq >Adressliste
$ bg %1
[1]+ sort Adressen | uniq >Adressliste  &
$ _
```

das fg–Kommando setzt den gleichen Job im Vordergrund fort (foreground). Eine genaue Beschreibung dieser Funktionen ist ab Seite 68 zu finden.

```
$ fg %1
sort Adressen | uniq >Adressliste
$ _
```

## 1.4 Zahlensysteme

Die internen Abläufe aller Computer sind in Speichereinheiten, sogenannten Bytes organisiert. Um Daten aus der realen Welt mit einem Computer bearbeiten zu können, müssen sie im Arbeitsspeicher vorliegen. Dazu müssen sie so dargestellt werden, daß sie in die Speichereinheiten passen.

Die realen Daten werden "digitalisiert" oder "codiert".[16] Zur Codierung finden verschiedene Modelle Verwendung.

Als Formate für die digitale Darstellung von Daten 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 rechnen wie mit den "normalen" Dezimalzahlen. Dieser faszinierenden Eigenschaft und der boolschen Algebra verdanken wir erst die Möglichkeit, Computer zum Rechnen zu benutzen.

Jede natürliche Zahl läßt sich im binären Zahlensystem darstellen.

**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 Bit ist bei Computern mit acht Bits pro Byte immer null. Neben dem Alphabet werden in dieser Tabelle eine ganze Reihe von 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 wird die Menge aller 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.

[16] Bei sehr vielen Daten geht im Zuge dieser Umwandlung Information verloren. Ein Computer kennt im Prinzip nur diskrete Zustände, während die reale Welt erst durch die stetigen Veränderungen ihren Reichtum an Farben und Formen erlangen konnte. Über diese Armut des Computers können auch keine Megabytes, Megahertz oder Megaflop hinwegtäuschen.

Die Darstellung von Dezimalzahlen größer als 255 bzw. kleiner als –128 durch Bytes wird durch die Verwendung mehrerer Bytes für eine Zahl realisiert. Mit zwei Bytes können kleine natürliche Dezimalzahlen (short integer) von 0 bis 65535 bzw. ganze Zahlen von –32768 bis 32767 dargestellt werden. Größere Zahlen müssen durch entsprechend mehr Bytes codiert werden. Der GNU–C–Compiler von Linux bietet standarmäßig vier Byte große *long integer* an.

Als annäherndes Modell für rationale oder reelle Zahlen benutzen Computer Fließkommazahlen. Eine Fließkommazahl mit einfacher Genauigkeit (float) wird intern durch vier Bytes (=32 Bits) dargestellt, von denen ein Bit das Vorzeichen, 8 Bit den Exponenten mit eigenem Vorzeichen und die restlichen 23 Bit die Mantisse darstellen. Fließkommazahlen mit doppelter Genauigkeit (double) werden durch (unter Linux normalerweise) acht Bytes mit 11 Bit Exponenten und 52 Bit Mantisse dargestellt. Transzendente Zahlen können vom Computer nicht dargestellt und deshalb auch nicht bearbeitet werden. Rationale Zahlen, die außerhalb des hier dargestellten Bereichs liegen, können unter Linux nicht mit Standardfunktionen berechnet werden.

**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 identisch mit der Dezimalzahl 8.

Der Vorteil des oktalen Zahlensystems besteht in der leichten Umrechenbarkeit von Oktalzahlen in Binärzahlen und umgekehrt. Weil die Oktalzahlen "handlicher" sind als ihre binären Äquivalente, werden sie gern zur Darstellung von Bytes verwendet.

| | | |
|---|---|---|
| 0 | = | 000 |
| 1 | = | 001 |
| 2 | = | 010 |
| 3 | = | 011 |
| 4 | = | 101 |
| 5 | = | 101 |
| 6 | = | 110 |
| 7 | = | 111 |

Jede Oktalziffer kann als dreistellige Binärzahl dargestellt werden. 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 in handliche Stücke zerlegt werden.

Ein Byte besteht aus acht Bits. Zur Darstellung als Oktalzahl wird ein neuntes Bit gedacht, das immer Null ist.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| | 3 | | | 6 | | | 5 | |

**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.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| | F | | | | 5 | | |

Im Hexadezimalsystem entspricht ein Byte aus acht Bits genau einer zweistelligen Hexadezimalzahl.

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.

# Kapitel 2

# Reise durch's Dateisystem

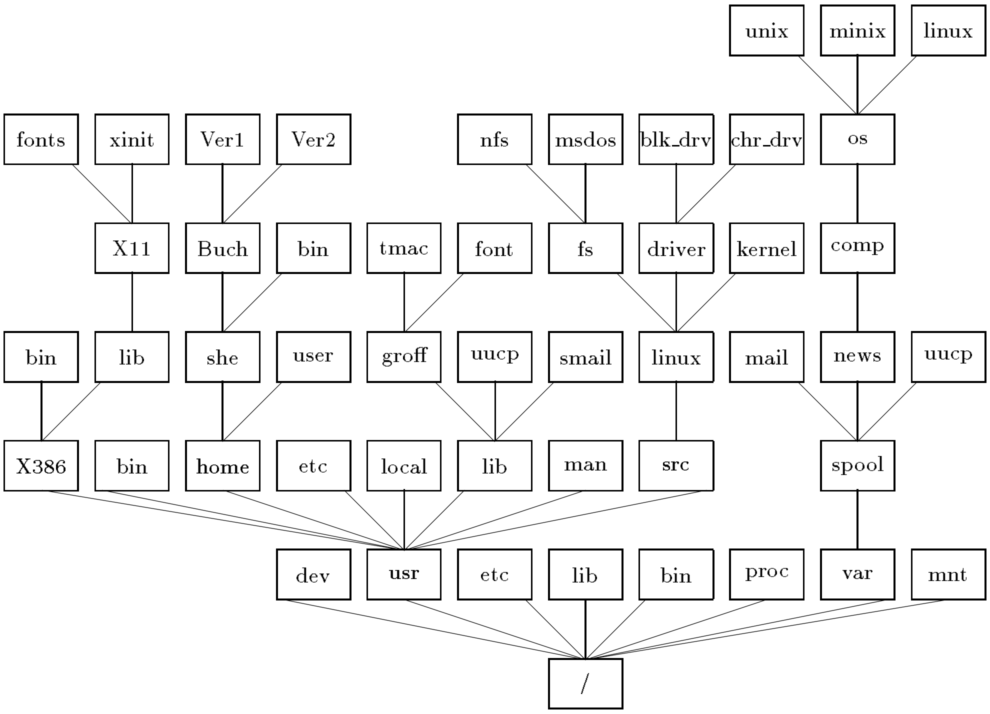


Abbildung 2.1: Der Verzeichnisbaum (unvollständig)

Die meisten Linux–Distributionen haben einen Umfang von 30 bis 40 Disketten. Ausgepackt belegen diese Pakete über 100 Megabyte auf der Festplatte. Um mit so einer Menge Daten überhaupt arbeiten zu können, müssen Sie sich einen groben Überblick über die Ordnung und die Funktion der Dateien verschaffen. Zu diesem Zweck wird in diesem Kapitel eine Rundreise durch das Dateisystem veranstaltet, auf der Ihnen die wichtigsten Verzeichnisse und Dateien vorgestellt werden.

Die Struktur des Linux–Dateisystems orientiert sich grundsätzlich an Unix. Es gibt jedoch auch bei den verschiedenen Unix–Versionen unterschiedliche Dateisystem–Strukturen, die bei den frühen Linux–Distributionen ohne klares Konzept zusammengewürfelt worden sind. Deshalb ist es zur Zeit nicht möglich ein Dateisystem–Layout für alle Distributionen zu beschreiben.

Es ist aber ein gut durchdachter Entwurf (Draft) für ein Standard–Dateisystem ausgearbeitet und ausführlich diskutiert worden, der sich stark an der Struktur von System V.4 orientiert. Dieser Entwurf ist nicht eine bloße Adaption eines bestehenden Layouts, sondern ihm liegen einige als wichtig erkannte Prinzipien zu Grunde. Durch diese Verbindung eines klaren Konzepts mit einem existierenden, modernen Modell–Dateisystem wird es (hoffentlich) in der Zukunft ein von Geschmack und Glaubensfragen unabhängiges Linux–Dateisystem geben.

Die Realisierung des neuen Modells in den Linux–Distributionen hat bereits begonnen. Wegen der großen Zahl der von dieser Umstellung betroffenen Programme und Pakete wird bis zur vollständigen Umsetzung einige Zeit vergehen.

Wenn Sie eine Linux–Distribution installiert haben, die dem Entwurf für ein Standard–Dateisystem (noch) nicht folgt, werden Sie einige der hier vorgestellten Dateien an anderen Orten und einige Verzeichnisse gar nicht finden. Trotzdem werden Sie viele Übereinstimmungen feststellen. Die Funktion und das Format der Dateien ändert sich natürlich mit dem bloßen Layout des Dateisystems nicht.

Die Beschreibung der einzelnen Dateien können Sie beim ersten Lesen des Kapitels überspringen. Wenn Sie ihr eigenes Linux–System eingerichtet haben und verwalten müssen, sollten Sie nach und nach alle Verzeichnisse und Dateien ihres Systems genauer studieren. In diesem Kapitel finden Sie notwendige Informationen und nützliche Hinweise, die Sie in die Lage versetzen werden, das Linux–System Ihren speziellen Bedürfnissen und Gegebenheiten anzupassen.

## 2.1 Konzepte

Vor Antritt der Reise sollten Sie einige grundlegende Konzepte des Linux–Dateisystems verstehen.

Wahrscheinlich kennen Sie vom MS–DOS Festplattenpartitionen als "Laufwerke" `C:\` oder `D:\`. Hier müssen Sie als Benutzer das "richtige" Laufwerk wählen, wenn Sie ein bestimmtes Verzeichnis oder eine bestimmte Datei erreichen wollen.

Wenn Sie die Linux–Distribution selbst installiert haben, werden Sie gesehen haben, daß auch Linux mehrere Festplattenpartitionen benutzen kann. Bei einem fertig installierten Linux–System brauchen Sie sich aber um solche Hardware–Angelegenheiten nicht mehr zu kümmern: Es ist eines der wichtigsten Konzepte des Linux–Dateisystems, die hardwareabhängigen Aspekte Ihres Arbeitssystems vollständig zu verbergen.

Auf der Benutzerebene bewegen Sie sich in einem einzigen, hierarchischen Dateisystem, in das alle Datenträger integriert sind. Es ist wie ein Baum aufgebaut, dessen Äste die Verzeichnisse und dessen Blätter die Dateien sind. Eine bestimmte Festplattenpartition — die Root–Partition — wird automatisch vom Kernel als "Root–Filesystem" aktiviert; andere Partitionen werden bei der Systeminitialisierung auf bestimmte Verzeichnisse in dieser Root–Partition aufgesetzt. Für Sie als Benutzer sind die "Nahtstellen" zwischen den Partitionen nur durch spezielle Hilfsprogramme sichtbar, aber für Ihre Arbeit mit dem System sind sie ohne Bedeutung.

Die Funktion von Verzeichnissen oder *Directorys* wird Ihnen wahrscheinlich bekannt sein: Sie können sie sich als eine Art Behälter vorstellen, in denen Dateien und/oder weitere Verzeichnisse enthalten sein können.

Dateien (*Files*) sind auf den ersten Blick Texte, Programme oder sonstige Daten, die in ihrer maschinenlesbaren Form als "Strom" von Bytes in Blöcken auf einem Massenspeicher (Festplatte oder Diskette) festgehalten und von dort wieder gelesen werden können. Dateien werden bei der Erzeugung mit einem Namen in ein Verzeichnis eingetragen und können danach unter diesem Namen angesprochen werden.

Darin unterscheidet sich das Linux–Dateisystem noch nicht von dem anderer Betriebssysteme. Erst bei näherer Betrachtung ändert sich das Bild: Bei Linux können oder müssen grob sechs Arten von Dateien unterschieden werden:

| | |
|---|---|
| Normale Dateien | (*regular files*) entsprechen dem oben dargestellten Bild. Sie können ebensogut lesbare Gedichte wie ausführbare Maschinenprogramme enthalten. Sie belegen normale Datenblöcke auf der Festplatte. |
| Verzeichnisse | (*directorys*) sind, genauer betrachtet, spezielle Dateien, in denen auf eine spezielle Art weitere Dateien "enthalten" sind. Auch sie belegen Datenblöcke auf der Festplatte. Der Aufbau dieser Spezialdateien wird im Kapitel über die Dateisysteme auf Seite 290 erläutert. |
| Gerätedateien | sind Bindeglieder zwischen den Hardware–Komponenten und Geräten (*devices*) am Rechner bzw. den Gerätetreibern im Kernel auf der einen Seite und der Software im Laufzeitsystem, also den Anwenderprogrammen, auf der anderen. Alle Benutzerprozesse, die auf ein angeschlossenes Gerät oder eine Hardwarekomponente zugreifen wollen, müssen das über eine solche Gerätedatei tun. |
| Sockets | (UNIX domain *sockets*) sind Spezialdateien aus dem Bereich der TCP/IP Vernetzung, mit denen der Datenaustausch zwischen zwei lokal laufenden Prozessen über das Dateisystem realisiert werden kann. |
| FIFOs | (named *pipes*) stellen eine zweite, einfachere Methode des Datentransports zwischen zwei Prozessen über das Dateisystem dar. Wie die Pipelines in der Shell können die FIFOs Daten nur in einer Richtung transportieren. |
| Links | sind zusätzliche Namen (Verzeichniseinträge) für existierende Dateien. Es werden symbolische Links und Hardlinks unterschieden. Während Hardlinks vollkommen gleichwertige Verzeichniseinträge für eine existierende Datei sind, bestehen symbolische Links aus einer Spezialdatei, deren Inhalt ein Zeiger auf eine andere Datei ist. |

### 2.1.1 Der File–System–Standard (Entwurf)

Der File–System–Standard ist das Projekt einer Gruppe von Linux–Systemadministratoren, –Entwicklern, und –Benutzern, die ihre Erfahrungen und Vorstellungen auf dem FSSTND–Channel[1] austauschen und diskutieren. Das vorläufige Ergebnis ist von Daniel Quinlan (quinlan@bucknell.edu) in einem Entwurf zusammengefaßt worden.[2]

Mehrere zentrale Prinzipien ziehen sich durch diesen Entwurf:

- Um der zunehmenden Bedeutung von Linux als verteiltes Betriebssystem auf lokalen TCP/IP Netzwerken Rechnung zu tragen, muß ein maximaler Anteil des Dateisystems für die gemeinsame Nutzung (über

[1] Als "channel" werden die offenen Mailing–Listen von linux-activists@Niksula.hut.fi bezeichnet, auf denen die Aktivisten der Linux–Gemeinde ihre Projekte diskutieren. Wenn Sie sich aktiv an der Entwicklung von Linux beteiligen wollen und die Möglichkeit haben, internationale Internet–Mail zu senden und zu empfangen, können Sie sich in jeden dieser Channels einschreiben. Informationen zur Benutzung der Channels erhalten Sie, indem Sie eine leere Mail an linux-activists-request@niksula.hut.fi schicken (Subject ohne Bedeutung)

[2] Dieses Dokument können Sie auf verschiedenen FTP–Servern finden, beispielsweise auf tsx-11.mit.edu:/pub/linux/docs/linux-standards/private/fsstnd.

NFS) vorbereitet sein. Eine Voraussetzung dafür ist, daß dieser Teil des Dateisystems beim normalen Betrieb nicht verändert wird.

Dieses Prinzip führt zu einer Unterscheidung der Daten in zwei Richtungen:

- verteilte Daten/lokale Daten
- variable Daten/statische Daten

- Die Unterteilung des Dateisystems sollte nach den bei Unix üblichen funktionalen Kategorien stattfinden. Das bedeutet, daß jeweils Binärdateien, Konfigurationsdateien, Hilfs–Texte und sonstige Daten in verschiedenen Verzeichnissen untergebracht werden.

- Die beiden Prinzipien oben führen insbesondere zur Forderung nach einer klaren und übersichtlichen Strukturierung der Konfigurationsdateien. In dieser Struktur muß sich auch die Unterscheidung zwischen netzweit verteilten und (rechner–) lokalen Daten sowie die Unterscheidung zwischen der von einer Distribution eingespielten und der zusätzlich (netz–) lokal hinzugefügten Software wiederspiegeln.

An dieser Stelle beginnt die eigentliche Reise durch das Dateiesystem. Auch wenn Ihnen an vielen Stellen ein Blick hinter die Kulissen gewährt wird, sehen Sie auf der Oberfläche ein leicht idealisiertes Dateisystem. Damit wird Ihnen nichts wichtiges vorenthalten, sondern im Gegenteil Ihr Blick auf das Wesentliche gerichtet.

## 2.2 Root–Partition und Wurzelverzeichnis

Wie bereits gesagt, bietet das Linux–Dateisystem das Bild eines Baumes mit einer Wurzel. Es ist naheliegend, eine Reise durch das Dateisystem an der Wurzel anzufangen.
Ein kurzes Listing zeigt uns das folgende Bild:

```
$ ls -F /
bin/         etc/         lost+found/  ruth/        usr/
boot/        home/        mnt/         sbin/        var/
dev/         lib/         proc/        tmp/         vmlinuz
$ _
```

Wie bereits oben erwähnt wurde, wird das Root–Filesystem vom Kernel automatisch aktiviert sobald er in den Arbeitsspeicher geladen und initialisiert ist. Das Wurzelverzeichnis ist von da an fest im Kernel verankert.[3]

Der Linux–Kernel erfüllt selbst nur die minimalen Aufgaben eines Betriebssystems. Damit der Kernel auch bestimmte Benutzerprogramme (wie beispielsweise init, getty oder eine Shell) ausführen kann, ist diese feste Verbindung notwendig. Das bedeutet, daß alle Programme, die vom Kernel unmittelbar beim Systemstart automatisch ausgeführt werden sollen, zusammen mit ihren Konfigurationsdateien im Root–Filesystem angesiedelt sein müssen.

Zusätzlich müssen allein mit dem Root–Filesystem eine Reihe von essentiellen Arbeiten der Systemverwaltung ausführbar sein. Dazu gehören

- das Erzeugen, Zusammensetzen und Reparieren von Dateisystemen,
- das Sichern und Zurücksichern der Systemdaten sowie die Installation neuer Systemteile und
- in Netzwerksystemen die Herstellung und die Prüfung einer Netzverbindung im lokalen Netz.

[3] Mit den Linux–Kernels ab Version 0.99.10 kann das Root–Filesystem re– und unmounted werden. Dadurch ist es möglich, das Root–Filesystem zuerst "Read–Only" zu mounten, um sicher einen File–System–Check durchführen zu können. Bevor die übrigen Partitionen aufgesetzt werden, wird durch ein remount–Kommando das Lesen und Schreiben auf das Root–Filesystem erlaubt. Beim Herunterfahren des Systems wird das Root–Filesystem vom Kernel getrennt und beim ext2–Filesystem das Valid–Flag in der I–Node des Wurzelverzeichnisses gelöscht.

Die Größe des Root–Filesystems bzw. die Abgrenzung zum "Benutzersystem" ist eine der Glaubensfragen beim Design des Dateisystems. Der File–System–Standard sieht die Beschränkung des Root–Filesystems auf das absolut notwendigste vor. Das hat vor allem den Vorteil, daß das potentielle Risiko eines Datenverlustes durch Fehler im Dateisystem bei einer kleinen Partition, auf der im wesentlichen nur statische Daten gespeichert sind sehr klein ist. So eine Partition läßt sich auch unter ungünstigen Bedingungen regelmäßig (zum Beispiel auf einer Floppy–Disk) sichern, so daß die Wiederherstellung des Systems auch nach einem Totalausfall kein ernstes Problem darstellt. Im Idealfall kann eine so angelegte Partition sogar auf einer Bootdiskette Platz finden, so daß das System ohne Festplatte gebootet werden kann (um danach über das lokale Netz zu arbeiten).

Wenn Sie sich das Listing des Wurzelverzeichnisses anschauen, werden Sie eine kleine Anzahl Verzeichnisse und eine einzige normale Datei finden. Die Datei kann in Abweichung vom oben gegebenen Beispiel auch `zImage` heißen und enthält den bootfähigen Kernel.

### Das Verzeichnis `/bin`

Die für Unix typische Strukturierung des Dateisystems nach funktionalen Gesichtspunkten wird bei den Verzeichnissen mit dem Namen `bin` besonders deutlich. Auf den verschiedenen Ebenen des Dateisystems gibt es jeweils ein Verzeichnis dieses Namens, denen allen gemeinsam ist, daß sie ausschließlich ausführbare Dateien enthalten.

Der Vorteil dieser Zusammenfassung besteht darin, daß die automatische Suche der Shell nach einem ausführbaren Programm auf diese Weise schnell und unkompliziert ist. Zu diesem Zweck werden der Shell alle `bin` Verzeichnisse mit ihrem Pfadnamen im Dateisystem in der PATH Umgebungsvariablen mitgeteilt.

Das `/bin` Verzeichnis im Root–Filesystem enthält alle Programmdateien, die für die essentiellen Aufgaben der Systemverwaltung gebraucht werden, die aber auch von den anderen Systembenutzern verwendet werden können.

Im File–System–Standard ist eine Liste der essentiell wichtigen Programmdateien für `/bin` aufgeführt.

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

Mit Ausnahme der Kommandos `netstat` und `ping`, die zum TCP/IP Networking gehören, sind alle Kommandos in diesem Buch ausführlich beschrieben.

### Das Verzeichnis `/boot`

Das neue Verzeichnis `/boot` enthält die Dateien des LILO Bootloaders, die nicht ausführbar sind und auch keine Konfigurgationsdateien sind. Das sind in der Regel der gesicherte Master–Boot–Record und die Sector–Map. Hier können auch zusätzliche Kernel–Images abgelegt werden. Nach dem File–System–Standard Entwurf kann und soll allein der Default–Kernel im Wurzelverzeichnis liegen.

Eine Erklärung des LILO Bootloaders finden Sie ab Seite 216.

## 2.3 Das Verzeichnis `/dev`

Eines der Charakteristika von Linux ist die vollständige Abschirmung der Hardware- von der Benutzerebene. Wann immer ein Anwenderprogramm Daten an ein angeschlossenes Gerät oder eine Hardwarekomponente (Device) schicken oder von dort lesen will, muß es eine Spezialdatei öffnen und die Daten in diese Datei schreiben oder aus dieser Datei lesen.

Die Systemverwalterin hat die gleichen Möglichkeiten, Zugriffsrechte auf die Gerätedateien zu erteilen oder zu verweigern, wie bei allen anderen Dateien auch. Außerdem kann der Gerätetreiber im Kernel konkurrierende Zugriffe mehrerer Programme auf ein Gerät verhindern.

Ein weiterer entscheidender Vorteil dieses auf den ersten Blick vielleicht etwas restriktiv anmutenden Mechanismus ist die Fähigkeit des Kernels, Prozesse bis zum Eintreten eines Hardwareereignisses in Schlaf zu versetzen. Auf diese Weise kann die Rechenzeit, die sonst in einer Warteschleife für das sogenannte *polling* verschwendet würde, sinnvoll für andere Prozesse verwendet werden.

In dem Verzeichnis `/dev` sind die "Bilder" der Hardwarekomponenten und Geräte des Systems abgelegt. Die Bilder sind keine Dateien im herkömmlichen Sinne. Sie belegen keine Datenblöcke auf der Festplatte, sondern nur I–Nodes. Diese Spezialdateien stellen eine Verbindung zwischen dem Anwenderprogramm und dem Gerätetreiber im Kernel her. In manche diese Dateien kann geschrieben oder aus ihnen gelesen werden, wie bei normalen Dateien. Der Datenstrom wird vom Kernel übernommen und an das entprechende Gerät weitergeleitet.

Die Verbindung zum Kernel wird über Slots oder Kanäle hergestellt, die numeriert 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 (gepufferten) Geräte, wie z. B. Disketten oder Festplatten, und die zeichenorientierten (ungepufferten) Geräte, wie Drucker, Terminal oder Maus. Damit hat jede Gerätedatei drei "Koordinaten", mit der sie vom Kernel unabhängig von ihrem Namen eindeutig identifiziert werden kann.

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 Liste der vom Linux–Kernel unterstützten Geräte wächst mit der Beliebtheit von Linux ständig an. Um trotz aller Freiheit ein Mindestmaß an Koordination bei der Belegung von Devicenummern zu bewahren, führt Rick Miller (rick@ee.uwm.edu), der "Linux Device Registrar", die offizielle Deviceliste und veröffentlicht sie gelegentlich im USENET. Seit der Linux–Version 0.99.14 gibt es außerdem die Include–Datei < `linux/major.h` >, in der die Major–Device–Numbers aller vom Standard–Kernel unterstützten Geräte eingetragen sind.

Gerätedateien können von der Superuserin mit dem mknod Kommando erzeugt werden. Bei der Installation einer neuen Linux–Distribution wird durch das Shellscript `MAKEDEV` automatisch ein Satz Gerätedateien erzeugt.

Die Namensgebung für die Gerätedateien ist Konvention. Jede Datei kann beliebig umbenannt werden oder durch Links mit anderen Namen verbunden werden, die Benutzung des Gerätes ändert sich dadurch nicht. Der File–System–Standard sieht aber vor, daß bei der Installation einer Linux–Distribution keine symbolischen Links automatisch erzeugt werden sollen.

Bei der Beschreibung der Gerätedateien werden gleichartige Geräte durch Wildcards bezeichnet, wie sie in der Shell verwendet werden. Ein Fragezeichen steht hier für ein einziges Zeichen, ein Asterisk (Stern) steht für mehrere Zeichen. Sie können sich die vollständige Liste der angesprochenen Gerätedateien mit dem ls Kommando anzeigen lassen, indem Sie genau den in der Beschreibung angegebenen Ausdruck als Argument verwenden. Beispielsweise die Bezeichnungen `/dev/fd?` und `/dev/hd*` erweitern sich so:

```
$ ls /dev/fd?
/dev/fd0  /dev/fd1
$ ls /dev/hd*
/dev/hda   /dev/hda3  /dev/hda6  /dev/hdb   /dev/hdb3  /dev/hdb6
/dev/hda1  /dev/hda4  /dev/hda7  /dev/hdb1  /dev/hdb4  /dev/hdb7
/dev/hda2  /dev/hda5  /dev/hda8  /dev/hdb2  /dev/hdb5  /dev/hdb8
$ _
```

Für den Anwqender sind speziell die Dateien `/dev/fd?`, `/dev/lp?`, `/dev/null`, `/dev/zero`, `/dev/ttyS?` und `/dev/cua?` von Interesse. Die anderen Dateien werden hauptsächlich von der Systemverwalterin benötigt.

### 2.3.1 Der Arbeitsspeicher

#### /dev/core

Die Datei /dev/core stellt den Kernelspeicher (/dev/kmem) im Core–File–Format dar. Dieses Format kann vom GNU–Debugger gdb bearbeitet werden.

#### /dev/mem und /dev/kmem

Die Dateien /dev/mem und /dev/kmem bilden den Arbeitsspeicher des Rechners ab. 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/ramdisk

Die Datei /dev/ramdisk bildet die RAM–Disk in das Dateisystem ab, wenn beim Übersetzen des Kernels bzw. mit dem rdev–Kommando oder auf der LILO Kommandozeile ein Teil des Speichers als solche bestimmt wurde. Die RAM–Disk ist ein rohes Blockdevice und kann benutzt werden wie eine Diskette.

Wenn die RAM–Disk in das Dateisystem eingebunden werden soll, muß sie ein Dateisystem enthalten. Dieses Dateisystem kann von der Bootdiskette[4] geladen werden oder mit dem mkfs–Kommando eingerichtet werden. Wenn die RAM–Disk von der Bootdiskette geladen wurde, wird sie als Rootfilesystem benutzt, auf das alle weiteren Dateisysteme aufgesetzt werden. Sonst kann sie wie eine Diskette mit dem mount–Kommando auf ein Verzeichnis aufgesetzt werden, beispielsweise auf /tmp.

#### /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.

### 2.3.2 Runde Scheiben

#### /dev/fd*

Die Dateien /dev/fd* repräsentieren die Diskettenlaufwerke (floppy disks). Weil der Kernel das Diskettenformat automatisch erkennen kann, sind für die meisten direkten Diskettenzugriffe die beiden Dateien /dev/fd0 und /dev/fd1[5] ausreichend. Dennoch werden für alle gängigen Diskettenformate spezielle Einträge mit entsprechenden Minor Device Numbers angeboten. Auf diese Weise kann der Treiber gezwungen werden, das Diskettenlaufwerk mit bestimmten Parametern anzusprechen, was zum Beispiel zum Formatieren von Disketten notwendig ist.

[4] Um die RAM–Disk so zu initialisieren, muß auf der Bootdiskette an einer bestimmten Stelle ein gültiges Dateisystem beginnen. Bei früheren Bootdisketten mit "rohen" Kernelimages mußte das der Block 512 sein. Bei neuen Bootdisketten, die mit LILO gestartet werden und das Kernelimage im Dateisystem haben, wird das komplette Dateisystem ab Block 0 geladen. Wenn kein passendes Dateisystem zur Initialisierung auf der Bootfloppy gefunden wurde, bleibt die RAM–Disk uninitialisiert.

[5] Wenn Sie von der Möglichkeit Gebrauch machen, einen zweiten Floppy–Controller zu betreiben, werden natürlich auch die Diskettenformate in /dev/fd2 und /dev/fd3 entsprechend automatisch erkannt.

Das System der Namensgebung für die Dateien ist willkürlich, aber leicht zu verstehen. Nach den Buchstaben '`fd`' für Floppy Disk kommt die Laufwerksnummer (`0`, `1`, `2` oder `3`)) und dann folgt einer der Buchstaben `D` oder `H`. `D` steht für Double–Density, was in diesem Fall nichts mit dem Medium, sondern mit dem Laufwerk zu tun hat: Es handelt sich hier um ein Laufwerk das höchstens 40 Sektoren pro Spur verarbeiten kann. Die heute in der Regel verwendeten Laufwerke können 80 Sektoren pro Spur verarbeiten und werden als High–Density Laufwerke angesprochen; der Kennbuchstabe ist also ein `H`. Diese Buchstaben können noch groß oder klein geschrieben sein. Ein großer Buchstabe steht 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 137 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, so daß der Anwender zur Laufzeit des Programms die Privilegien des Dateieigentümers oder der Gruppe hat.

Mit dem sogenannten user–mount–Kommando kann die Systemverwalterin jedem Systembenutzer das Einbinden eigener Disketten in das allgemeine Dateisystem erlauben. Mit dem normalen mount–Kommando ist es aus Gründen der Systemsicherheit nur der Superuserin (ruth) erlaubt, das Dateisystem zu verändern (→ Seite 209).

Eine weitere Möglichkeit zur Sicherung persönlicher Daten auf Disketten bietet das mcopy–Programm aus dem mtools–Paket (→ Seite 162).

### `/dev/hd*`

Die Dateien `/dev/hd*` repräsentieren die "normalen" (IDE, RLL, MFM) Festplatten. Dabei stehen `/dev/hda` und `dev/hdb`[6] für die gesamte (rohe) Festplatte und die durchlaufend numerierten Dateien stehen für die einzelnen Partitionen. Dabei bezeichnen die Nummern 1 bis 4 die primären Partitionen, die Nummern 5 bis 8 die erweiterten Partitionen.

Im Gegensatz zu den Diskettenlaufwerken sind die Festplatten nur für die Superuserin (ruth) direkt ansprechbar. Alle anderen Anwender sind an die normalen Zugänge über das Dateisystem gebunden.

Besondere Vorsicht ist beim Umgang mit den rohen Geräten `/dev/hda`, `dev/hdb` usw. geboten. Diese Dateien sollten nur mit fdisk oder anderen speziell dafür vorgesehenen Programmen bearbeitet werden. Eine versehentliche Änderung des ersten Datenblockes dieser Datei vernichtet die Partitionstabelle und macht alle Daten auf der Festplatte bis auf weiteres unbrauchbar.

### `/dev/mcd0`

Die Datei `/dev/mcd0` steht für ein Mitsumi CMRC–lu CD–ROM Laufwerk. [7]

### `/dev/sd*`

Die Dateien `/dev/sd*` bilden die SCSI Festplatten ab. Die Major Device Nummer für die SCSI Festplatten 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/sonycd`

Diese Gerätedatei repräsentiert das Sony CDU–535 CD–ROM Laufwerk.

[6] Auch hier können zwei weitere Festplatten `/dev/hdc` und `/dev/hdd` benutzt werden, wenn ein zweiter Festplatten–Controller verwendet wird.

[7] Damit der Treiber bei den unspezifischen Installationsdiketten initialisiert werden kann, muß dem Kernel die IRQ und die IO–Adresse in der Kommandozeile übergeben werden. Eine Kommandozeile kann beim Booten mit LILO nach dem Bootprompt eingegeben werden, der nach dem Drücken der ALT Taste beim Booten erscheint.

`/dev/sr?`

Bei diesen Dateien handelt es sich um die Gerätedateien für die SCSI CD–ROMs.

`/dev/xd*`

Der Linux–Kernel unterstützt auch 8–Bit XT–Festplatten–Controller. Die Festplatten und Partitionen sind nach dem gleichen Schema benannt wie die AT–Bus Platten.

### 2.3.3 Peripherie

**Busmäuse**

Linux unterstützt die Busmäuse von Logitech und die PS/2 Busmaus, die MicroSoft "InPort" und ATI XL Busmaus. Die dazugehörigen Gerätedateien heißen `/dev/logibm`, `/dev/psaux`, `/dev/inportbm` und `/dev/atibm`.

`/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 nocheinmal speziell für ausgehende Modemverbindungen (dial out) zur Verfügung.

Im Unterschied zu den `ttyS?` Devices werden die `cua?` Dateien automatisch "non–blocking", also unabhängig vom Carrier–Detect, geöffnet. Die `cua?` und die entsprechenden `ttyS?` Devices können nicht gleichzeitig geöffnet werden.

Wenn beispielsweise `/dev/ttyS1` durch einen ankommenden Anruf geöffnet ist, wird jeder Versuch, das Device `/dev/cua1` zu öffnen, mit der Fehlermeldung 'EBUSY' abgewiesen. Andererseits kann das `/dev/ttyS1` Device nicht geöffnet werden, solange `/dev/cua1` in Benutzung ist.

`/dev/lp?`

Die Gerätedateien `/dev/lp?` bilden die parallelen Druckerschnittstellen im Dateisystem ab. 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; falls der Drucker dazu bereit ist und kein anderer Prozeß den Drucker belegt. Wenn ein Druckerdämon installiert ist, kann der direkte Zugriff auf den Drucker für die Systembenutzer verboten sein.

Wenn der Drucker aus irgendwelchen Gründen nicht bereit ist, Daten anzunehmen, gibt der Kernel automatisch eine Fehlermeldung aus. Ist das Gerät beispielsweise durch einen anderen Job belegt, wird der Anwender darauf hingewiesen.

Normalerweise wird die parallele Schnittstelle im *polling*–Mode betrieben. Das bedeutet, beim Senden von Daten an den Drucker wird in einer Warteschleife der Status des Druckers dauernd abgefragt. Nur im Fall der Bereitschaft werden Zeichen an den Drucker geschickt.

Um den mit dieser Betriebsart verbundenen Rechenaufwand zu vermindern, kann der Drucker auch interruptgesteuert betrieben werden. Zu diesem Zweck muß der Druckertreiber im Kernel mit dem lpcntl–Programm initialisiert werden. Die Druckerschnittstelle `/dev/lp1` kann beispielsweise in der `rc.local` Datei mit der folgenden Zeile auf Interruptsteuerung umgestellt werden:

```
# Interruptsteuerung für LPR1 IRQ 7 (Standard)
lpcntl /dev/lp1 7
```

#### `/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 (Drucker, ASCII–Terminals, Maus oder Modem) in das Dateisystem ab.

Seit der Kernelversion 0.99.5 sind alle seriellen Schnittstellen zweimal im Dateisystem abgebildet, einmal als `/dev/ttyS?` und ein zweites als `/dev/cua?`. Im Unterschied zu den `cua?` Devices **blockieren** die `ttyS?` Gerätedateien normalerweise jeden Versuch, das Device zu öffnen, solange, bis die entsprechende /dev/cua? Datei geschlossen ist und ein Carrier–Detekt von der serieellen Schnittstelle gemeldet wird.

Ein auf der Modemleitung `/dev/ttyS1` wartendes getty wird beispielsweise automatisch solange blockiert, bis das Modem die Carrier–Detect Leitung "hoch" setzt. Wenn in diesem Moment die Datei `/dev/cua1` geöffnet ist (weil gerade ein ausgehender UUCP–Call das Modem belegt, der Carrier also zu dieser Verbindung gehört), wird das getty weiter blockiert. Erst wenn ein auf "Auto–Answer" gesetztes Modem eine ankommende Modemverbindung registriert, wird die von getty angeforderte Gerätedatei `/dev/ttyS1` geöffnet und dadurch das login auf diesem Port eingeleitet.

Die Blockierung des open(2) Systemaufrufs für die Devices `/dev/ttyS?` kann durch den Modus 'O_NON-BLOCK' ('O_NDELAY') umgangen werden. Das so geöffnete Device verhält sich dann wie das entsprechende `/dev/cua?`.

#### `/dev/audio /dev/dsp /dev/midi /dev/mixer /dev/sequencer`

Diese Gerätedateien sind für Soundkarten und ähnliche Audio–Geräte.

### 2.3.4 Daten am laufenden Band

#### `/dev/rmt?` und `/dev/tape*`

Linux unterstützt im Standard–Kernel neben SCSI Streamer auch QIC–02 Bandlaufwerke mit eigener Adapterkarte (beispielsweise Archive SC400/SC402/SC499, Everex 811V/831V oder Wangtek 5150).

Die Dateien `/dev/rmt?` bilden die Streamer im Dateisystem ab.

In jedem Fall handelt es sich bei den Streamern um ungepufferte, zeichenorientierte Geräte. Die QIC–02 Streamer benutzen die Major Device Number 12, durch die Minor Device Number wird der Treiber für eine bestimmte Bandsorte eingestellt:

**0** automatische Erkennung

**2** QIC-11 (24 MB, 4 Spuren, 10 000 ftpi)

**4** QIC-24 (60 MB, 9 Spuren, 10 000 ftpi)

**6** QIC-120 (120 MB, 15 Spuren, 12 500 ftpi)

**8** QIC-150 (150 MB, 18 Spuren, 12 500 ftpi)

Einträge für QIC-300 und QIC-600 (Minor 10 und 12) sind im Source–Code für den Treiber enthalten, aber als ungetestet kommentiert.

Die hier aufgezählten Nummern stehen für "no rewind on close"; wenn das niedrigste Bit 1 (die Gerätenummern also ungerade) ist, arbeitet der Streamer "rewind on close". Die Gerätedateien mit ungeraden Minor Device Nummern werden vom `MAKEDEV` Script nicht automatisch erzeugt.

Wenn das 7. Bit der Minor Device Number gesetzt ist, gibt der Gerätertreiber zusätzliche Debugging–Information aus. Die vom `MAKEDEV` Script der aktuellen Distributionen erzeugte Datei `/dev/tape-d` ist ein Beispiel für diesen Fall. Ein Öffnen der Gerätedatei `/dev/tape-reset` (Minor 255) bewirkt einen Reset des Tape–Adapters und erforderlichenfalls ein Zurückspulen des Bandes.

#### `/dev/st?` und `/dev/nst?`

Die Dateien `/dev/st?` und `/dev/nst?` stellen die SCSI Streamer dar (zeichenorientiert!).

Die `nst?` unterscheiden sich von den `st?` Devices dadurch, daß die `n*` Versionen beim schließen der Datei das Band nicht zurückspulen (no rewind on close). Die Minor Gerätenummern sind für die SCSI Streamer Null für "rewind on close" und 128 für "no rewind on close". Diese einfache Numerierung ist möglich, weil die SCSI Streamer die Bandsorte selbst erkennen.

Je nach Herkunft des tar oder cpio Programms, kann es sinnvoll sein, die SCSI–Streamer–Devices mit Links von den rmt? und nrmt? Dateien zu versehen, damit die Programme ohne ausdrückliche Angabe einer Datei oder eines Devices automatisch auf das korrekte Bandlaufwerk zugreifen.

#### `/dev/ftape`

Die Gerätedateien `/dev/ftape` und `/dev/nftape` repräsentieren die Floppy–Streamer. Die Entwicklung von Treibern für Floppystreamer hat große Fortschritte gemacht, so daß mit zuverlässigen Implementationen in nächster Zukunft zu rechnen ist.

Die Gerätetreiber für Floppy–Tapes werden nicht fest in den Kernel eingebunden. Sie benutzen stattdessen die Modules–Schnittstelle des Kernels, die seit Linux–Version 0.99.15 zur Standarddistribution gehört. Zum Laden eines Moduls gibt es das insmod–Programm (→ S. 201). Module können mit dem rmmod–Kommando (→ S. 211) wieder aus dem Kernel entfernt werden.

## 2.4 Das Verzeichnis `/etc`

Im Verzeichnis `/etc` befinden sich Konfigurationsdateien für das Basissystem. In früheren Linux–Distributionen waren in diesem Verzeichnis auch die meisten Programme für die Systemverwalterin untergebracht. Weil diese Vermischung dem Prinzip der funktionalen Trennung widerspricht und außerdem dieses wichtige Verzeichnis sehr unübersichtlich gemacht hat, wird mit dem File–System–Standard das neue Verzeichnis `/sbin` eingeführt, das die ausführbaren Dateien speziell für die Systemverwalterin aus dem Root–Filesystem aufnimmt.

Im Verzeichnis `/etc` verbleiben nur die echten Konfigurationsdateien für die Programme des Root–Filesystems, sowie die rechnerspezifischen Konfigurationsdateien für Programme der restlichen Distribution.

Alle Dateien dieses Verzeichnisses enthalten normalen Text und können/müssen von der Systemverwalterin mit einem normale Editor bearbeitet werden, um die von diesen Konfigurationsdateien gesteuerten Programme an das lokale System anzupassen.

Ein Listing des Standard–`/etc`–Verzeichnisses sieht beispielsweise so aus:

```
$ ls -F
HOSTNAME      fdprm         hosts.equiv   motd          rpc
```

```
Mail.rc      fstab        hosts.lpd    mtab         securetty
Xconfig      ftpusers     inetd.conf   mtools       services
aliases      gateways     inittab      networks     shadow
bootptab     gettydefs    issue        nntpserver   shells
csh.login    group        keytables/   passwd       smail.conf
default/     gshadow      ld.so.conf   printcap     syslog.conf
diphosts     host.conf    lilo.conf    profile      termcap
disktab      hosts        localtime@   protocols    ttytype
elm.rc       hosts.allow  login.defs   rc.d/
exports      hosts.deny   manpath.conf resolv.conf
$ _
```

Aus Gründen der Systemsicherheit sollten "normale" Benutzer die Dateien im Verzeichnis `/etc` nur lesen, aber nicht beschreiben können.

### /etc/fdprm

Die Datei `/etc/fdprm` stellt eine Liste der möglichen Parameter zur direkten Konfigurierung der Diskettenlaufwerke mit dem `setfdprm`–Kommando bereit. Durch Veränderung dieser Datei kann die fachkundige Systemverwalterin non–standard Diskettenformate in den Floppy–Gerätetreiber programmieren.

### /etc/fstab

In der Datei `/etc/fstab` sind die dauerhaften Parameter für die Zusammensetzung des Dateisystems gespeichert. Hier legt die Systemverwalterin 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 vier 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 einer Gerätedatei 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 bezeichnet den Typ des Dateisystems. Vorausgesetzt, der entsprechende Treiber wurde bei der Kernelkonfiguration aktivert, können die folgenden Dateisystemtypen angegeben werden:

**minix** ist das von A. Tanenbaum's Minix Betriebssystem übernommene Dateisystem. Es auch heute noch häufig für Diskettendateisysteme benutzt. Die Partitionsgröße ist auf maximal 64 MB beschränkt. Die Dateinamen können 14 (30) Zeichen lang sein.

**ext2** ist das zweite erweiterte Dateisystem von Remy Card. Es kann als das Standarddateisystem für neuere Linux–Installationen bezeichnet werden. Die Dateinamen können bis zu 255 Zeichen lang sein, die maximale Größe für eine Partition liegt bei 2 GB.

**ext** ist das alte erweiterte Dateisystem von Remy Card. Es ist durch das zweite erweiterte Dateisystem `ext2` abgelöst worden und ist nur noch auf alten Linux–Installationen zu finden.

**xiafs** ist das alternative Linux–Dateisystem von Frank Q. Xia. Es ist "schlanker" als das `ext2`, bietet aber weniger Features und Erweiterungsmöglichkeiten.

**msdos** ist das DOS Dateisystem. Linux kann DOS Disketten oder Festplattenpartitionen fest in den Verzeichnisbaum einbinden.

**proc** ist das Prozeßdateisystem. Es handelt sich hierbei nicht um ein "physisches" Dateisystem sondern um eine Repräsentation relevanter Daten aus dem laufenden Linux–Kernel. Dieses Dateisystem wird üblicherweise auf dem Verzeichnis `/proc` aufgesetzt.

**nfs** ist das Network File System. In diesem Dateisystem können Verzeichnisäste verschiedener Rechner im lokalen Netzwerk ausgetauscht werden.

**iso9660** ist das Dateisystem auf CD–ROMs. Der Linux–Kernel versteht die "Rock Ridge Extensions" zum Standard–ISO9660–Dateisystem, mit denen beispielsweise lange Dateinamen benutzt werden können.

**hpfs** ist das Dateisystem von OS/2. Dieses Dateisystem kann zur Zeit nur zum Lesen gemountet werden.

**xenix** ist das erste der drei unterstützten System V Dateisysteme. Wie der Name schon sagt, kann mit dieser Option ein Xenix–Dateisystem gemountet werden.

**coherent** ist das Coherent–Dateisystem. Es werden nur Dateisysteme erkannt, die keinen Dateisystem–Namen und keinen Pack–Namen haben.

**sysv** ist das Standard–Dateisystem von Unix System V Release 2. Dieses Dateisystem wird zur Zeit nur auf Disketten unterstützt, Festplattenpartitionen können nicht gemountet werden.

**swap** ist kein Dateisystem sondern die Kennzeichnung für den Swapbereich, in den schlafende Prozesse (teilweise) ausgelagert werden, um Platz für die laufenden Prozesse im Arbeitsspeicher zu schaffen.

**ignore** ist der Eintrag zur Kennzeichnung von Partitionen, die hier nur so herumstehen und ignoriert werden sollen.

Der vierte Eintrag der `/etc/fstab` 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.

**suid** die SUID und SGID Bits an ausführbaren Dateien und an Verzeichnissen dieser Partition werden behandelt, wie im Abschnitt über Eigentum und Zugriffsrechte ab Seite 230 beschrieben.

**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.

**rw** erlaubt das Lesen und das Schreiben auf dieser Partition.

**ro** verbietet das Schreiben auf diese Partition.

**sw** kennzeichnet eine Swappartition.

**defaults** entspricht "normalerweise" der Kombination `suid`, `rw`. Im Spezialfall des "usermount" Kommandos entspricht `default` für nicht privilegierte Benutzer der Kombination `nosuid`, `noexec`, `nodev`, `ro`.

**sync** die Lese–/Schreibfunktionen werden ungepuffert (synchron) ausgeführt (nur beim ext2fs implementiert).

**check** veranlaßt den Kernel, beim Mounten eines ext2fs eine Konsistenzprüfung durchzuführen. In der Linux-Version 1.0 werden bei diesem Test die Zahlen der freien I-Nodes und Datenblöcke in Super- und Gruppen–Block mit den Bitmaps verglichen. In zukünftigen Versionen werden weitere Konsistenzprüfungen hinzukommen. Wenn der Kernel einen Fehler feststellt, wird das Valid–Flag auf einen speziellen Wert gesetzt, um dem e2fsck–Programm den Fehler anzuzeigen (nur für ext2fs).

**sb=*Zahl*** veranlaßt den Kernel, den Superblock aus dem mit der Zahl bezeichneten Block einer ext2fs Partition zu lesen. Das ext2fs legt typischerweise Kopien des Superblocks in den Blöcken 8193, 16385 usw. an (nur für ext2fs).

Wenn das erste Zeichen einer Zeile ein '#' ist, wird die komplette Zeile als Kommentar behandelt und ignoriert.

### /etc/gettydefs

Diese Datei wird von dem im System V Stil arbeitenden getty_ps ausgewertet. Hier werden die Einstellungen des Terminaltreibers für die verschiedenen Geräte festgelegt. Eine genaue Beschreibung finden Sie im Abschnitt über getty_ps .

`/etc/group`

In dieser Datei sind die Benutzergruppen und ihre Mitglieder festgehalten.
Die Datensätze der group Datei haben folgendes Format:
*Gruppenname:Paßwort:Gruppennummer:Mitgliederliste*
Wie bei der Paßwortdatei gibt es auch für die Gruppendatei eine Erweiterung durch das Shadow–Paßwortsystem. Die Gruppenpaßwörter, die in der normalen group Datei wie in der passwd Datei für alle Systembenutzer verschlüsselt, aber lesbar gespeichert sind, werden hier in der separaten Datei `/etc/gshadow` gespeichert. Hier werden auch zusätzliche Informationen zur Gruppe gespeichert, beispielsweise der Name des zum Ein– und Austragen von Mitgliedern autorisierten Gruppenverwalters.
Neben den Gruppen, die die Systemverwalterin 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 der Superuserin ruth

**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 seine 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

Die Gruppenkennzahlen in Klammern sind, mit Ausnahme von 0 für root, beliebige Beispiele.
Mit dem `newgrp`–Kommando (→ Seite 164) kann jeder hier eingetragene Anwender die aktuelle Benutzergruppe wechseln. Wenn im Paßwort Feld ein verschlüsseltes Paßwort wie im entsprechenden Feld der `/etc/passwd` Datei eingetragen ist, können alle Systembenutzer, die das unverschlüsselte Paßwort kennen, in diese Gruppe wechseln.[8]

[8] Das verschlüsselte Paßwort kann beispielsweise von der Systemverwalterin aus der `/etc/passwd` in die `/etc/group` Datei kopiert werden, nachdem ein entsprechender Eintrag durch das `passwd`–Kommando erstellt wurde. Das `passwd` Kommando vom Shadow–Paßwortsystem erlaubt, das Gruppenpaßwort direkt zu ändern.

Sinn vieler der oben aufgeführten Gruppen ist es, den Benutzern den kontrollierten Zugriff auf bestimmte Teile des Systems (z.B. News, UUCP, Diskettenlaufwerke, Drucker) zu ermöglichen, indem bestimmte Progamme SGID laufen. Dadurch erhalten die Benutzer zur Laufzeit des Programms die Rechte der entsprechenden Gruppe, ohne selbst Mitglied dieser Gruppe zu sein.

### /etc/hosts

Die Datei `/etc/hosts` gehört zu den Konfigurationsdateien des TCP/IP Netzwerkes. Hier werden die typischen vier Byte langen IP–Adressen den verbalen Namen der Netzwerkrechner fest zugeordnet. In Jugend des Internet mußten in dieser Datei alle Rechner des internationalen Netzes eingetragen sein, zu denen eventuell irgendwann einmal eine direkt benannte Verbindung aufgebaut werden könnte (das heiß eigentlich jeder Rechner des Internet).

Unter Linux wird die Auflösung eines Rechnernamens in seine IP–Adresse (und umgekehrt) durch den Domain Name Service mit dem **bind**–Paket erledigt. Um bei Einzelrechnern und kleinen lokalen Netzwerken den administrativen Aufwand eines Nameservers zu sparen, kann trotzdem weiterhin die `/etc/hosts` Datei benutzt werden. Dazu wird der als *resolver* bezeichnete Teil des Nameservers durch entsprechende Einträge in der Datei `/etc/host.conf` veranlaßt, zuerst in der `hosts` Datei nach einer passenden Auflösung für eine Adresse zu suchen. Die Funktionen des resolver sind in der C–Standardbibliothek enthalten, deshalb wird er automatisch von allen Programmen mit Netzwerkfähigkeit (z.B. ftp, telnet, smail, aber auch vom X–Server) benutzt.

Die Einträge in `/etc/hosts` sind einfacher Text und bestehen aus den IP–Adressen der Hostrechner am Anfang einer Zeile und den offiziellen oder inoffiziellen Namen dieser Rechner, jeweils durch Leerzeichen oder Tabulatoren getrennt.

```
127.0.0.1       localhost atlantis.lunetix.de
193.98.158.225  atlantis atlantis.lunetix.de
193.98.158.1    soho soho.lunetix.de
193.98.158.2    cicero cicero.lunetix.de
193.98.158.5    ovid ovid.lunetix.de
```

### /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 getrennten Einträgen.

*Terminal*:*Termcapeintrag*:*Gettykommando*

Der erste Eintrag bezeichnet das Terminal (`tty1`, `tty2`, `ttyS1` ...), wie es in die `/etc/utmp` Datenbank eingetragen 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 getrennten 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/initrunlvl` 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 einem Mehrbenutzermodus gestartet oder vom Einbenutzermodus dorthin gewechselt wird. Das init Programm wartet mit der Bearbeitung weiterer Zeilen in der `inittab`, bis das in dieser Zeile aufgerufenen Kommando beendet ist.

**boot** Das Kommando im vierten Feld dieser Zeile wird wie bei bootwait ausgeführt, jedoch wird sofort mit der Bearbeitung der `inittab` fortgefahren, ohne auf die Beendigung des Kommandos zu warten.

**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. Das bedeutet, daß init jedes mit respawn gestartete Kommando dauernd überwacht. Wenn das Kommando normal beendet oder während der Laufzeit durch ein Signal terminiert wird, wird es sofort wieder neu gestartet. Wenn das Kommando aufgrund eines Fehlers unmittelbar nach dem Start abbricht, gibt init das Kommando nach einer bestimmten Anzahl von Versuchen auf.

**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 für die Synchronisation des Dateisystems und ersetzt so das update–Kommando. Im vierten Feld wird angegeben, in welchem Abstand (in Sekunden) der sync Systemaufruf ausgeführt werden soll. Die update Funktion wird erst nach sysinit gestartet, so daß von sysinit beispielsweise das Root–Filesystem geprüft und repariert werden kann.

**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/initrunlvl`

Diese Datei wird vom sysvinit-Programm benutzt, das daraus den Runlevel nach dem Systemstart ausliest. Dieser Parameter verdeckt den entsprechenden Eintrag in der Konfigurationsdatei `/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. Zusammen mit dem getty_ps können in der `issue` Datei bestimmte Sonderzeichen und Platzhalter verwendet werden.

### /etc/login.defs

Die Datei /etc/login.defs gehört zum login Programm aus dem Shadow Paßwortsystem. Es liest daraus die Parameter mit denen beispielsweise eingestellt wird, wie oft ein fehlgeschlagenes Login wiederholt werden darf und ob und wo die Meldungen über solche Fehlschläge festgehalten werden. Hier können auch einige Umgebungsvariablen für den Benutzer festgelegt werden, die der Shell (oder einem anderen Programm) mitgegeben werden. Diese Datei ist sehr ausführlich kommentiert.

### /etc/motd

In dieser Datei kann 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 die Superuserin (ruth) 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 Benutzernummern und das Heimatverzeichnis der Anwender gespeichert. Außerdem werden in der "normalen" passwd Datei auch die verschlüsselten Paßwörter gespeichert. Für öffentlich zugängliche Systeme mit besonders hohen Sicherheitsansprüchen gibt es ein sogenanntes Shadow–Paßwortsystem. Hier werden die Paßwörter in der separaten Datei shadow gespeichert, in der zusätzliche Information, beispielsweise über das "Alter" eines Paßwortes gespeichert ist. Die shadow Paßwortdatei ist im Unterschied zur passwd Datei für die Systembenutzer unlesbar.
Die Datensätze der Paßwortdatei bestehen aus:
*Benutzername*:*Paßwort*:*Benutzernummer*:*Gruppennummer*:*GCOS*:*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. Im Shadow–System ist eine "Paßwortalterung" eingebaut, die den Benutzer nach einer gewissen Zeit zwingt, das Paßwort zu ändern.
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:

**ruth** (0) die Superuserin

**root** (0) der traditionelle Eintrag für den Superuser[9]

**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

[9] Der traditionelle root Eintrag sollte immer in /etc/passwd enthalten sein. Um die "persönliche" Zuordnung des privilegierten Accounts zu erreichen, muß der entsprechende Eintrag in einer Zeile vor dem root Account stehen.

**ftp** (10) der anonyme FTP Zugang

**nuucp** (11) der anonyme UUCP Zugang

Die Benutzerkennzahlen sind mit Ausnahme der Null für root beliebig.

Einige der oben genannten Accounts (z. B. bin oder daemon) sind mit Sperrpaßwörtern 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 36). 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 Drucker des Systems. Sie wird vom lpd Druckerdämon ausgewertet, der die Druckjobs im System verwaltet. Eine Beschreibung ist im Kapitel über Dämonen auf Seite 246 zu finden.

### /etc/profile

Die Datei `/etc/profile` wird von allen Loginshells (aller Benutzer) gelesen und als Shellscript 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.

**Beispiel:**

```
# /etc/profile

export OPENWINHOME=/usr/openwin
export DISPLAY=":0"

PATH="/bin:/usr/bin:/usr/local/bin:/usr/X386/bin:$OPENWINHOME/bin"
PS2='> '
umask 022
ulimit -c 0
```

### /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. Das kernelunabhängige procps benötigt diese Datenbank auch, wenn es die Namen der Kernelfunktionen anzeigen soll, in denen ein Prozeß schläft.[10]

Die Datei `/etc/psdatabase` wird vom kernelabhängigen ps mit der Option '−U' angelegt. Dazu wird die Datei `/usr/src/linux/tools/system` gelesen, in der Daten über den Kernel enthalten sind. Das procps hat die Option nicht. Stattdessen kommt das procps mit dem psupdate–Kommando, das die gleiche Funktion erfüllt.

[10] Die anderen Funktione des procps Paketes benötigen die psdatabase nicht mehr. Sie beziehen alle Information aus den "Dateien" im Prozeßdateisystem.

### /etc/rc

Die Datei /etc/rc ist die Systeminitalisierungsdatei. 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.

Die genaue Funktion und der Aufbau der Systeminitialisierungsdatei kann sich für die beiden Versionen von init ganz erheblich unterscheiden. Dabei ist wieder das sysvinit das bei weitem kompliziertere, aber auch das vielseitigere der beiden Modelle.

Mike Jagdis hat seinem sysvinit eine ganze Kaskade von Initialisierungsdateien mitgegeben, die bei jedem Moduswechsel bestimmen, welche Prozesse gestartet werden sollen, welche angehalten und welche weiterlaufen sollen. Dazu wird das /etc/rc–Script von init mit einem Parameter aufgerufen, der genau dem gewünschten Runlevel entspricht. Das /etc/rc–Script sucht seinerseits in einem dem Runlevel entsprechend benannten Verzeichnis nach weiteren Scriptdateien. Wenn in den Runlevel 4 gewechselt werden soll, sucht es beispielsweise in dem Verzeichnis /etc/rc4.d

Die Scriptdateien in den rc?.d Verzeichnissen beginnen allen entweder mit einem K für kill oder einem S für start. Zuerst werden die Killscripts aufgerufen, denen der Parameter "stop" in der Kommandozeile übergeben wird. Danach werden die Startscripts mit dem Parameter "start" ausgeführt. Die Parameter werden von den mit dem sysvinit gelieferten Shellscripts ausgewertet. Es werden die gleichen Scriptdateien zum Starten und zum Anhalten der Prozesse verwendet. Das ermöglicht es, einen Satz dieser Dateien in dem zentralen Verzeichnis /etc/init.d zu installieren und diese Dateien mit Links auf die entsprechenden Start– und Killscripts in den Verzeichnissen für die einzelnen Runlevel zu verbinden.

Weil es sich um normale Shellprogramme handelt, wird die genaue Arbeitsweise der Scripts nicht weiter erklärt (→ bash, Seite 54).

Das **simpleinit** kann mit diesen Initialisierungskaskaden nicht viel anfangen, weil es keine Runlevel unterstützt. Es benutzt die /etc/rc Datei viel direkter, beispielsweise um den update Prozeß im Hintergrund zu starten oder den cron Dämon aufzurufen. Hier wird auch das Dateisystem montiert, indem das mount -a Kommando aufgerufen wird, und im Dateisystem aufgeräumt, indem überflüssige Dateien gelöscht und die Partitionen gecheckt werden.

Natürlich kann auch das sysvinit mit einem einfachen rc File ohne den ausgefeilten Start/Stop Mechanismus arbeiten. In den aktuellen Distributionen werden die Initialisierungdateien von Mike Jagdis nicht mitgeliefert.

Um auch beim simpleinit eine gewisse Strukturierung für die doch manchmal sehr umfangreichen Initialisierungen einzuführen, werden aus der zentralen /etc/rc Datei heraus weitere Shellscripts mit speziellen Aufgabenbereichen aufgerufen.

**rc.local** Mit diesem Script werden typischerweise lokale Einstellungen, wie zum Beispiel die Festlegung des Systemnamen mit hostname, die Änderung der Tastaturtabelle mit dem loadkeys–Kommando oder die Einstellung der Systemzeit mit clock, vorgenommen.

**rc.inet** In älteren Distributionen (vor Kernel 0.99.10) wurden mit dem Script rc.net die TCP/IP Netzwerkfunktionen des Kernels initialisiert und die Netzwerkdämonen gestartet. Seit der Einführung des Net–2 Codes werden stattdessen die Dateien rc.inet1 und rc.inet2 benutzt, mit denen die Kernelinitialisierung und die Dämonenbeschwörung getrennt werden. Eine Beschreibung der TCP/IP Initialisierung würde den Rahmen dieses Handbuches sprengen. Im Anhang sind Hiweise auf andere Bücher enthalten, die dieses Thema erschöpfend behandeln.

**rc.serial** Im Zusammenhang mit dem setserial Programm kommt gelegentlich dieses Script zur Initialisierung der seriellen Schnittstellen zum Einsatz. Hier werden die seriellen Gerätetreiber für alle vom Standard abweichenden Konfigurationen initialisiert.

### /etc/securetty

In der Datei /etc/securetty werden die Ports (Terminals) angegeben, an denen sich die Superuserin (ruth) 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/syslogd.conf

Der syslogd Systemschreiber erfährt aus dieser Datei, welche Meldungen er wohin schreiben soll. Eine genaue Beschreibung der Datei finden Sie auf Seite 253.

### /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 bei der Shellinitialisierung gesetzt wird.

Eine genaue Erklärung aller möglichen Einträge dieser Datei würde den Rahmen dieses Buches sprengen. Es gibt eine TEXinfo–Datei zu `termcap` und Tim O'Reilly hat Bücher über `termcap` und über die curses–Library herausgebracht, die Ihnen alle Fragen zu diesem Thema beantworten können.

### /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 Root–Filesystem vor Beschädigung zu schützen, wird diese Datei auch häufig durch ein Link mit einer gleichnamigen Datei in `/usr/adm` oder einem anderen Verzeichnis in einem weniger sensiblen Systembereich verbunden.

Beim Hochfahren des Systems sollte diese Datei geleert werden.

### 2.4.1 Nichts über TCP/IP

Der komplette Thema des TCP/IP Networking ist auch in dieser Auflage des Linux–Anwenderhandbuches ausgeklammert. Damit Sie einen groben Überblick erhalten, welche Dateien und Programme in diesen Bereich fallen, sind sie hier aufgeführt.

| | |
|---|---|
| HOSTNAME | Fully Qualified Domain Name |
| diphosts | eine Art `/etc/passwd` für SLIP Login |
| exports | Verzeichnisse und Rechnernamen, die diese Verzeichnisse per NFS mounten dürfen. |
| ftpaccess | Woher? Wann? Was? ... |
| ftpusers | ... und Wer NICHT? |
| gateways | Hier werden die Rechner des lokalen Netzes aufgeführt, die Kontakt mit andern Netzen haben. |
| host.conf | eine der Konfigurationsdateien für den Domain Name Service |
| hosts.allow | Hier werden die Rechner und Netze eingetragen, mit denen Verbindungen aufgebaut werden dürfen. |
| hosts.deny | Diese Rechner und Netze können sich nicht mit diesem Rechner verbinden. |
| hosts.equiv | Diese Rechner werden "gleichrangig" behandelt. |
| inetd.conf | In dieser Konfigurationsdatei wird bestimmt, welche Services vom `inetd` Netzwerkdämon gestartet werden. |
| named.boot | eine Initialisierungsdatei für den `named` Dämon des Domain Name Services. |
| networks | die Namen und IP–Adressen der lokalen Netze. |
| nntpserver | der Name und die IP–Adresse des News–Rechners. |
| protocols | die Liste aller Internet–Protokolle die vom Kernel unterstützt werden. |
| resolv.conf | Diese Konfigurationsdatei bestimmt, ob und wo ein Name–Server erreicht werden kann. |
| rpc | enthält die Zuordnung der durch den `rpc`–Service bereitgestellten Netzdienste zu Socketnummern. |
| services | Diese Datei enthält eine Liste aller "well known services", sowie einer Reihe weiterer Standard–Services im Internet. |

## 2.5 Home, Sweet Home

Nachdem die Reise in den beiden vorhergehenden Verzeichnissen durch Neuland und unwegsames Gelände führte, erreichen Sie nun in einem Unterverzeichnis von `/home` Ihren persönlichen Bereich im Linux–System.

Im `/home` Verzeichnis befinden sich die privaten Verzeichnisse, die Heimatverzeichnisse aller Systembenutzer. Die Existenz solcher individuellen Bereiche ist für ein Mehrbenutzersystem selbstverständliche Notwendigkeit. Auch wenn Sie die einzige Benutzerin Ihres Systems sind, sollten Sie sich einen Account ohne Privilegien und ein Heimatverzeichnis anlegen.

In Ihrem Heimatverzeichnis können Sie nach belieben Verzeichnisse anlegen und Daten speichern und löschen.[11] Hier werden Sie Ihre persönliche eMail aufbewahren und Ihre Projektdateien anlegen, zum Beispiel die Entwürfe für ein neues, besseres Linux–Handbuch.

Im Sinne des File–System–Standard zählt der `/home` Bereich damit eindeutig zum variablen, unbegrenzt wachsenden Systembereich. In lokalen TCP/IP Netzwerken, in denen Sie als Benutzerin an verschiedenen Rechnern arbeiten können, wird das `/home` Verzeichnis möglicherweise über NFS verteilt auf allen Rechnern

[11] Durch eine Erweiterung des Linux–Dateisystems — das sogenannte `quota`–System — kann der frei verfügbare Festplattenplatz von der Systemverwalterin zugeteilt (quotisiert) werden.

des Netzes gemeinsam benutzt. Das hat den Vorteil, daß Sie an jedem Arbeitsplatz immer in dem selben Heimatverzeichnis arbeiten.
Auch wenn der `/home` Bereich lokal angelegt ist, wird er normalerweise auf einer eigenen Partition untergebracht. Diese physikalische Trennung der System– und Benutzer–Daten erhöht die Betriebssicherheit, sie erleichtert ein System–Update und die Datensicherung.

### Das möblierte Zimmer

Wenn Sie sich an einem Linux–Rechner einloggen, ist Ihr Heimatverzeichnis immer das erste Arbeitsverzeichnis. Neben Ihren privaten Daten befinden sich deshalb in diesem Verzeichnis noch individuelle Initialisierungsdateien, mit denen Sie Ihre Arbeitsumgebung einrichten und in dem für eine Computeroberfläche möglichen Rahmen individuell gestalten können.
Eine Standard–Ausstattung mit Initialisierungsdateien wird Ihnen von der Systemverwalterin beim Einrichten Ihres Accounts ins Heimatverzeichnis gestellt. Bei einem normalen Listing werden Sie diese Dateien nicht sehen, weil ihre Namen mit einem Punkt beginnen. Durch den Schalter **−a** (für all) werden aber auch die "versteckten" Dateien in ihrem Heimatverzeichnis angezeigt.

```
$ ls -F
Mail/    News/    bin/     public/
$ ls -aF
./             .bash_login    .seyon/        Mail/
../            .bash_logout   .tcshrc        News/
.Xmodmap       .bashrc        .term/         bin/
.bash_history  .emacs         .xinitrc       public/
$ _
```

Die "sichtbaren" Verzeichnisse `Mail` und `News` gehören zum Mailer–Front–End bzw. zum News–Reader und werden automatisch erzeugt, wenn sie beim ersten Aufruf des entsprechenden Programms nicht existieren. `~/bin` kann in den Suchpfad der Shell integriert werden und ist für private, ausführbare Dateien vorgesehen. `public` dient zur Aufbewahrung von Dateien, die auch anderen Systembenutzern zugänglich sein sollen.
Die "unsichtbaren" Dateien `.Xmodmap` und `.xinitrc` werden im Kapitel zum X11 Window System erklärt. Die Verzeichnisse `.seyon` und `term` werden im Kapitel über Datenreisen beschrieben.
Alle Dateien, deren Namen mit `.bash` beginnen, gehören zur `bash`, der Bourne–Again–Shell von Chet Ramey und Brian Fox. Die Funktion dieser Dateien ist mit Beispielen in der Kommandobeschreibung ab Seite 98 erklärt.
Die verbleibenden Dateien `.tcshrc` und `.emacs` sind an keiner anderen Stelle des Buches erklärt, deshalb folgt hier jeweils ein Beispiel.

```
$ cat .tcshrc
# tcshrc
if ($?prompt) then
        umask 022
        set cdpath = ( ~ /usr/src /usr )
        set notify
        set history = 100
        setenv OPENWINHOME /usr/openwin
        set path = ( /bin /usr/bin /usr/local/bin /usr/X386/bin\\
            $OPENWINHOME/bin /usr/games ~/bin . )
endif
set prompt = "%m:%~%# "
set term = console
alias pwd 'echo $cwd'
alias ls 'ls -F'
alias term 'term < /dev/cua1 > /dev/cua1 2> /dev/null &'
$ _
```

```
$ cat .emacs
(add-hook 'text-mode-hook '(lambda () (auto-fill-mode 1)))
(standard-display-european 1)
(transient-mark-mode 1)
$ _
```

## 2.6 Das Verzeichnis /lib

Im Verzeichnis `/lib` befinden sich die "Shared Libraries" für die dynamisch gelinkten Programme des Root-Filesystems.

Im File–System–Standard wird gesagt, daß die eigentlichen C–Funktionsbibliotheken und die Shared Libraries für das X Window System nicht nach `/lib` gehören, weil die damit arbeitenden Programme nicht auf dem Root–Filesystem angesiedelt sind, sondern zum verteilten Systembereich gehören.

Um die Rückwärtskompatibilität zu sichern, wird allein ein symbolischer Link von `/lib/cpp` auf den C–Präprozessor zugelassen.

## 2.7 Das Verzeichnis /proc

Das Verzeichnis `/proc` beherbergt das Prozeßdateisystem. Mehr Information zu diesem Thema finden Sie im Kapitel über Dateisysteme, ab Seite 298.

## 2.8 Das Verzeichnis /sbin

Eine der konzeptionellen Neuerungen durch den File–System–Standard ist das `/sbin` Verzeichnis. Es enthält alle Programmdateien für die essentiellen Aufgaben der Systemverwaltung, deren Ausführung der Superuserin vorbehalten ist.

Durch die Einführung dieses Verzeichnisses kann einerseits das bisher mit diesen Programmen stark überlastete und unübersichtliche Verzeichnis `/etc` klarer strukturiert werden, anderseits bleiben die Programme, deren Ausführung nur mit Privilegien möglich ist, weiterhin von den normalen Anwenderprogrammen getrennt.

Ähnlich wie beim Verzeichnis `/bin` sieht der File–System–Standard auch beim `/sbin`–Verzeichnis eine Beschränkung auf die essentiell notwendigen Programme vor.

```
$ ls
arp         fsck.ext2   ifsetup     mkfs.minix  reboot      swapon
brc         fsck.minix  init        mkfs.xiafs  route       telinit
chown       fsck.xiafs  loadkeys    mknod       setserial   umount
e2fsck      getty       mke2fs      mkswap      shutdown    update
elvprsv     halt        mkfs        mount       ssync       uugetty
fsck        ifconfig    mkfs.ext2   rdev        swapoff
$ _
```

## 2.9 Das Verzeichnis /tmp

Im Verzeichnis `/tmp` werden von einigen Programmen temporäre Dateien zur Zwischenspeicherung von Laufzeitdaten angelegt. Zum Erzeugen einer Datei in diesem Verzeichnis braucht ein Prozeß keine besonderen Rechte. Das Löschen von Dateien ist nur den Prozessen des Datei–Eigentümers erlaubt, wenn das Sticky–Bit für das `/tmp` Verzeichnis gesetzt ist.

Weil in `/tmp` Dateien unvorhersehbarer Größe zu unberechenbaren Zeitpunkten erzeugt werden und um die übermäßige Fragmentierung der Root–Partition zu verhindern, wird das Verzeichnis `/tmp` normalerweise nicht im Root–Filesystem angesiedelt. Zur "Umsiedlung" gibt es verschiedene Möglichkeiten: zum Beispiel kann anstelle eines echten Verzeichnisses ein symbolischer Link auf das Verzeichnis `/var/tmp` angelegt werden. Eine andere Möglichkeit besteht darin, eine eigene Partition oder auch eine RAM–Disk auf dieses Verzeichnis aufzusetzen.

Der temporäre Charakter der Daten im `/tmp`–Verzeichnis wird dadurch unterstrichen, daß der Inhalt dieses Verzeichnisses in beliebigen Abständen, beispielsweise bei jedem Systemstart, gelöscht wird.

## 2.10 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.

Alle Programme des `/usr` Systems sind für alle Rechner eines lokalen Netzes gleichermaßen interessant und/oder wichtig. Deshalb wird mit dem File–System–Standard festgelegt, daß die Daten im `/usr` Ast des Dateisystems über ein lokales Netz verteilt benutzbar sein sollen. Eine wichtige Voraussetzung dafür ist, daß dieser Ast "Read–Only" gemountet werden kann.

Um das zu erreichen, wird ein weiterer Ast des Dateisystems mit dem Namen `/var` eingeführt, der die variablen Teile des `/usr` Systems aufnimmt.

Ein beispielhaftes Listing des `/usr`–Verzeichnisses zeigt die folgenden Unterverzeichnisse:

```
$ ls -F /usr
TeX/          dict/         games/        local/        spool@
X11@          doc/          include/      man/          src/
X386/         etc/          info/         sbin/         tmp@
bin/          g++-include/  lib/          share/
$ _
```

Der Eintrag `X11` ist ein symbolischer Link auf `/usr/X386`. Der Eintrag `tmp` ist in diesem Beispiel ein symbolischer Link auf `/var/tmp`, es ist aber ebensogut möglich, daß hier eine eigene Partition oder eine RAM–Disk darauf aufgesetzt wird.

### /usr/TeX

In diesem Zweig des Verzeichnisbaums befindet sich das TEX Satzsystem von Donald Knuth. Mit diesem hervorragenden Programmpaket und den dazugehörenden Makros und Hilfsprogrammen lassen sich Geschäftsbriefe ebenso einfach setzen, wie umfangreiche Buchprojekte im Format dieses Handbuches realisieren.[12]

Das TEX System ist übrigens nicht Teil des File–System–Standard Entwurfs und auch nicht in jeder Distribution enthalten.

Wenn das TEX–Paket installiert ist, sollten Sie das Verzeichnis `/usr/TeX/bin` durch einen entsprechenden Eintrag in Ihrer Shell–Initialisierungsdatei in den PATH einfügen.

### /usr/X386

In den Unterverzeichnissen von `/usr/X386` befindet sich das X Window System — die grafische Benutzeroberfläche von Linux.

[12] Die Verwendung von TEX ist eine der Glaubensfragen von Linux. Die Oberfläche ist spröde, wie die vom vi. Aber mit Hilfe von Tastaturmakros lassen sich wunderbar einfach die scheinbar lästigen Satzbefehle in den Text einfügen. Der emacs bietet mit dem AUC–TEX Makropaket für LATEX zusätzlichen Komfort. Wie bei allen Glaubensfragen liegt die Entscheidung für oder gegen TEX (und LATEX) natürlich allein bei Ihnen. Als Autor eines mehrere hundert Seiten umfassenden LATEX–Dokuments erlaube ich mir, Sie zu einem Versuch zu ermutigen.

### /usr/bin

In diesem Verzeichnis befinden sich fast alle Benutzerkommandos der Linux–Distribution. Allein die zu den essentiellen Aufgaben der Systemverwaltung notwendigen Benutzerkommandos befinden sich in `/bin` und die Programme von Paketen mit eigenen Zweigen im Verzeichnisbaum sind in den `bin` Verzeichnissen dieser Zweige untergebracht.

### /usr/dict

Dieses Verzeichnis ist für die Wörterbücher (dictionaries) der ispell Rechtschreibkorrektur vorgesehen.

### /usr/doc

Der File–System–Standard sieht dieses Verzeichnis für verschiedene Dokumentationen von allgemeinem Interesse vor.

### /usr/etc

Alle Konfigurationsdateien zu einer Linux–Distribution, die nicht zu Programmen im Root–Filesystem gehören und nicht rechnerspezifisch sind, gehören in das Verzeichnis `/usr/etc`. Beispielsweise die `magic` Datenbank für das file–Kommando, die Musterdateien für die Benutzerverwaltung und die Konfigurationsdatei für den syslogd gehören hierher.

### /usr/g++-include

Dieses Verzeichnis enthält die Header–Dateien für den GNU–C++ Compiler.

### /usr/games

```
Would you like to play a game? ttt
Funny, the only way to win is not to play at all.
```

### /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 zwischen der originalen Distribution und den lokalen Erweiterungen zu erreichen, gibt es hinter dem Verzeichnis `/usr/local` eine Hierarchie von Verzeichnissen, die im Prinzip der von /usr entspricht. Der File–System–Standard legt fest, daß keine Linux–Distribution Daten in diesem Zweig des Verzeichnisbaums anlegen darf.

### /usr/info

In diesem Verzeichnis sind die Textdateien des TEXinfo Dokumentations–Systems untergebracht.

### /usr/lib

Das Verzeichnis `/usr/lib` beherbergt traditionell den C–Compiler und die C–Funktionsbibliotheken, sowie unveränderliche, nichtausführbare Arbeitsdaten zu verschiedenen Programmpaketen.

### /usr/man

Hier sind die Hilfe–Texte zu allen Kommandos, Dateiformaten, Spielen, C–Bibliotheksfunktionen etc., die sogenannten "manual pages" untergebracht. 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 oder gzip komprimiert sein. Das man–Kommando dekomprimiert den Inhalt automatisch vor der Ausgabe. Die Verzeichnisse jeder Gruppe sind numeriert. Die Numerierung entspricht den in der Erklärung vom man–Kommando beschriebenen Aufteilung der Manpages auf Seite 158.
Die bereits existierenden und die zukünfig noch entstehenden deutschen Manualpages werden, dem File-System-Standard entsprechend, in einem gleichartigen Satz von Unterverzeichnissen im Verzeichnis `/usr/local/man/german` untergebracht. Damit das man–Kommando korrekt darauf zugreifen kann, muß dieses Verzeichnis in der Datei `/etc/manpath.config` eingetragen sein.

### /usr/sbin

Zusammen mit dem Verzeichnis `/sbin` nimmt `/usr/sbin` die ausführbaren Dateien aus `/etc` und `/usr/etc` auf, deren Ausführung allein der Superuserin vorbehalten ist. Der im File–System–Standard festgelegten Trennung zwischen Root–Filesystem und verteiltem Benutzersystem entsprechend, werden in `/usr/sbin` alle Programmdateien der oben beschriebenen Art untergebracht, die nicht zu essentiellen Systemverwaltungsaufgaben notwendig sind.
Beispielsweise alle Programme zur Benutzerverwaltung aus dem Shadow–Paßwortsystem gehören hier her. Das folgende Listing zeigt beispielhaft das `/usr/sbin` Verzeichnis der Debian–Basisdistribution:

```
$ ls /usr/sbin
clock       iflink      level-1       pwck         showkey
crond       in.bootpd   logger        ramsize      stopalop
ctrlaltdel  in.ftpd     lpc           rdev         sulogin
dip         in.nntpd    lpd           readlog      swapdev
dirdump     in.ntalkd   lptest        rootflags    syslogd
dmesg       in.popd     makehole      routed       tcpd
doshell     in.rlogind  miscd         rpc.bwnfsd   tune2fs
dpasswd     in.rshd     mklost+found  rpc.mountd   updatedb
dumpkeys    in.telnetd  mktable       rpc.nfsd     useradd
fdformat    in.tftpd    named         rpc.pcnfsd   userdel
fdisk       inetd       named-xfer    rpc.portmap  usermod
fixperms    ipcrm       named.reload  rpc.ugidd    vidmode
frag        ipcs        named.restart rwhod        wdsetup
groupadd    kbdrate     newusers      selection
groupdel    kd          pac           sendlog
groupmod    ldconfig    pkg           setfdprm
grpck       level-0     psupdate      setserial
```

### /usr/share

Die Funktion des Verzeichnisses `/usr/share` besteht in der Zusammenfassung architekturunabhängiger Daten in verteilten Systemen mit unterschiedlichen Rechnerarchitekturen.

### /usr/src

Dieser Zweig des Dateisystems ist für die Aufnahme der Quelltexte (Sourcen) für alle Programme des Standardsystems vorgesehen.
Eine sinnvolle Unterteilung in Unterverzeichnisse bleibt der Systemverwalterin überlassen. Allein das Verzeichnis `/usr/src/linux` mit den Kernelsourcen wird von anderen Systemkomponenten genau an dieser Stelle erwartet. Besonders wichtig sind die symbolischen Links der Verzeichnisse `/usr/include/linux` und `/usr/include/asm` auf die entsprechenden Verzeichnisse in `/usr/src/linux/include`.

/usr/spool

Das Verzeichnis /usr/spool wird aus Gründen der Abwärtskompatibilität als symbolischer Link auf das neue Verzeichnis /var/spool weitergeführt.

## 2.11 Das Verzeichnis /var

Die Festlegung des /usr Verzeichnisses auf statische Daten mit dem Ziel, dieses Verzeichnis über NFS Read–Only mounten zu können, erfordert zwangsläufig die Auslagerung aller Dateien und Verzeichnisse, die für eine zweckmäßige Nutzung während des Betriebs verändert werden können/müssen. Diese Daten werden bei den neuen Linux–Distributionen in das Verzeichnis /var verlegt.
Ein Listing zeigt beispielsweise folgendes Bild:

```
$ ls -F
adm/       lock/      named/     preserve/  spool/     tmp/
$ _
```

Das Verzeichnis /var/adm enthält die System–Logfiles. Das sind vor allem die wtmp Datenbasis und die vom syslogd angelegten Dateien.

/var/lock ist das vom File–System–Standard empfohlene Verzeichnis zum Anlegen der Lockfiles.

/var/preserve ist das Sicherungsverzeichnis, in dem die nach einem Crash von elvis das unfertige Textfile automatisch durch das elvprsv–Kommando gesichert werden kann.

In Verzeichnis /var/spool werden die Daten der einzelnen News&Mail Programme in entsprechende Unterverzeichnisse abgelegt. Je nach Auslegung des Systems kann hinter dem Verzeichnis /var/spool/news eine sehr große Hierarchie beginnen, die für jede Nachrichtenrubrik ein Verzeichnis enthält. In diesen Verzeichnissen sind dann die einzelnen Artikel in numerierten Dateien abgelegt. Im Verzeichnis /var/spool/uucp werden die für eine Netzwerkkopie mit dem uucp Programm bzw. mit dem uucico Daemon vorgesehenen Dateien zwischengelagert. In /var/spool/lp werden die Dokumente des lpd Druckerspoolers zwischengespeichert, wenn in /etc/printcap kein anderes Spoolverzeichnis angegeben ist.

Im Verzeichnis /var/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 die Systemverwalterin (ruth) kann alle Dateien unabhängig von den Zugriffsrechten lesen.

# Kapitel 3

# Von GNU's, Muscheln und anderen Tieren

## 3.1 Intro(1) — oder die Erklärung der Erklärung

Das folgende Kapitel enthält Beschreibungen der wichtigsten Benutzerkommandos des Linux–Basissystems. Im darauffolgenden Kapitel werden die wichtigsten Kommandos für die Systemverwalterin erklärt. Beide Kapitel orientieren sich an den Manual–Pages, den Online–Hilfen, die mit den Programmen verteilt werden.

**Syntax:**

Nach einer knappen, einzeiligen Darstellung der Funktion eines bestimmten Kommandos wird nach dem Untertitel "**Syntax:**" der Aufruf auf der Kommandozeile mit allen Optionen dargestellt. Dazu wird eine strenge Form benutzt:
**name** [−Optionen ...] [*Argumente* ...]

**name** ist der Name des Kommandos. Das ist sowohl der vollständige Name der Datei, in der das Programm gespeichert ist als auch der Name, unter dem dieses Kommando aufgerufen werden muß. Besondere Endungen zur Kennzeichnung des Dateityps gibt es bei Linux nicht.

−Optionen sind "Befehle an das Programm", mit denen dessen Verhalten manipuliert werden kann. Eine Option ist durch einen einzelnen Buchstaben bezeichnet.[1] Bei den Optionen kann zwischen "Schaltern" und "Reglern" unterschieden werden. Während Schalter allein durch ihr Auftreten bzw. durch ihre Abwesenheit auf der Kommandozeile bestimmte Ereignisse auslösen, muß zu den Reglern noch ein Argument angegeben werden. Allen Optionen gemeinsam ist, daß sie durch ein Minuszeichen eingeleitet werden.

Neben den Optionen, die nur durch einen Buchstaben bezeichnet werden, gibt es bei den GNU Utilities häufig ausdrücklich benannte Optionen. Diese Optionen werden durch zwei Minuszeichen eingeleitet. In der Kommandobeschreibung werden diese Optionen nicht mehr aufgeführt. Ihre Funktion wird in allen Fällen auch durch Buchstabenoptionen abgedeckt.

*Arumente* können entweder zu den oben erklärten regelnden Optionen gehören (Optionsargumente), oder sie können sich direkt auf das Kommando beziehen (Kommandoargumente, Operanden). In jedem Fall soll durch die kursive Schrift angedeutet werden, daß diese Argumente nicht wörtlich eingegeben werden sollen. Argumente dürfen nicht mit einem Minuszeichen beginnen. Als Ausnahme ist als Kommandoargument meistens ein einzelnes Minuszeichen als Symbol für den Standardeingabekanal erlaubt.

[1] Von dieser einfachen und sinnvollen Regel gibt es prominente Ausnahmen. Beispielsweise haben alle Programme des X11 Window Systems einen gemeinsamen Satz von Schaltern und Reglern mit längeren Namen (z.B. `-title` `-geometry` oder `-fn` (font)). Die GNU Tools bieten zusätzlich zu den Buchstabenoptionen Schalter und Regler mit vollständigen verbalen Namen, die allerdings zur besseren Unterscheidung immer mit zwei Minuszeichen eingeleitet werden.

{ Argument1,Argument2} ist die Darstellung für Optionsargumente mit einem eingeschränkten Wertebereich. Hier darf nur eines der zur Auswahl stehenden Argumente wörtlich eingesetzt werden.

[ ] eckige Klammern schließen in der Regel Kommandoteile ein, die wahlfrei sind. 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.

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[2] angesprochen wird, sowie die Umschalttasten ALT, CONTROL und ESCAPE.

Gelegentlich werden Tastenkombinationen CONTROL–Irgendetwas angesprochen. Dabei wird neben der ausgeschriebenen Tastenbezeichnung auch die Abkürzung CTRL oder die symbolische Darstellung durch ein Dach (Caret) '^' verwendet. Zum Beispiel steht ^D für CONTROL-D: das ist das EOF–Zeichen. Dieses Zeichen wird durch gemeinsames Drücken der CONTROL- und der 'D'–Taste erzeugt.

**Beispiel:**

Beispiele werden in `Courier` (Schreibmaschinenschrift) gegeben. Wenn ganze Kommandozeilen vorgeführt werden, wird der $Standardprompt der Shell dargestellt. Die Zeilen bis zum nächsten Prompt zeigen gegebenenfalls die Bildschirmausgabe des Beispiels:

```
$ echo 'Hallo Welt!'
': Event not found.
$ _
```

**Siehe auch:**

Unter diesem Stichwort wird bei vielen Kommandobeschreibungen auf andere Stellen des Buches verwiesen, die weitere Information zum Thema bieten.

Zum Glück ist dieses Handbuch nicht die einzige Quelle gültiger Information. Ein umfangreiches Online–Hilfesystem bietet eine Vielzahl weiterer Antworten — wenn auch in der Regel in englischer Sprache. Am Ende vieler Kommandobeschreibungen in diesem Buch sind Verweise auf die Online–Hilfen gegeben.

Dabei kommen vor allem zwei Hilfe–Systeme in Frage:

- das man–Kommando, das zur Anzeige der klassischen Manual–Pages benutzt wird. Zu jedem Kommando gibt es solche "manpages" (so sollte es zumindest sein). Die werden angezeigt, indem das man–Kommando mit dem fraglichen Kommandonamen als Argument aufgerufen wird. Unter der grafischen Benutzeroberfläche von Linux, dem X Window System, gibt es zur Anzeige der Manual–Pages das mausgesteuerte xman–Kommando. Nähere Information zum man–Kommando gibt es auf Seite 158 des Handbuches; zu xman wird beim Programmstart automatisch ein Hilfstext angezeigt.

- das info Hilfesystem, dem die TEXinfo Dateien zugrunde liegen. Diese Hilfstexte sind so konzipiert, daß sie sowohl mit dem TEX Satzsystem formatiert und anschließend ausgedruckt werden, als auch, entsprechend aufbereitet, mit speziellen Programmen durchsucht und gelesen werden können. Als Programme kommen info bzw. sein grafisches Pendant xinfo oder der emacs Editor in Frage. Der Editor bietet einen speziellen Info–Modus, mit dem exakt die gleiche Funktionalität wie mit dem separaten info Programm erzielt wird.

[2] Unter Linux wird durch die RETURN Taste ein Zeilenvorschub (CONTROL-J) erzeugt. Der Cursor wird automatisch in die erste Spalte gesetzt. Bei MS–DOS wird zusätzlich mit dem NEWLINE noch ein Wagenrücklauf (carriage–return, CONTROL-M) erzeugt, um den Cursor explizit in die erste Spalte zu setzen. Diese Sonderzeichen tauchen als ^M auf, wenn Textdateien von DOS importiert werden.

Bei der Referenzierung der Online–Hilfen wird entweder im Falle der Manual–Pages die Sektion angegeben, also beispielsweise

**Siehe auch:**
geqn(1), gsoelim(1), grotty(1)

für die Verwandten des groff aus Sektion 1, oder es steht anstelle der Sektion das Wort "info" in den Klammern, was dann der entsprechende Verweis auf das TEXinfo–System ist.

### 3.1.1 Die 13 goldenen Regeln für ein gelungenes Kommando

Für die Syntax eines Kommandos gibt es im POSIX 1003.2 Standard dreizehn Richtlinien. Diese Regeln sind nicht zwingend, es wird aber allen Autoren neuer Utilities empfohlen, sich daran zu halten. In einem Shellscript kann mit der getopts Shellfunktion (→ Seite 89) eine nach den Regeln 3–10 eingegebene Kommandozeile analysiert werden. In C–Funktionen übernimmt getopt(3) diese Aufgabe.

1. Kommandonamen sollten zwei bis neun Zeichen lang sein.
2. Kommandonamen sollten nur aus ASCII–Kleinbuchstaben oder Ziffern bestehen.
3. Jede Optionsbezeichnung sollte aus einem einzelnen alphanumerischen Zeichen bestehen. Die Option −W soll für herstellerspezifische Erweiterungen reserviert sein.
4. Alle Optionen sollten von einem Minuszeichen '−' eingeleitet werden.
5. Optionen ohne Argumente (Schalter) sollten hinter einem einzigen Minuszeichen gruppiert werden können.
6. Jedes Optionsargument sollte von der Option, auf die es sich bezieht, durch (ein) Leerzeichen getrennt sein.
7. Ein Optionsargument sollte nicht optional sein.
8. Wenn mehrere Optionsargumente gleichzeitig erlaubt sind, sollten diese als ein einziges Kommandozeilenargument erscheinen. Dazu können die Optionsargumente entweder in Anführungszeichen eingeschlossen, oder, durch Komma getrennt, ohne Leerzeichen aufgelistet werden.
9. Alle Optionen sollten vor den Kommandoargumenten angegeben werden.
10. Zwei Minuszeichen −− sollten als Markierung für das Ende der Kommandozeilenoptionen interpretiert werden. Alle folgenden Argumente sollten als Operanden für das Kommando behandelt werden, auch wenn sie mit einem Minuszeichen beginnen. Das −− Symbol sollte nicht als Option oder Operand benutzt werden.
11. Die Reihenfolge der Optionen untereinander sollte keine Rolle spielen, es sei denn, eine Option ist als ausschließlich und dominant dokumentiert. Solche Optionen können alle vorhergehenden inkompatiblen Optionen abschalten. Wenn ein Regler (Option mit Optionsargument) wiederholt wird, sollte die Interpretation in der Reihenfolge des Auftretens erfolgen.
12. Die Reihenfolge der Kommandoargumente (Operanden) kann von Bedeutung sein, und positionsabhängige Interpretationen sind für ein Utility spezifisch.
13. Für Utilities, die als Operanden Dateien benutzen, die sie zum Lesen oder Schreiben öffnen, sollte ein einzelnes, von Blanks eingeschlossenes Minuszeichen '−' den Standardeingabekanal bezeichnen. Wenn es aus dem Zusammenhang eindeutig hervorgeht, kann auch der Standardausgabekanal so bezeichnet werden.

## 3.2 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

## 3.3 bash

### Funktion

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

### Syntax

**bash** [*Optionen*] [*Datei*]

### Beschreibung

In der Computersteinzeit wurden Programme in Lochkarten gestanzt und zusammen mit den Daten in mechanische Kartenleser gepackt. Die Auswahl zwischen den möglichen Programmen fand also nicht an einer Systemkonsole, sondern vor den Regalen des Kartenarchivs statt. Heute sind die Programme und die Daten auf magnetischen Datenträgern gespeichert. Die Festplatte einer durchschnittlichen Linux–Installation bietet mindestens 300 verschiedene Programme zur Auswahl.

Die Frage, wie die Auswahl eines konkreten Programms stattfindet, ist ebenso trivial wie bodenlos tiefgründig. Die Antwort ist ein Meta–Programm, das automatisch geladen wird und dessen Hauptaufgabe es ist, weitere Programme zu laden. Unter UNIX und seinen Verwandten wird so ein Programm als Shell bezeichnet. Sie hat, im Gegensatz zu vielen vergleichbaren Programmen anderer Betriebssysteme, den Status eines Benutzerprogramms und kann deshalb nach Belieben ausgetauscht werden.[3]

An der Benutzeroberfläche entfesselt sich leicht eine Art Glaubenskrieg zwischen den Protagonisten unterschiedlicher Modelle. Zwischen den grafischen (mausgesteuerten, bildschirmorientierten) und den textuellen (tastaturgesteuerten, zeilenorientierten) Benutzeroberflächen scheinen sich die Geister zu scheiden. Die Vorteile der grafischen Benutzerführung liegt vor allem in ihrer leichten Erlernbarkeit. Durch die Präsentation der möglichen Aktionen in Menüs kann der ungeübte Benutzer intuitiv den Weg zu seiner Problemlösung finden. Zeilenorientierte Oberflächen, sogenannte Kommandozeileninterpreter, haben ihren Vorteil in der Vielseitigkeit. Während in einem Menüsystem zwangsläufig nur die Funktionen zu erreichen sind, für die es Menüeinträge gibt, können im Kommandozeileninterpreter alle Kommandos mit allen zulässigen Optionen aufgerufen werden. Genau diese Vielseitigkeit macht den Kommandozeileninterpreter — die Shell — zu einem unverzichtbaren Werkzeug der Systemverwalterin, das auch in kommerziellen Systemen mit menügesteuerter Administratorshell nicht ersetzt werden kann.

[3] Zu diesem Zweck gibt es das chsh–Programm, mit dem jeder Systembenutzer seien Loginshell wechseln kann.

Die "Standardshell" von AT&T Unix ist die nach ihrem Entwickler Steven R. Bourne benannte Shell (die unter der Kommandobezeichnung sh aufgerufen wird). Neben ihren Diensten als interaktiver Kommandozeileninterpreter bietet die Bourne–Shell noch eine mächtige Sprache zum Erstellen von Shellprogrammen. Solche Shellscripts erlauben schnell und unkompliziert die Zusammenfassung immer wiederkehrender Kommandofolgen als Batchdatei. Die Möglichkeiten der Shellprogrammierung gehen aber noch viel weiter, wie die weiter hinten folgende Beschreibung zeigen wird.

Neben der "alten" Bourne–Shell gibt es noch eine ganze Reihe weiterer Shells. Andere bekannte Shells sind die an der Berkeley Universität entwickelte csh und die nach ihrem Entwickler David Korn benannte Shell ksh. Die Shells unterscheiden sich vor allem in den Shellsprachen, sie haben aber auch neue Eigenschaften zur Erleichterung der interaktiven Benutzung als Kommandozeileninterpreter. Die "Bourne Again Shell" bash vereint die altbekannte und bewährte Shellsprache der Bourne Shell mit den fortschrittlichen und beliebten Interaktionsfunktionen der anderen Shells. Die bash ist als Standardshell in allen Linux–Distributionen enthalten.

Es ist erklärter Anspruch der bash, zur Standard–Bourne–Shell kompatibel zu sein, und Ziel ist die volle Übereinstimmung mit dem POSIX–1003.2–Standard. Die Kompatibilität zur Bourne–Shell ist besonders wichtig, um die vielen Shellscripts für diese Standardshell auch mit der bash benutzen zu können. Bei der bash–1.12 gibt es einen Fehler in der case Anweisung, die zur Inkompatibilität führt. Dieser Fehler ist in Version 1.13 behoben.

## 3.3.1 Interaktive Shell und Shellprogrammierung

Die alltägliche Arbeit mit der Shell findet interaktiv statt. Das heißt, die Shell gibt eine Eingabeaufforderung (Prompt) aus, die vom Benutzer mit einer Folge von Tastatureingaben, der Kommandozeile, beantwortet wird. Eine Kommandozeile wird durch ein Zeilenende–Zeichen (RETURN) abgeschlossen. Die Shell interpretiert daraufhin die eingegebene Zeile und führt die gegebenenfalls darin formulierten Kommandos aus. Normalerweise gibt die Shell nach der vollständigen Bearbeitung der Kommandozeile wieder eine Eingabeaufforderung aus, um den gleichen Vorgang erneut einzuleiten.

Für die Shell macht es wenig Unterschied, ob sie eine Kommandozeile direkt von der Tastatur liest, oder ob sie die gleiche Zeile aus einem "eingefrorenen Datenstrom", einer Datei, erhält. Für den Anwender ist es enorm praktisch, immer wiederkehrende Kommandofolgen in einer Textdatei zusammenzufassen und dann diese Datei anstelle der Tastatureingabe bearbeiten zu lassen. Genau das ist der Ursprung der Shellprogrammierung.

In der Realität bieten alle Unix–Shells Programmiersprachen, die weit über diese Stapelverarbeitung hinausgehen. Es können Variable benutzt werden, Verzweigungen und Schleifen sind möglich, es können sogar Script–Funktionen definiert werden, die eine strukturierte Programmierung wie in den bekannten Hochsprachen erlauben.

Nun besteht umgekehrt kein vernünftiger Grund, weshalb die Elemente der Shellprogrammierung nicht auch auf der Kommandozeile verwendet werden können. Natürlich wird niemand ein zig–zeiliges Shellprogramm direkt auf der Kommandozeile eingeben. Andererseits macht es wenig Sinn, für jede vierzeilige for–Schleife ein Shellscript zu schreiben.

Damit wird klar, daß die Grenzen zwischen interaktiver Benutzung und Shellprogrammierung fließend sind. Je nach Blickwinkel werden aber die Schwerpunkte anders gesetzt. Diese Beschreibung richtet sich in erster Linie an interaktive Benutzer. Sie erhebt aber auch den Anspruch auf Vollständigkeit. Um den Text einigermaßen lesbar zu machen, sind die Abschnitte mit ausgesprochen shellscriptspezifischen Inhalten in einem kleineren Schriftgrad gedruckt. Solche Textteile können beim ersten Lesen übersprungen werden.

## 3.3.2 Der Kommandozeileneditor

Alle Tastatureingaben, die zeilenweise erfolgen, können mit bestimmten Steuerzeichen "editiert" werden. Mit dem Steuerzeichen CONTROL-U kann normalerweise die komplette Zeile gelöscht werden, CONTROL-C bricht die Eingabe ab, CONTROL-D[4] steht für das Dateiende (EOF, End Of File), also für das Ende der interaktiven Eingabe insgesamt, CONTROL-H oder BACKSPACE löscht das letzte Zeichen.

[4] Das Verhalten der Shell bei Eingabe von CONTROL-D kann mit der Shellvariablen ignoreeof eingestellt werden.

Diese primitiven Funktionen werden vom Terminaltreiber des Kernels im "cooked"–Modus direkt angeboten. Die bash bietet darüberhinaus einen Kommandozeileneditor, der in Umfang und Funktion den Standardeditoren vi und emacs nachempfunden ist.[5] Die Tastaturbelegung des Kommandozeileneditors kann mit dem set Shellkommando (→ Seite 92) zwischen den von beiden genannten Unix–Editoren bekannten Standardbelegungen umgeschaltet werden. Normalerweise arbeitet der Kommandozeileneditor im emacs Modus.

Die Editorbefehle im emacs Modus 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 das nicht der Fall ist, kann die ESCAPE–Taste (ESC) benutzt werden. Während CONTROL und ALT gemeinsam mit dem Buchstaben gedrückt werden müssen, wird ESC vor dem Buchstaben gedrückt (zwei Anschläge).[6]

In den Erklärungen wird CONTROL mit '**C-**' und die Metataste mit '**M-**' gekennzeichnet. In manchen Fällen müssen die CONTROL und ALT zusammen gedrückt werden, das wird dann durch **M-C-** symbolisiert.

## 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 Wörtern 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 Wörtern 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.

## Automatische Erweiterung von Kommando- und Dateinamen

**erweitern (TAB)** versucht die bis zu diesem 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 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 Wörter eines einfachen Kommandos werden dagegen nur noch mit Datei- und Verzeichnisnamen im aktuellen Verzeichnis (Arbeitsverzeichnis) verglichen. In einigen Fällen (z. B. bei Pipelines) werden auch die ersten Wörter der folgenden Kommandos auf diese Weise richtig zugeordnet.

Probieren Sie diese Funktion des Kommandozeileneditors einfach aus. Sie werden schnell merken, wie mächtig und vielseitig sie ist.

[5] Der Kommandozeileneditor kann durch Angabe der Option `-nolineediting` in der Kommandozeile beim Aufruf der Shell abgeschaltet werden.

[6] Die Funktion der ALT Tasten bei den ASCII Terminals der Console wird beim Übersetzen des Kernels festgelegt. Das Verhalten in der grafischen X11 Oberfläche wird in der Datei .Xmodmap konfiguriert.

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 TAB gedrückt wird, werden alle erkannten Möglichkeiten angezeigt.[7]

Die Kommandobezeichnung 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 Kommandobezeichnung ist possible-completions.

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

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

**Benutzernamenerweiterung (M-~)** versucht, das Wort bis zur Einfügemarke zu einem Benutzernamen zu erweitern. Die Kommandobezeichnung ist complete-username.

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

**Variablenerweiterung (M-$)** versucht das Wort bis zur Einfügemarke zu einem Variablennamen zu erweitern. Die Kommandobezeichnung ist complete-variable.

**mögliche Variable (C-x $)** zeigt alle möglichen Variablennamen an, auf die das Wort bis zur Einfügemarke paßt. Die Kommandobezeichnung 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 Kommandobezeichnung ist complete-hostname.

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

## Änderungen des Textes

**Zeichen löschen (C-d)** löscht das Zeichen unter dem Cursor. Wenn CONTROL-d als erstes Zeichen einer leeren Kommandozeile eingegeben wird, erzeugt diese Tastenkombination ein EOF Zeichen für das Ende der interaktiven Eingabe.[8] Die Kommandobezeichnung ist delete-char.

**letztes Zeichen löschen (BACKSPACE)** löscht das Zeichen vor dem Cursor. Die Kommandobezeichnung ist backward-delete-char.

**wörlich einfügen (C-q, C-v)** wird benutzt um CONTROL–Zeichen in die Kommandozeile zu schreiben, die normalerweise durch den Editor abgefangen und bearbeitet werden. Die Kommandobezeichnung ist quoted-insert.

**Tabulator einfügen (M-TAB)** schreibt ein TAB in die Kommandozeile. Die Kommandobezeichnung 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 eines normalen Buchstabens. Die Kommandobezeichnung ist self-insert.

---

[7] Wenn eine bestimmte Anzahl überschritten wird (normalerweise 100), muß die Anzeige aller Möglichkeiten nochmal bestätigt werden. Die Zahl wird in der readline–Variablen `completion-query-items` (→ Seite 63) festgelegt.

[8] Die Interpretation von EOF kann durch die Shellvariable IGNOREEOF (bzw. ignoreeof) beeinflußt werden.

**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 Kommandobezeichnung ist transpose-chars.

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

**GROSSBUCHSTABEN (M-u)** wandelt alle Buchstaben des Wortes von der Einfügemarke an in Großbuchstaben um. Die Kommandobezeichnung ist upcase-word.

**kleinbuchstaben (M-l)** wandelt alle Buchstaben des Wortes von der Einfügemarke an in Kleinbuchstaben um. Die Kommandobezeichnung 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 Kommandobezeichnung ist capitalize-word.

## Ausschneiden und Einfügen

**bis zum Zeilenende ausschneiden (C-k)** löscht die komplette Kommandozeile von der Einfügemarke an und speichert sie im Ausschneide–Ringspeicher. Die Kommandobezeichnung ist kill-line.

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

**Anfang des Wortes ausschneiden (M-BACKSPACE)** schneidet alle Zeichen vor der Einfügemarke bis zum nächsten Sonderzeichen aus und speichert sie 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 Kommandobezeichnung 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 Kommandobezeichnung ist unix-line-discard.

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

**ausgeschnittenes Stück einfügen (C-y)** fügt das zuletzt in den Ausschneide–Ringspeicher geschriebene Stück vor der Einfügemarke ein. Das Stück wird dabei aus dem Ringspeicher entfernt. Die Kommandobezeichnung ist yank.

**Ausschneide–Ringspeicher 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 Kommandobezeichnung 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 Kommandobezeichnung ist digit-argument.

**universelles Argument (nicht Belegt)** entspricht dem gleichnamigen Tastaturbefehl vom emacs–Editor. Es ist allerdings keiner Tastenkombination zugeordnet. Die Kommandobezeichnung ist universal-argument.

## Verschiedenes

**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 Kommandobezeichnung 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 Kommandobezeichnung ist undo.

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

### 3.3.3 Der Kommandozeilenspeicher (history)

Die bash speichert eine gewisse Anzahl kompletter Kommandozeilen ab und erlaubt dem Benutzer, den Kommandozeilenspeicher auf zwei verschiedene Weisen anzusprechen.

Die Anzahl der gespeicherten Kommandozeilen kann mit der Shellvariablen HISTSIZE eingestellt werden. Weitere Einstellungen können mit den Variablen HISTFILE HISTFILESIZE und history_control vorgenommen werden. Die Benutzung dieser Varaiblen wird ab Seite 74 beschrieben.

#### Der Kommandozeilenspeicher im Editor

Im Gegensatz zu den Vorbildern emacs oder vi ist für den Kommandozeileneditor der Arbeitsbereich auf die aktuelle Zeile beschränkt. Absätze und Seiten existieren hier nicht. Mit dem Kommandozeilenspeicher eröffnet sich aber doch eine zweite Dimension, die das Blättern und Suchen in alten Zeilen zu einer praktischen Erweiterung des Editors macht.

**Abschluß einer Kommandozeile (RETURN)** Wie bereits gesagt, wird jede Kommandozeile mit einem Zeilenende (RETURN) abgeschlossen. Dabei ist es egal, wo sich die Einfügemarke innerhalb der Zeile gerade befindet, es wird immer die gesamte Zeile bearbeitet. Wenn sie nicht leer ist, wird sie sofort im "rohen" Zustand in den Kommandozeilenspeicher übernommen.[9] Wenn die Shellvariable history_control das Wort 'ignorespace' enthält, werden Kommandozeilen, die mit einem Leerzeichen beginnen, nicht gespeichert. Enthält die Variable das Wort 'ignoredups', werden nur die Kommandozeilen gespeichert, die sich von der zuletzt gespeicherten unterscheiden.

Wie alle Funktionen des Editors kann diese Funktion auf beliebige Tasten gelegt werden, wenn auch in diesem Fall davon abzuraten ist. Die Kommandobezeichnung ist accept-line.

**Kommando zurück (C-p)** blättert 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.

[9] Wenn die History–Substitution im C–Shell–Stil eingeschaltet ist, wird sie noch durchgeführt bevor die Zeile gespeichert wird.

**Kommando vorwärts (C-n)** blättert 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 schrittweise rückwärts im Kommandozeilenspeicher nach einer Kommandozeile mit exakt passendem Muster. Das gesuchte Muster wird interaktiv eingegeben und die erste passende Zeile sofort angezeigt. Reguläre Ausdrücke im Muster werden nicht interpretiert, sondern Buchstäblich mit den gespeicherten Kommandozeilen verglichen. Das Suchmuster kann mit BACKSPACE schrittweise zurückgenommen werden; das erste andere Editorkommando während der interaktiven Eingabe des Suchmusters beendet die Suche. Die Kommandobezeichnung ist reverse-search-history.

**Kommando vorwärts suchen (C-s)** sucht vorwärts im Kommandozeilenspeicher von der aktuellen Zeile an nach einer Kommandozeile mit einem bestimmtem Muster. Die Bearbeitung des Musters erfolgt in der gleichen Weise wie beim reverse-search-history Editorkommando. Die Kommandobezeichnung ist forward-search-history.

**Kommandozeile erweitern (M-C-e)** führt die weiter unten beschriebene History–Expansion im C–Shell–Stil und die Synonymerweiterung aus, ohne die Kommandozeile auszuführen. Die Zeile kann daraufhin weiter bearbeitet werden. 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.

### History im C–Shell–Stil

Zusätzlich zu den oben beschriebenen Tastenfunktionen des Editors kann der Kommandozeilenspeicher mit einem History–Mechanismus benutzt werden, dessen Verwendung der C–Shell nachempfunden ist. [10] Diese History–Expansion kann mit dem Shellkommando `set +H` ausgeschaltet werden.

In der C–Shell ist es möglich, einzelne Wörter aus bestimmten Kommandozeilen im Kommandozeilenspeicher in die aktuelle Kommandozeile zu integrieren. Dazu wird zuerst eine Kommandozeile referenziert 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 Ausrufeszeichen '!' die History–Substitution ein. Nur wenn das Ausrufezeichen nach einem Fluchtsymbol '\' steht oder von einem SPACE, RETURN, TAB, '=' oder '(' gefolgt wird, findet die Substitution nicht statt.

Anstelle des Ausrufezeichens kann in der Shellvariablen histchars 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.

[10] Die History–Erweiterung findet normalerweise nicht direkt im Kommandozeileneditor statt, sondern unmittelbar nach Abschluß der Zeile, noch bevor sie von der Shell in Token zerlegt wird. Die Erweiterung direkt im Editor kann mit der oben beschriebenen shell-expand-line–Funktion erzwungen werden.

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

**^*Alt*^*Neu*** ist eine Abkürzung für `!!:^Alt^Neu^` zur Ersetzung von Zeichenkette *Alt* durch *Neu* in der letzten Kommandozeile. Diese Konstruktion muß allein auf der Kommandozeile stehen.

**!#** steht für die aktuelle Kommandozeile bis hier hin.

### 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 Wörter einer Kommandozeile sind vom Zeilenanfang an mit Null beginnend numeriert.

*n* bezeichnet das *n*te Wort. Das Wort 0 ist in der Regel der Kommandoname.

^ (Caret) steht für das erste Argument (das ist Wort Nummer 1).

$ bezeichnet das letzte Argument.

**%** steht für das bei ?***Zeichenkette***? gefundene Wort.

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

***** steht für alle Argumente, also die Wörter 1-$. Wenn das Kommando nur aus einem Wort besteht, wird die leere Zeichenkette zurückgeliefert.

### Modifikation der bezogenen Kommandozeilen

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

**h** schneidet alle Zeichen nach dem letzten Slash von dem referenzierten Bereich der Kommandozeile ab. Die Funktion ist in gewisser Weise mit dem dirname–Shellutility (→ Seite 113) vergleichbar.

**r** schneidet eine Endung der Form '.xxx' nach dem letzten Punkt ab.

**e** läßt nur die Endung in der oben beschriebenen Form übrig.

**t** entfernt aus dem referenzierten Bereich der Kommandozeile alles bis zum letzten Slash. Die Funktion ist dem basename Shellutility (→ Seite 54) vergleichbar.

**p** zeigt die entstandene Kommandozeile sofort an, ohne sie auszuführen.

**[g]s/*Alt*/*Neu*[/]** ersetzt die Zeichenkette *Alt* bei ihrem ersten Auftreten durch *Neu*. Durch den Zusatz **g** kann erreicht werden, daß alle Vorkommen von *Alt* ersetzt werden. Anstelle der Slashes können beliebige andere Zeichen benutzt werden.

### Beispiele für die History–Funktion

Auf den ersten Blick ist die C–Shell History–Funktion wahrscheinlich etwas spröde. Die folgenden Beispiele zeigen einige Vorteile dieser Funktion bei der täglichen Arbeit mit der C–Shell.

Häufig wird eine Datei in ein Verzeichnis verschoben und unmittelbar anschließend das aktuelle Verzeichnis dorthin gewechselt.

```
$ mv broo.fazz.tar.gz /ftp/pub/comp/i386/Linux/utils/misc
$ cd !$
$ _
```

Mit '!$' wird das letzte Wort der vorhergehenden Kommandozeile eingefügt. Das gleiche Ergebnis wird auch durch die META-. Funktion des Kommandozeileneditors ausgeführt.

Gelegentlich braucht man nicht das komplette Argument, sondern nur einen Teil davon:

```
$ less /usr/X386/include/X11/Xaw/SmeBSBP.h
[Ausgabe von less]
$ find /usr/src/X11 -name !:t
find /usr/src/X11 -name SmeBSBP.h
find: /usr/src/X11: No such file or directory
$ _
```

Der Operator !:t liefert SmeBSBP.h. Hier kann die Referenz auf das letzte Argument weggelassen werden, weil die :t Operation nur die Zeichen nach dem letzten Slash liefert.

Sehr nützlich ist auch die schnelle Substitution einer Zeichenfolge:

```
$ tar tvfz /ftp/pub/Incoming/das.ist.neu.tar.gz
[Ausgabe von tar]
$ ^tv^x
tar tfz /ftp/pub/Incoming/das.ist.neu.tar.gz
$ _
```

Hier wird die tar–Option t durch x ausgetauscht. Zuerst wird also der Inhalt des Archivs angezeigt, dann wird es im nächsten Schritt ausgepackt. Mit dieser Methode lassen sich ebenso leicht Tippfehler in der letzten Kommandozeile verbessern.

### 3.3.4 Anpassung des Kommandozeileneditors

Der Kommandozeileneditor der bash benutzt die GNU–**readline** Bibliotheksfunktionen. Die Zuordnung von Tastenkombinationen zu bestimmten Editorfunktionen ist im Prinzip frei. Wie bereits gesagt, bietet die bash zwei Standardbelegungen an, zwischen denen mit dem set Shellkommando (→ Seite 92) umgeschaltet werden kann.

Wem diese Standardbelegungen nicht genügen, bietet readline zusätzlich die Möglichkeit, jede Funktion individuell mit einer Tastenkombination zu verknüpfen. Diese Um- bzw. Neubelegung der Tasten kann zur Laufzeit mit dem bind Shellkommando (→ Seite 85) durchgeführt werden.

Beispielsweise lassen sich die History–Funktionen im vi–Modus durch folgende Zeilen auf die Cursortasten legen:

```
$ bind -m vi \"\\e[A\":previous-history
$ bind -m vi \"\\e[B\":next-history
$ _
```

Es ist auch möglich, eine neue Tastaturbelegung dauerhaft einzurichten, indem die Zuordnung von Kommandos und Tastenkombinationen in der Datei ~/.inputrc im Heimatverzeichnis abgespeichert wird.[11]

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

***Taste*:*Kommandobezeichnung***

Die Kommandobezeichnungen sind bei den Beschreibungen angegeben.

Für die Tastenkombinationen können die Konstruktionen '\C-' für die Kombination mit CONTROL und '\e-' für die Metakombinationen mit ALT oder ESC verwendet werden. Außerdem können die folgenden Tasten benannt werden:

RUBOUT für Backspace
DEL für Delete
ESC für Escape
SPACE oder SPC für Leerzeichen
RETURN, RET, NEWLINE oder LFD für Zeilenende
TAB für Tabulator

[11] Beim Erstellen dieser Datei bietet das bind -d Kommando Hilfestellung. Eine Unterscheidung der verschiedenen Modi der Editoren, wie sie vom bind–Kommando erlaubt wird, ist in der .inputrc–Datei bei der aktuellen Version der bash nicht möglich.

Neben den Editorkommandos können auch Zeichenketten und Kommandokombinationen (Makros) auf bestimmte Tasten gelegt werden.

Zusätzlich können zur Anpassung von readline einige Schalter gesetzt und Variable belegt werden. Sie werden in der Form

`set` *Schalter Wert*

in der `.inputrc`–Datei eingetragen. Die Werte in Klammern stellen die Voreinstellung dar.

**horizontal-scroll-mode** (Off) Wird dieser Schalter auf "On" gesetzt, so wird beim Schreiben der Kommandozeile über den linken Bildschirmrand hinaus die Zeile nach links verschoben, ansonsten wird sie umgebrochen.

**editing-mode** (emacs) Durch Belegen dieser Variablen mit "vi" wird die vi Tastaturbelegung eingeschaltet. Einzige Alternative ist der voreingestellte emacs–Modus.

**mark-modified-lines** (Off) Wird dieser Schalter "On" gesetzt, werden alle Zeilen in der history mit einem Asterisk '*' gekennzeichnet, die im Editor verändert wurden.

**prefer-visible-bell** (Off) Mit diesem Schalter kann die optische Warnung (falls vorhanden) anstelle der akustischen eingeschaltet werden.

**comment-begin** (':') In dieser Variablen kann das Zeichen bestimmt werden, das durch das vi-comment Kommando in die erste Spalte der Kommandozeile geschrieben wird. Durch den Doppelpunkt (Voreinstellung) wird die Ausführung der Zeile unterdrückt.

**meta-flag** (Off) Wenn dieser Schalter "On" gesetzt wird, erlaubt die Shell 8-Bit Zeichen (zum Beispiel Umlaute) auf der Kommandozeile. Normalerweise wird das höchste Bit gekappt und damit nur ASCII–Zeichen interpretiert.

**convert-meta** (On) Bleibt dieser Schalter gesetzt, werden Zeichen mit gesetztem achten Bit zu einer ESC–Sequenz mit dem entsprechenden ASCII–Zeichen verarbeitet.

**completion-query-items** (100) In dieser Variablen kann eine Anzahl bestimmt werden, bis zu der mögliche Erweiterungen ohne Nachfrage angezeigt werden.

Eine ausführliche TEXinfo–Beschreibung aller readline–Funktionen kann mit dem info–Kommando nachgelesen werden.

Die aktuelle Tastaturbelegung wird mit dem Kommando '`bind -v`' ausgegeben (→ bind auf Seite 85).

#### Der erweiterte Editor: sekundärer Prompt

Viele Operationen der Shell werden mit bestimmten Symbolen eingeleitet und müssen mit weiteren Symbolen abgeschlossen werden. Beispielsweise muß eine for–Schleife durch ein done abgeschlossen werden, eine ( Klammer durch eine ), eine mit Anführungszeichen begonnene Zeichenkette muß mit Anführungszeichen abgeschlossen werden und so weiter…

Die Shell unterstützt die Eingabe solcher Kommandos, indem sie nach einem Zeilenende in einer unvollständigen Kommandozeile einen neuen Prompt — in der Regel ein > — ausgibt, bis der erwartete Abschluß des begonnenen Kommandos eingegeben ist. Die so eingegebenen Teile werden zusammen als eine Kommandozeile interpretiert.

Wenn die Shellvariable command_oriented_history gesetzt ist, wird ein aus mehreren Zeilen bestehendes Kommando als ein einziger Eintrag im History–Speicher behandelt.

### 3.3.5 Interpretation der Kommandozeile

Mit dem accept-line Kommando (RETURN) im Editor ist der im eigentlichen Sinn interaktive Teil der Shellbenutzung zu Ende. Die folgende Interpretation der Zeile durch die Shell führt normalerweise zur Erzeugung eines neuen Prozesses, in dem das eben aufgerufene Programm abläuft. Bevor das geschieht, werden aber von der Shell selbst noch eine ganze Reihe von Veränderungen an der Zeile durchgeführt. Diese Interpretation findet bei der interaktiven Shell auf exakt die gleiche Weise statt wie im Shellprogramm.

Die Kommandozeile wird analysiert, indem sie anhand von "Trennzeichen" in atomare Sinneinheiten (Wörter oder Token) geteilt wird. Dabei kann die gesamte Zeile in mehrere Kommandos zerfallen, die getrennt weiterverarbeitet werden.

Die Wörter der einzelnen Kommandos werden sequentiell (der Reihe nach) auf bestimmte Symbole oder Sonderzeichen durchsucht. Bestimmte Sonderzeichen führen zur Umlenkung der Ein- und Ausgabekanäle. Durch andere Symbole werden bestimmte Wörter verändert oder ersetzt. Dieser Vorgang wird als Parameter–Substitution bezeichnet. Danach wird die Kommandozeile noch "aufgeräumt". Abschließend werden die Programme (intern oder extern) lokalisiert und mit ihren Optionen und Argumenten aufgerufen.

### 3.3.6 Kommentare

In Shellscripten können an beliebigen Stellen Kommentare durch ein Nummern- oder Hash–Zeichen '#' eingeleitet werden. Der gesamte Zeilenrest wird bei der Interpretation von der Shell ignoriert.

Im Unterschied zur Bourne–Shell bietet die interaktive `bash` diese Möglichkeit normalerweise nicht auf der Kommandozeile (wohl aber beim `source` Shellkommando). Durch das Kommando `set -o interactive-comments` kann die Verwendung von Kommentaren in der interaktiven Shell ermöglicht werden.

Auf der Kommandozeile (wie auch im Shellscript) werden Zeilen, die mit einem Doppelpunkt beginnen, zwar der Parameter–Erweiterung unterworfen, sie werden aber nicht ausgeführt.

Ein häufiger Spezialfall tritt auf, wenn auf ein Kommentarzeichen in der ersten Zeile ein Ausrufezeichen folgt und auf dem Rest der ersten Zeile ein Interpreterprogramm benannt ist. Wenn so eine Textdatei ausführbar ist und anstelle eines Kommandos aufgerufen wird, übergibt der Linux–Kernel die Textdatei direkt dem benannten Interpreterprogramm, ohne extra eine Shell aufzurufen. Natürlich kann auch eine Shell als Interpreterprogramm für ein Shellscript benannt werden. Das folgende Script ersetzt beispielsweise die fehlende `pwd` Funktion der `tcsh`:

```
#!/bin/bash
IFS=" "
builtin pwd
```

### 3.3.7 Der Status

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.

Anders als z.B. in C–Funktionen gilt in der `bash` (wie auch in den anderen Shells) folgende Konvention:

- Der Status Null bedeutet, daß bei der Programmausführung kein Fehler aufgetreten ist. Dieser Status wird auch als "wahr" interpretiert.
- Jeder von Null verschiedene Status wird in Shellprogrammen als "falsch" interpretiert. Ob für so einen Status tatsächlich ein Fehler bei der Ausführung des Kommandos verantwortlich ist, hängt von dem speziellen Kommando ab.

  Häufig kann aus dem Statuswert auf die Art des Fehlers zurückgeschlossen werden. Wenn ein Kommando beispielsweise durch ein Signal beendet wurde, ist der Rückgabewert (Status) der Wert des Signals + 128.

  Der Status kann in sein logisches Gegenteil verkehrt werden (0 oder 1), indem dem gesamten Kommando ein '!␣' vorangestellt wird. Um diese Konstruktion von der History–Substitution im C–Shell–Stil zu unterscheiden, muß dem Ausrufezeichen unmittelbar ein Blank oder '(' folgen.

  In der Shell kann der Status des zuletzt ausgeführten Kommandos aus der Shellvariablen ? gelesen werden.

### 3.3.8 Shell Grammatik

#### Reservierte Wörter

Wie bei jeder anderen Programmiersprache gibt es für die Sprache zur Shellprogrammierung reservierte Wörter. Diese Wörter 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 Wörter sind reserviert:
! case do done elif else esac fi for function if in then until while { }

### Atome, Wörter, Token

Die "Atome" einer Kommandozeile — auch als Wörter oder Token bezeichnet — werden anhand bestimmter Trennzeichen identifiziert. Die einfachste und natürliche Trennung zweier Wörter findet durch Leerzeichen statt. Dabei ist die Anzahl der Trennzeichen gleichgültig. Anstelle von Leerzeichen können auch TABs benutzt werden. Wegen ihrer offenkundigen Ähnlichkeit werden diese Trenner auch als **Blank**s bezeichnet.

Die folgenden zur Shellprogrammierung verwendeten **Sonderzeichen** führen automatisch auch zur Trennung von Wörtern:

| & ; ( ) < > und das Zeilenende (RETURN)

Diese Zeichen haben in jedem Fall spezielle Aufgaben und können deshalb nicht einfach innerhalb von Argumenten an ein Kommando übergeben werden. Eine Reihe weiterer Zeichen werden nur in bestimmten Situationen als Operatoren erkannt. In diesen Fällen können auch sie nicht innerhalb von Argumenten an ein Kommando weitergegeben werden. Besondere Aufmerksamkeit ist bei den folgenden Zeichen geboten:

! * ? $ ` ' { } [ ] ∧ = # ` " \

Wenn eines der oben genannten (Sonder-) Zeichen als Argument an ein Kommando übergeben werden soll, muß es für die Shell "entwertet" werden.

### Kommandos — nicht unbedingt einfach

In einem Multitasking–Betriebssystem macht es durchaus Sinn, mehr als ein Kommando in einer Zeile aufzurufen. So ein zusammengesetztes Kommando besteht aus mehreren einfachen Kommandos. Zur Trennung von einfachen Kommandos werden Kontroll–Operatoren verwendet, die aus den oben genannten Sonderzeichen zusammengesetzt sind:

| & || && ; ;; ( )

Außer mit dem eigentlichen Kommandonamen (dem Namen der ausführbaren Datei) kann ein einfaches Kommando auch mit einer Zuweisung an eine Shellvariable beginnen.

Für jedes einfache Kommando können die offenen Datenkanäle umgeleitet, sowie neue erzeugt werden.

## 3.3.9 Quotierung

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

Es gibt drei Formen der Quotierung:

1. durch das Fluchtsymbol \ (Backslash)

2. durch Hochkomma ' (Quote)

3. durch Anführungszeichen " (Doublequote)

Das Fluchtsymbol "entwertet" das unmittelbar folgende Sonderzeichen. Ein durch das Fluchtsymbol entwertetes Zeilenende wird ignoriert.

Die in Hochkommata eingeschlossenen Wörter werden von der Shell nicht weiter bearbeitet. Lediglich ein Hochkomma darf nicht in Hochkommata eingeschlossen werden; auch nicht wenn, es durch ein Fluchtsymbol eingeleitet wird.

Von den in Anführungszeichen eingeschlossenen Wörtern erkennt die Shell nur die Sonderzeichen $, ' und \ als solche. Alle anderen Wörter bleiben unbearbeitet. Das Fluchtsymbol 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 Fluchtsymbol eingeleitet wird.

Die speziellen Parameter * und @ haben eine besondere Bedeutung, wenn sie zwischen Anführungszeichen auftauchen (→ Seite 77).

### 3.3.10 Ein-/Ausgabe–Umleitung

Jedes Programm erhält automatisch beim Start drei offene "Datenkanäle": die Standardeingabe, die Standardausgabe und die Standardfehlerausgabe.

Bevor eine Kommandozeile ausgeführt wird, können die bestehenden Eingabe- und Ausgabekanäle umgelenkt, sowie neue erzeugt werden. Auf diese Weise können Dateien im Dateisystem zum Lesen bzw. Schreiben für das Kommando geöffnet werden, die nach dessen Beendigung automatisch wieder geschlossen werden. Beispielsweise kann so die Ausgabe des ls–Kommandos zur weiteren Bearbeitung in eine Datei geschrieben werden.

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 (→ exec, Seite 88).

Wenn in einem Kommando mehrere Umleitungen gelegt werden, ist die Reihenfolge signifikant. Ein Beispiel hierfür ist bei der Verdopplung der Dateikennung auf Seite 67 gegeben.

#### Eingabeumleitung

[*n*]< ***Wort***

erzeugt entweder 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 Shellvariable 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 Shellscripts direkt als Standardeingabe für ein Kommando zu benutzen. Es werden alle Zeilen des Dokumentes gelesen und als Eingabe an das Kommando weitergeleitet, bis eine Zeile auftaucht, die nur den Begrenzer enthält. 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 Fluchtsymbol '\' eingeleitetes Zeilenende ignoriert. Um das Fluchtsymbol selbst, sowie die Zeichen $ und ' im Dokument darstellen zu können, müssen sie ebenfalls durch ein Fluchtsymbol 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.

### Verdopplung 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 Verdopplung der Eingabekennung.

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

Die Konstruktion

```
$ ls /usr/local/foo > inhalt 2>&1
$ cat inhalt
ls: /usr/local/foo: No such file or directory
$ _
```

lenkt die Standardausgabe und die Standardfehlerausgabe in die Datei `inhalt` um.

Die Konstruktion

```
$ ls /usr/local/foo 2>&1 > inhalt
ls: /usr/local/foo: No such file or directory
$ cat inhalt
$ _
```

lenkt dagegen nur die Standardausgabe in die Datei `inhalt` um.

Beim ersten Beispiel wird zuerst die Standardausgabe in die Datei umgelenkt, 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.

### 3.3.11 Pipelines

Die engste Verknüpfung mehrerer einfacher Kommandos wird durch eine sogenannte Pipeline hergestellt. Das Charakteristikum einer Pipeline ist die Zusammenlegung des Standardausgabekanals des einen Kommandos mit dem Standardeingabekanal des zweiten. Dabei wird die Multitasking–Fähigkeit von Linux ausgenutzt und die beiden (oder mehr) Kommandos als separate Prozesse gleichzeitig gestartet.

Die Syntax für die Zusammenfassung zweier Kommandos in einer Pipeline sieht folgendermaßen aus:

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

Der entscheidende Kontrolloperator ist das '|' Zeichen (Pipe).

> Der Status der gesamten Pipeline ist gleich dem Status des letzten Kommandos in der Pipeline (→ Seite 64).
>
> Die Verknüpfung der Standardkanäle findet vor einer eventuellen Umlenkung durch eines der Kommandos statt. Das bedeutet, daß die Zusammenlegung von Standardfehlerausgabe und Standardausgabe tatsächlich in der Pipeline mündet.
>
> Die parallele Ausführung aller einfachen Kommandos einer Pipeline legen es nahe, die komplette Pipeline syntaktisch als Einheit zu betrachten. Wenn im folgenden nicht ausdrücklich von einfachen Kommandos die Rede ist, steht "Kommando" auch für Pipelines.
>
> Es können mehrere Kommandos durch Pipelines verkettet werden. Dabei sind aber nur zwei Pipelines für jedes einfache Kommando erlaubt — jeweils eine für die Standardeingabe und die Standardausgabe. Soll ein Kommando aus mehreren Pipelines gleichzeitig lesen, kann das ab bash–1.13 durch die Prozeß–Substitution (→ Seite 81) mit Named–Pipes realisiert werden.

### 3.3.12 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 (→ Seiten 85, 89 und 90).

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 '`fg %1`' den angehaltenen oder im Hintergrund laufenden Prozeß mit der Jobnummer 1 im Vordergrund zum Laufen.

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 entweder 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.

## 3.3.13 Listen

Neben der Möglichkeit, zwei oder mehr einfache Kommandos in einer Pipeline parallel aufzurufen, erlaubt die Shell weitere Aufrufe mehrerer Kommandos in einer einzigen Zeile. Diese Aufrufe werden als Listen bezeichnet.

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

### Listen mit ; und &

Ein Semikolon kann zum Abschluß eines Kommandos benutzt werden. Auf diese Weise können beliebige Kommandos zu einer Liste zusammengefügt werden, indem sie, durch Semikolon getrennt, auf einer Zeile eingegeben werden. Die Ausführung der beiden Kommandos geschieht unabhängig, nacheinander.

Mit dem & Operator wird ebenfalls ein Kommando abgeschlossen. Auch mit diesem Operator können also zwei Kommandos auf einer Zeile getrennt werden. Jedes mit einem & abgeschlossene Kommando wird unabhängig von den anderen Kommandos der gleichen Zeile im Hintergrund ausgeführt. Es können auf diese Weise also zwei oder mehr Kommandos parallel aufgerufen werden. Diese Kommandos stehen während der Ausführung nicht in Verbindung zueinander. Wenn alle Kommandos einer Zeile durch ein & abgeschlossen werden, erscheint bei der interaktiven Shell sofort die nächste Eingabeaufforderung.

### Bedingte Ausführung

Die Kontrolloperatoren && und || verknüpfen zwei Kommandos logisch miteinander. Das zweite Kommando dieser Liste wird in Abhängigkeit vom Ergebnis (Status) des ersten ausgeführt.

Bei der Verknüpfung durch

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

wird das *Kommando2* nur dann ausgeführt, wenn *Kommando1* fehlerfrei abgeschlossen wurde. Man kann das als eine logische UND–Verknüpfung der beiden Kommandos betrachten. Der Status der UND–Liste ist wahr (null), wenn beide Kommandos "wahr" sind. Wenn bereits das erste Kommando Null liefert, wird das zweite zur Bewertung des Gesamtausdrucks nicht mehr benötigt, also wird es auch nicht ausgeführt.

Das Pendant zur UND–Liste ist die ODER–Verknüpfung durch

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

Hier wird das *Kommando2* nur dann ausgeführt, wenn bei der Bearbeitung vom *Kommando1* ein Fehler aufgetreten ist. Analog zur UND–Verknüpfung kann auch die Verkettung durch || als logische ODER–Verknüpfung betrachtet werden. Der Status der ODER–Liste ist wahr, wenn das erste oder das zweite Kommando einen Status Null liefert. Wenn das bereits beim ersten Kommando erfüllt ist, wird das zweite nicht mehr ausgeführt.[12]

[12]Der Status einer Liste ist immer das Ergebnis des zuletzt ausgeführten Kommandos. Aus diesem Grund wird das zweite

### 3.3.14 Gruppen und Kontrollstrukturen: Blöcke, Schleifen, Verzweigungen, Funktionen

Einzelne Kommandos oder Listen können auf verschiedene Weisen weiter zu Gruppen zusammengefaßt werden:

**{*Liste*;}** Mit den geschweiften Klammern werden die Kommandos der Liste zu einer einfachen Gruppe zusammengefaßt. Diese Klammerung entspricht der in mathematischen Ausdrücken und dient der Assoziierung nieder- oder gleichrangig verknüpfter Kommandos.

Die Operatoren zur Shellprogrammierung unterscheiden sich in Rang und Assoziativität. Prinzipiell wird eine Kommandozeile von links nach rechts abgearbeitet. Die Zusammenfassung in UND- oder ODER–Listen hat gegenüber der Auflistung mit Semikolon oder & Vorrang. Die Zusammenfassung einfacher Kommandos in Pipelines hat wiederum größere Assoziativität als die Listen. Die beiden Zeilen

```
{ cat /etc/gettydefs || cat /etc/gettytab; }| grep 38400
cat /etc/gettydefs || cat /etc/gettytab | grep 38400
```

unterscheiden sich syntaktisch nur in der Klammerung, die größere Assoziativität der Pipeline führt in der zweiten Zeile aber zu einer impliziten Klammerung der beiden letzten Kommandos. Die erste Zeile schreibt die erste existierende der beiden angegebenen Dateien in den Standardausgabekanal, der durch eine Pipeline dem `grep`–Kommando zur Auswertung übergeben wird. Die zweite Zeile schreibt entweder die Datei `/etc/gettydefs` auf den Bildschirm (Standardausgabe), oder sie übergibt die Datei `/etc/gettytab` dem `grep`–Kommando.

Die geschweiften Klammern trennen selbst keine Kommandos (siehe oben). Aus diesem Grund ist ein Leerzeichen zwischen der einleitenden Klammer und dem ersten Kommando der eingeschlossenen Liste notwendig. Um die Trennung zum ersten Kommando nach der abschließenden Klammer eindeutig anzugeben, ist es sinnvoll, die eingeklammerte Liste immer mit einem Semikolon abzuschließen. Im Gegensatz zur Bourne–Shell wird in dem Beispiel oben das Kommando korrekt interpretiert, wenn das Semikolon weggelassen wird, weil die der Klammer folgende Pipeline die Kommandos trennt.

Im Unterschied zur Bourne–Shell wird bei der `bash` auch in dem Beispiel oder bei einer Ausgabeumlenkung für die komplette Klammer keine Subshell erzeugt.

**(*Liste*)** Durch runde Klammerung werden die Kommandos einer *Liste* in einer eigenen (Shell-) Umgebung ausgeführt. Wenn von den Kommandos der Liste Veränderungen an der Shellumgebung vorgenommen werden, sind diese außerhalb der eingeklammerten Gruppe nicht sichtbar. Wenn zum Beispiel innerhalb einer so geklammerten Gruppe das Verzeichnis gewechselt wird, beziehen sich alle weiteren Kommandos innerhalb der Gruppe auf dieses Verzeichnis; außerhalb der Klammerung bleibt das "alte" Verzeichnis aktuell. Beispiel:

```
cd /
$ (cd /usr/bin; ls e*); ls -F
egrep    elvis    elvprsv  elvrec   env      ex       expand
bin/        etc/        lost+found/  root/       usr/
boot/       home/       mnt/         sbin/       var/
dev/        lib/        proc/        tmp/        vmlinuz
$ _
```

erzeugt eine Subshell, in der aus dem aktuellen Wurzelverzeichnis in das Verzeichnis `/usr/bin` gewechselt wird. Hier können beispielsweise alle Dateien angezeigt werden, deren Name mit einem 'e' beginnt. Mit den schließenden Klammern wird auch die Subshell beendet. Das darauffolgende Listing findet wieder im ursprünglich aktuellen Verzeichnis statt.

---

Kommando einer ODER–Verknüpfung auf keinen Fall mehr ausgeführt, weil sonst das Ergebnis des Gesamtausdrucks von diesem zweiten Kommando abhängen würde.

**for *Name* [in *Wort*] do *Liste* done** Mit dieser Konstruktion stellt die bash die von vielen Programmiersprachen bekannte und bewährte for–Schleife bereit. Mit *Name* wird eine Shellvariable definiert, die in jedem Schleifendurchlauf einen neuen Wert erhält. Die Anzahl der Schleifendurchläufe entspricht der Anzahl der vorhandenen Werte.

Die Werte werden normalerweise nach dem Schlüsselwort in übergeben. Dazu können mehrere *Wörter* angegeben werden, die von der Shell erweitert werden (→ Seite 78). Für jeden Durchlauf wird ein Token in der Variablen *Name* übergeben.

Wenn der 'in *Wort*' Teil fehlt, wird die Liste für jeden gesetzten Positionsparameter (→ Seite 77) einmal ausgeführt.

Die interaktive Eingabe einer for–Schleife wird (wie die Eingabe jedes anderen aus mehreren Teilen bestehenden Konstrukts) von der Shell unterstützt, indem sie einen sekundären Prompt (PS2) zur Eingabe einer Schleife über mehrere Zeilen anbietet. Dieser sekundäre Prompt wird ausgegeben, solange die Schleife nicht mit done abgeschlossen ist.

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

In dem folgenden Beispiel werden im aktuellen Verzeichnis alle Dateien mit der Endung 'foo' umbenannt, so daß sie auf 'bar' enden.

```
$ for i in *.foo; do
> base=‘basename $i .foo‘
> mv $i $base.bar
> done
$ _
```

Das basename–Kommando ist auf Seite 54 beschrieben. Die Zuweisung der Ausgabe dieses Kommandos an die Variable base findet durch "Kommandosubstitution" statt. Diese Shelloperation wird auf Seite 80 beschrieben. Diese Operation wird duch die "Backquotes" ausgelöst, nicht durch Hochkomma.

Die bash bietet mit ihrer Parametererweiterung eine andere schöne Methode, die Endung vom Dateinamen zu trennen: mit der Konstruktion `base=${i%.foo}` (→ Seite 80). Das spart den Aufruf eines externen Kommandos und verbessert damit die Performance.

Wenn eine Variable innerhalb einer for–Schleife verändert wird, ist der neue Wert auch außerhalb der Schleife sichtbar. Die bash verhält sich damit POSIX 1003.2 konform. Eine Ausnahme kann für die Zählvariable erreicht werden, indem das Kommando `set -l` gegeben wird. In diesem Fall wird die Zählvariable nach dem letzten Durchlauf auf den Wert vor Beginn der Schleife zurückgesetzt.

**while *Liste* do *Liste* done** Von anderen Programmiersprachen ebenso bekannt ist die while ... do ... done Schleife. Hier wird der Schleifenkörper 'do *Liste* done' so lange wiederholt, bis die in 'while *Liste*' formulierte Bedingung falsch ist. Das ist der Fall, wenn das letzte Kommando der while *Liste* einen Status ungleich null liefert.

Bei der interaktiven Eingabe einer while–Schleife wird so lange der sekundäre Prompt (PS2) ausgegeben, bis die Schleife mit done abgeschlossen ist.

Der Status der while–Schleife ist gleich dem Status des letzten Kommandos des 'do–Teils' oder Null wenn kein Kommando ausgeführt wurde.

Im folgenden Beispiel wird die while–Schleife benutzt, um die Unterverzeichnisse z. B. des `/usr/local/man` Verzeichnisses anzulegen.

```
$ declare -i zahl=1
$ while [ $zahl -lt 10 ]
> do
> mkdir man$zahl
> mkdir cat$zahl
> zahl=$[zahl+1]
> done
$ _
```

Die Deklaration von `zahl` als Integer–Variable ist an dieser Stelle nicht notwendig, weil die Zuweisung einer Zahl und die Verwendung in arithmetischen Ausdrücken diese Interpretation implizieren.

Zur Inkrementierung der Zählvariablen wird die Arithmetische Erweiterung benutzt, die auf Seite 80 beschrieben ist. Die bash erlaubt ab Version 1.13 auch die direkte Inkrementierung einer Integer–Variablen durch den '+=' Zuweisungsoperator in einer arithmetischen Erweiterung oder mit dem let Shellkommando. Beide Varianten sind in der Standard–Bourne–Shell nicht vorgesehen.

**until** ***Liste*** **do** ***Liste*** **done** Die until–Schleife entspricht der while–Schleife mit dem Unterschied, daß der do–Teil ausgeführt wird, so lange wie das letzte Kommando der 'until *Liste*' einen Status ungleich Null liefert.

**if** ***Liste*** **then** ***Liste*** [**elif** ***Liste*** **then** ***Liste*** ...][**else** ***Liste***] **fi** Mit der if–Konstruktion werden die Kommandos der then ***Liste*** unter der Bedingung ausgeführt, daß das letzte Kommando der if ***Liste*** "wahr" ist, also einen Status null liefert.

Mit der optionalen elif–Konstruktion können beliebig lange if–else Ketten erzeugt werden. Wenn der 'elif *Liste*' Teil des letzten elif nicht Null liefert, wird der abschließende else Teil bearbeitet.

Wie bei den Schleifen wird auch bei der interaktiven Eingabe einer if–Anweisung der sekundäre Prompt ausgegeben, bis die Konstruktion mit einem fi abgeschlossen ist.

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

Das folgende Beispiel zeigt, wie in der Initialisierungsdatei `~/.bashrc` zwischen einer Shell im xterm und den Textbildschirmen unterschieden werden kann.

```
if [ $WINDOWID ]; then
   TERM=xterm
   export XDVIFONTS=/usr/TeX/lib/tex/fonts/%f.%d%p
   export OPENWINHOME=/usr/openwin
   export PAGER=/usr/X386/bin/xless
else
   export PAGER=/usr/bin/less
fi
```

Ein weiteres Beispiel zeigt, wie in einem Shellscript unterschieden werden kann, ob die aufrufende Shell interaktiv arbeitet oder nicht.

```
if [ "${-#*i}" = "$-" ]; then
   echo Die Shell arbeitet nicht interaktiv
else
   echo Die Shell arbeitet interaktiv
fi
```

Der in diesem Beispiel benutzte Ausdruck `["${-#*i}" = "$-" ]` ist ein schönes Beispiel für die Parametererweiterung, wie sie auf Seite 80 erklärt ist. Im Spezialparameter - sind die Optionsflags der Shell gespeichert. Durch die "Erweiterung" `${-#*i}` wird aus dieser Zeichenkette ein 'i' (und alle Zeichen davor) entfernt, wenn es enthalten ist. Der umschließende test (→ Seite 94) vergleicht die so eventuell verkürzte Zeichenkette mit dem Original. Wenn sie gleich sind, ist die 'i' Option der interaktiven Shell nicht gesetzt.

Die Anführungszeichen sind in dem Beispiel notwendig, weil die Shellvariable '-' auch leer sein kann. In diesem Fall würde sie ohne die Anführungszeichen aus der Kommandozeile entfernt, was in dem Vergleich zu einem verdeckten Syntaxfehler und einem falschen Ergebnis des Tests führen könnte.

**case** ***Wort*** **in** [***Muster*** [ |***Muster*** ...] **)** ***Liste*** ;; ...] **esac** Mit der case Anweisung können leicht Verzweigungen programmiert werden, bei denen viele Fälle unterschieden werden.

Das **case** ***Wort*** wird von der bash erweitert, dann wird die daraus entstandene Zeichenkette mit den Mustern verglichen und bei Übereinstimmung die Liste von Kommandos ausgeführt. In den Suchmustern können reguläre Ausdrücke wie bei der Pfadnamen–Erweiterung (z. B. '*' und '?') verwendet werden.

Wenn ein übereinstimmendes Muster gefunden wurde, wird die case Anweisung beendet und 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.

Das dem letzten Beispiel zugrunde liegende Problem kann auch mit der case Anweisung gelöst werden. Diese Lösung ist zwar kein typisches Beispiel für eine Vielfachverzweigung, sie bietet sich aber wegen der Auswertung regulärer Ausdrücke durch case in der Praxis eher an als das oben gegebene Beispiel.

```
case $- in
        *i\*) echo hier sollte nach der Zeichenkette 'i*'
              echo gesucht werden.
              echo   *****FEHLER***** in der bash-1.12.;;
        *i*)  echo Die Shell arbeitet interaktiv.;;
        *)    echo Die Shell arbeitet nicht interaktiv.;;
esac
```

In der bash Version 1.12 konnte mit der case Anweisung nicht nach regulären Ausdrücken selbst gesucht werden, die Quotierung der Wildcards wurde ignoriert. Ab Version 1.13 ist dieser Fehler behoben.

[**function**] ***Name*** **()** {***Liste***} definiert eine neue Scriptfunktion *Name*.

Scriptfunktionen 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 die unter dem Funktionsnamen gespeicherte *Liste* von Kommandos ausgeführt. Es wird dazu kein neuer Prozeß erzeugt.

Im Unterschied zu alias–Synonymen können Scriptfunktionen Argumente verarbeiten. 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 Variablen für Scriptfunktionen zu erzeugen. Normale Shellvariablen 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 nur dann an alle Untershells 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.

### 3.3.15 Parameter

Aus dem Blickwinkel des Shellprogrammierers ist die zentrale Rolle von Variablen unmittelbar klar. Variable sind symbolische Namen für Platzhalter, in denen Werte (Zahlen oder Zeichenketten) gespeichert werden können.

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

***Name*=[*Wert*]**

Als Namen für Variable kommen beliebige alphanumerische (ASCII-) Zeichenketten in Frage, die mit einem Buchstaben beginnen. Der Unterstrich '_' wird zu den erlaubten Buchstaben gerechnet.

Zwischen den Bestandteilen der Zuweisung dürfen keine Leerzeichen stehen.

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

Es ist nicht notwendig (aber möglich), Variable zu deklarieren. Automatisch erzeugte Variable speichern alle ihnen zugewiesenen Werte als Zeichenketten. Die Interpretation des Inhalts einer Variablen erfolgt "aus dem Zusammenhang". Das heißt, in arithmetischen Ausdrücken werden Ziffernfolgen automatisch als Zahlen interpretiert. Durch ausdrückliche Deklaration einer Variablen als Integer (mit der Shellfunktion declare -i) wird erreicht, daß jede Zuweisung automatisch der arithmetischen Erweiterung unterworfen wird. Damit ist garantiert, daß eine solche Variable nach ihrer Initialisierung immer Zahlen enthält.

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.

### Shell- und Umgebungsvariable

Shellvariable haben auch in der interaktiven Shell eine große Bedeutung.

- Zum einen können bestimmte Variable aus der allgemeinen Prozeßumgebung (Environment) der Shell wie Shellvariable benutzt werden. Einige Umgebungsvariablen existieren bereitsn wenn die Shell aufgerufen wird. Neue Umgebungsvariable können mit dem export–Shellkommando erzeugt werden. Die Prozeßumgebung wird vom Elternprozeß an die Kinder vererbt, so daß alle Umgebungsvariablen der bash an die von dieser Shell gestarteten Prozesse weitergegeben werden. Die Kindprozesse können dann aus ihrer Prozeßumgebung wichtige Informationen über das System, in dem sie laufen erhalten.
- Zum anderen benutzt die Shell selbst Variable nicht nur als Platzhalter in Shellprogrammen, sondern sie verwendet bestimmte Shellvariable zur Anpassung ihres eigenen Verhaltens an eine bestimmte Systemumgebung. Der Benutzer kann diese Variablen verändern und damit das Erscheinungsbild der Shell seinen Bedürfnissen und seinem Geschmack entsprechend einrichten.

Auf der Benutzerebene können Umgebungsvariable wie Shellvariable beliebig erzeugt, gelesen und verändert werden. Deshalb erscheinen die Umgebungsvariablen als eine Untermenge der Shellvariablen. Umgebungsvariable können mit dem printenv–Kommando oder dem export Shellkommando angezeigt werden. Sämtliche definierten Shellvariablen werden mit dem set Shellkommando ausgegeben.

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 Wörtern 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 (→ Seite 79).

Die PATH Variable wird in der Regel von der Systemverwalterin in der Datei `/etc/profile` für alle Login–Shells gemeinsam auf einen bestimmten Wert gesetzt. Dieser Wert kann vom Benutzer beliebig verändert werden. Die PATH Variable kann nicht mit dem unset Shellkommando gelöscht werden.

Aus Gründen der Systemsicherheit ist es sinnvoll, das aktuelle Verzeichnis (bezeichnet durch einen Punkt) nur als letztes der zu durchsuchenden Verzeichnise anzugeben. Anderenfalls könnte ein Standardprogramm aus dem Systempfad versehentlich mit einem gleichnamigen Kommando im aktuellen Verzeichnis verwechselt werden.

**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 86).

**CDPATH** ist eine durch Doppelpunkt getrennte Liste von Verzeichnissen. Wenn beim cd Shellkommando kein absoluter Verzeichnisname und kein Verzeichnis relativ zum 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 Shellscripts 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. Ist die Variable gesetzt (egal welcher Wert), erfolgt eine Warnung, wenn auf die Postdatei zugegriffen wurde. Damit kann sowohl festgestellt werden, ob neue Mail angekommen ist, als auch jeder (möglicherweise illegale) Lese–Zugriff auf die Mailbox überwacht werden.

Dieser Schalter funktioniert nur, wenn die Mailbox in einem Verzeichnis liegt, dessen Dateisystem die Zugriffszeit verwaltet (nicht Minix oder Extended1).

**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 83) 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 (→ set, Seite 92). Voreinstellung ist '+'.

**NO_PROMPT_VARS** ist ein Schalter. Wenn die Variable gesetzt ist, werden nur die im Abschnitt über die Eingabeaufforderung (→ Seite 83) beschriebenen speziellen Parameter erweitert.

**HISTSIZE** setzt die Anzahl der im Kommandozeilenspeicher erinnerten Zeilen fest. Alle älteren Zeilen werden vergessen.

**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. Voreinstellung für HISTFILE ist `~/bash_history` Wenn Sie (z.B. im X Window System) mehrere Shells gleichzeitig mit derselben Sicherungsdatei benutzen, bleiben nur die Kommandozeilen der zuletzt beendeten Shell erhalten.

**HISTFILESIZE** bestimmt die Anzahl der im HISTFILE zwischen zwei Sitzungen gespeicherten Kommandozeilen. Alle älteren Zeilen gehen verloren.

**OPTERR** ist ein Schalter. Wenn die Variable den Wert 1 enthält, werden die Fehlermeldungen der getopts Shellfunktion ausgegeben. Enthält sie eine 0, werden die Fehlermeldungen unterdrückt.

**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 abgewartet, 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 88).

**FIGNORE** kann eine durch Doppelpunkte getrennte Liste von Dateiendungen enthalten, die bei der automatischen Dateinamenerweiterung ignoriert werden sollen. Beispielsweise kann diese Variable mit ".o:.bak:.old:~" belegt werden.

**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 Wörter zusammen werden nicht erkannt. Wenn die Variable nicht gesetzt ist oder irgendeinen anderen Wert enthält, werden alle Kommandozeilen in den Kommandozeilenspeicher geschrieben.

**command_oriented_history** ist ein Schalter. Wenn die Variable gesetzt ist, versucht die Shell alle Zeilen eines mehrzeiligen Kommandos in einem History–Eintrag zu speichern.

**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 Datei `/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.

Die folgenden Variablen werden automatisch durch die Shell gesetzt. Die meisten können vom Benutzer nicht verändert werden.

**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 91).

**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 Shellevel. Bei jedem Aufruf einer neuen 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 jede Sekunde automatisch um eins.

**LINENO** enthält die aktuelle Zeilennummer im Shellscript. Wenn die Variable innerhalb einer Scriptfunktion aufgerufen wird, entspricht die Zahl den bis zum Aufruf innerhalb der Funktion ausgeführten einfachen Kommandos. Außerhalb von Shellscripten ist diese Variable nicht sinnvoll belegt. Wenn die LINENO–Shellvariable mit dem unset–Kommando gelöscht wird, kann sie nicht wieder mit ihrer automatischen Funktion erzeugt werden.

**OPTARG** enthält das Argument zu der zuletzt von getopts ausgewerteten Option (→ Seite 89).

**OPTIND** enthält den Index der zuletzt von getopts ausgewerteten Option (→ Seite 89).

### Positionsparameter

Ein Positionsparameter wird durch eine positive ganze Zahl bezeichnet, also durch Ausdrücke wie `$1`, `$2` ...referenziert. Die Positionsparameter werden beim Aufruf der Shell in der Kommandozeile oder durch das set Shellkommando (mit den Optionen '–' oder '––' → Seite 92) gesetzt.
Wenn ein Positionsparameter mit mehr als einer Stelle (>9) bezeichnet werden soll, muß er in geschweifte Klammern gesetzt werden: z.B. `${12}`.

### Spezialparameter

Einige Parameter haben eine 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 Shellscript ausgeführt wird.

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

### 3.3.16 Erweiterung

Nachdem die Kommandozeile in einzelne Kommandos zerlegt ist, werden die Wörter jedes einzelnen Kommandos nach Token durchsucht, die zu ersetzen sind. Es gibt acht Formen der Erweiterung:

1. Klammer–Erweiterung { , }
2. Tildenerweiterung ~
3. Parameter- und Variablen–Erweiterung ${ }
4. Kommando–Substitution ‘ ‘ oder $( )
5. Arithmetische Erweiterung $[ ]
6. Prozeß–Substitution < ( )
7. Worttrennung
8. Pfadnamen–Erweiterung * ? [ ]

Nur durch Klammer–Erweiterung, Worttrennung oder Pfadnamen–Erweiterung können neue Wörter in der Kommandozeile entstehen. Die Ergebnisse aller anderen Erweiterungen werden als einzelne Wörter interpretiert. Einzige Ausnahme ist der ‘`$@`’ Spezialparameter, dessen Inhalt ungeachtet von Blanks als einziges Wort behandelt wird.

#### Klammer–Erweiterung

Die Klammer–Erweiterung generiert aus einer in geschweiften Klammern eingeschlossenen Liste von Bausteinen mehrere Zeichenketten (Wörter). Wenn die Klammer von konstanten Zeichenketten eingeschlossen ist, werden diese Zeichenketten an jedes der erzeugten Wörter angefügt. Es können mehrere Klammern verschachtelt werden.

Die dem Beispiel zur `for`-Schleife zugrunde liegende Aufgabe kann durch Klammer–Erweiterung auf einer einzigen Kommandozeile formuliert werden:

```
$ mkdir /usr/local/man/{man,cat}{1,2,3,4,5,6,7,8,9,n,l}
$ _
```

Dieses Beispiel zeigt, daß im Unterschied zur Pfadnamenerweiterung hier auch neue Dateinamen erzeugt werden können.

Die Klammer–Erweiterung wird vor allen anderen Ersetzungsoperationen durchgeführt. Es handelt sich um eine reine Textoperation, es werden keine Zeichen aus den Bausteinen verändert. Das geschieht erst in den darauf folgenden Schritten.

Mit dieser Behandlung geschweifter Klammern verhält sich die bash absichtlich anders als die Bourne–Shell, die solche Zeichen als Teil von Wörtern nicht verändert. Um die Kompatibilität zur sh wiederherzustellen, kann die Klammer–Erweiterung durch die Kommandozeilenoption −nobraceexpansion oder durch das Kommando `set +o braceexpand` abgeschaltet 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 Umgebungsvariablen HOME bzw. das Heimatverzeichnis des aktuellen Anwenders ersetzt.

Wenn der Tilde ein '+' folgt, wird das aktuelle Verzeichnis aus der Shellvariablen PWD eingesetzt. Folgt der Tilde ein '−', so wird mit dem letzten aktuellen Verzeichnis aus der Variablen OLDPWD erweitert.

Die Shell führt eine Tildenerweiterung bei der Zuweisung von Zeichenketten an Shellvariable durch. Dabei werden neben der Tilde am Wortanfang auch solche erkannt und ersetzt, die auf den Doppelpunkt ':' folgen. Auf diese Weise kann Tildenerweiterung auch für die Variablen PATH, CDPATH und MAILPATH durchgeführt werden.

Die Tildenerweiterung findet nach der Klammer–Erweiterung und vor allen anderen Erweiterungen statt.

### 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 Tilden–Erweiterung, 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.

Normalerweise wird eine Parametererweiterung an der Stelle im Shellscript durchgeführt, wo der entsprechende Wert das erste Mal benutzt wird. Bei der im folgenden vorgestellten Möglichkeit, Variable mit Defaultwerten zu belegen, ist es oft erwünscht, solche Belegungen gemeinsam an bestimmten Stellen des Scripts durchzuführen. Weil die Parametererweiterung nur als Bestandteil eines Kommandos oder einer Zuweisung durchgeführt wird, bietet sich zu diesem Zweck die : Funktion an. Diese "Shellfunktion“ hat keine direkte Wirkung. Als Seiteneffekt können aber Parameter auf der Kommandozeile erweitert werden.

**${*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 des Wortes 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 nichtinteraktive Shell wird verlassen. Wenn der Parameter gültig gesetzt ist, wird sein Wert eingesetzt.

**${*Parameter*:+*Wort*}** erzwingt die Benutzung eines anderen Wertes. Wenn der Parameter weder leer, noch 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*}**
Der Anfang der im *Parameter* enthaltenen Zeichenkette wird mit dem erweiterten *Wort* verglichen und der übereinstimmende Teil aus dem *Parameter* entfernt. Dabei werden Jokerzeichen (Wildcards) ('*', '?' und [...]) behandelt, wie bei der Pfadnamenerweiterung (Seite 82) beschrieben. Im Fall der Konstruktion mit einem '#' wird das kürzeste passende Muster vom Parameter entfernt, im Fall der zwei '##' das längste.

Beispielsweise kann der Kommandoname ohne Pfadanteil aus einem absoluten Kommando durch die Konstruktion `basename=${pathname##*/}` aus dem Pfadnamen in `pathname` extrahiert werden; eine Aufgabe, für die normalerweise das Shellutility `basename` benutzt wird.

**${*Parameter*%*Wort*}** und
**${*Parameter*%%*Wort*}**
Das Ende der im *Parameter* enthaltenen Zeichenkette wird mit dem erweiterten *Wort* verglichen und der übereinstimmende Teil aus dem Parameter entfernt. 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 Konstruktion mit zwei '%%' das längste.

Beispielsweise kann analog zum vorhergehenden Beispiel der reine Pfadanteil eines absoluten Kommandos durch die Konstruktion `dirname=${pathname%/*}` aus der Zeichenkette in `pathname` gezogen werden.

### 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 Apostrophs (Backquote, nicht Hochkomma) durchgeführt wird, behält das Fluchtsymbol '\' 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 Apostrophs ein Fluchtsymbol stehen.

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

### Arithmetische Erweiterung

Durch die arithmetische Erweiterung ist es möglich, mit Shellvariablen zu rechnen.

Durch die Konstruktion

$**[*Ausdruck*]**

wird anstelle des *Ausdrucks* das Ergebnis der darin formulierten arithmetischen Berechnungen eingesetzt.

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):

**+ −** die Vorzeichen

**! ∼** die logische und die bitweise Negation

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

**+ −** Addition und Subtraktion

**<< >>** bitweise Links- und Rechts–Shift–Operation

**<= >= < >** die Vergleiche

**== !=** gleich und ungleich

**&** bitweise Addition

**∧** bitweise XOR (Exclusiv–ODER)

**|** bitweise ODER

**&&** logisch UND

**||** logisch ODER

***Zuweisungen*** durch die Zuweisungsoperatoren = ∗= /= %= += −= <<= >>= &= ∧= |=kann wie in der Programmiersprache C eine Zuweisung mit einer Rechenoperation verbunden werden; zum Beispiel der Ausdruck
`let zahl=$zahl+1`
kann durch den Zuweisungsoperator '+=' zu
`let zahl+=1`
verkürzt werden

Alle Shellvariablen sind als Operanden erlaubt. Die Berechnung macht natürlich nur Sinn, wenn es sich bei den Werten um gültige Zahlen handelt. 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.

Die kombinierten Zuweisungsoperationen sowie die logischen UND-, ODER- und Shift–Operationen gibt es erst seit `bash` Version 1.13.

Der Ausdruck wird behandelt, als ob er in Anführungszeichen stünde. Zusätzliche Anführungszeichen im Ausdruck werden ignoriert. Alle Wörter des Ausdrucks werden vor der Berechnung einer Parametererweiterung, Kommandosubstitution und Quotenreduktion unterzogen.

### Prozeß–Substitution

Eine dem Pipelining ähnliche Erweiterung wird durch die Prozeß–Substitution angeboten. Durch die beiden Konstruktionen
< (*Subkommando*)
> (*Subkommando*)
wird das *Subkommando* gestartet und sein Aus- bzw Eingabekanal mit einer automatisch erzeugten "named pipe" oder FIFO–Datei verbunden. Auf der Kommandozeile wird die Konstruktion durch den Namen dieser FIFO–Datei ersetzt, so daß das eigentlich aufgerufene Kommando aus dieser Datei lesen bzw. dorthin schreiben kann.

Im Gegensatz zu einer normalen Pipeline können durch Prozeß–Substitution mehrere Pipelines gleichzeitig benutzt werden.

Die Subkommandos einer Prozeß–Substitution werden gleichzeitig mit den Parameter-, Kommando- und Arithmetik–Erweiterungen ausgeführt.

### 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 Token, 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 Wörter der Kommandozeile nach den Zeichen *, ? und [ durchsucht. Wenn ein solches Zeichen gefunden wird, wird das gesamte Wort, das dieses Zeichen enthält, 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. Andernfalls 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, 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 auf das Zeilenende

**[...]** paßt auf alle der in den eckigen Klammern eingeschlossenen Zeichen. Ein Paar 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 Wörtern erweitert wie in der C–Shell.

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

### Quoten–Reduktion

Nach allen Erweiterungen werden die unquotierten Fluchtsymbole und Apostrophe entfernt.

## 3.3.17 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 dasselbe Synonym nicht rekursiv ersetzt, so daß 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 (→ Seiten 84 und 96).

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

### 3.3.18 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 95).

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.

### 3.3.19 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 vorangegangen 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, ohne Pfad
- **\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 die Superuserin (ruth)
- *nnn* das Zeichen mit dem (oktalen) Code *nnn*
- \\\\ der Backslash '\\'

### 3.3.20 Wenn alles getan ist

Wenn die Shell alle Ersetzungen in der Kommandozeile vorgenommen und alle Umleitungen vorbereitet hat, ist der Zeitpunkt gekommen, auf den der Anwender die ganze Zeit gewartet hat. Die Shell versucht, das Kommando auszuführen. Dazu muß sie es aber erst lokalisieren. Das erste passende Kommando wird ausgeführt.
Als Kommandoname wird immer das erste Wort eines einfachen Kommandos erkannt. Ein Kommandoname kann mit Pfadnamen in einem Verzeichnis (absolut oder relativ) angegeben werden. Die Shell erkennt das an (mindestens) einem Slash '/' im Kommandonamen. Wenn kein Verzeichnis angegeben ist, versucht die Shell selbst, das Kommando zu finden. Dazu wird der Kommandoname zuerst in der Hashtabelle (→ Seite 89) gesucht, dann wird er mit den Synonymen, mit den Scriptfunktionen und schließlich mit den Shellfunktionen verglichen.
Wenn auf diese Weise kein Programm dieses Namens gefunden wird, werden alle in der PATH Umgebungsvariablen aufgeführten Verzeichnisse nach einer ausführbaren Datei dieses Namens durchsucht. Wenn auch hier kein passendes Kommando gefunden wird, gibt die Shell eine Fehlermeldung aus.
Wenn die Shell das Kommando lokalisieren kann, wird ein Argumentvektor zusammengestellt. Die Positionsvariable 0 wird mit dem vollständigen Pfadnamen des Kommandos belegt, die übrigen Argumente (wenn welche vorhanden sind) werden in den folgenden Positionsparametern gespeichert und in der Laufzeitumgebung des Kommandos, dem Environment, gespeichert. Dieser Argumentvektor kann dann beispielsweise in einem C–Programm durch die `main(int argc, char* argv)` Funktion übernommen und später ausgewertet werden.
Wenn die Ausführung fehlschlägt, weil die Datei keine ausführbaren Binärdaten enthält, versucht die bash automatisch, diese Datei als Shellscript auszuführen. Dazu wird eine neue Shell (Subshell) erzeugt, die wie bei einem Aufruf in der Kommandozeile neu (durch die in der ENV Umgebungsvariablen angegebene Datei) initialisiert wird.
Häufig beginnen Scriptprogramme mit den Zeichen '#!', gefolgt von einem Programmnamen. An dieser Konstruktion erkennt der Kernel selbst, daß es sich nicht um ein Binärprogramm handelt.[13] Beispielsweise die Zeile `#! /bin/bash -e` leitet ein bash Shellscript ein, das sofort abbricht, wenn eines der aufgerufenen Kommandos einen Status != Null liefert.

### 3.3.21 Eingebaute Shellkommandos

**:**

Syntax: **:** [*Argument* ...]
Das Shellkommando : hat keinen direkten Effekt. Im Gegensatz zum Kommentar werden aber die "Argumente" wie bei normalen Kommandos erweitert. Selbstverständlich muß das : Shellkommando an einer für Kommandos zulässigen Position in der Kommandozeile stehen.
Seine Bedeutung hat dieses Kommando in Shellscripts, wo bestimmte Parameter–Erweiterungen, beispielsweise Belegung von Shellvariablen mit Default–Werten (→ Seite 79), unabhängig von einem einzelnen Kommando ausgeführt werden sollen.

**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 Wörtern, 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.

[13] Die Binärprogramme benutzen ähnliche Symbole, die sogenannten Magic Numbers, mit denen das Betriebssystem die Eignung eines Programms für einen bestimmten Rechner feststellen kann. Seit der Version 0.99.13 unterstützt der Kernel neben dem Linux–spezifischen a.out Format auch das ELF (Extended Link Format) der neuen SYSV R4 Unixe. Die unterschiedlichen Formate erkennt der Kernel wieder an den Magic Numbers.

### bg

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

Das Kommando bg startet einen (mit ^Z) angehaltenen Prozeß (Job) im Hintergrund. Wenn keine Jobspezifikation (→ Seite 68) angegeben wurde, wird der zuletzt angehaltene Job gestartet.

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

### bind

Syntax: **bind** [−m *Keymap*][−ldv][−q *Kommandobezeichnung*]

**bind** [−m *Keymap*][−f *Dateiname*]

**bind** [−m *Keymap*][*Tastenfolge*: *Kommandobezeichnung*]

Mit dem bind Shellkommando kann die Tastenbelegung der readline Editorfunktionen angezeigt oder neu definiert werden.

Die readline Funktionen unterstützen mehrere Editoremulationen. Dazu werden mehrere Keymaps verwaltet.

Die Syntax einer Tastaturbelegung entspricht der für die '~/.inputrc' Datei (→ Seite 62).

Die Optionen haben die folgende Bedeutung:

**-m *Keymap*** wählt die bezeichnete Befehlstabelle *Keymap* zur Anzeige bzw. zur Veränderung; die folgenden Keymaps werden von readline verwaltet:

**emacs**

**emacs-standard**

**emacs-meta**

**emacs-ctlx**

**vi**

**vi-move**

**vi-command**

**vi-insert**

**-l** gibt eine Liste aller möglichen Kommandobezeichnungen (readline–Funktionen) aus

**-v** gibt eine Liste der aktuellen Editorfunktionen inklusive der gespeicherten Tastenbelegung 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

### 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 Liste von (durch Doppelpunkt getrennten) Verzeichnissen angegeben werden, in denen das 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 im Verzeichnisbaum nächsttiefere Verzeichnis heißt '..' und das letzte aktuelle Verzeichnis (OLDPWD) heißt '–'.

### command

Syntax: **command** [–pVv]*Kommando* [*Argument...*]
Das Shellkommando command führt das angegebene (einfache) *Kommando* ohne die normale shellinterne Identifizierung aus. Dadurch werden nur die fest eingebauten Shellfunktionen und die Dateien aus den Verzeichnissen in PATH ausgeführt.

**-p** schränkt die Suche nach dem Kommando auf den Standardpfad ein; auf diese Weise kommen nur die Standard–Systemkommandos zur Ausführung

**-v** (verbose) wortreich (→ Langenscheidts Taschenwörterbuch "Englisch")

**-V** (even more verbose) wortreicher ('tschuldigung)

### 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.
Das Kommando erlaubt folgende Optionen:

**-f** (zeigt) nur Funktionsnamen

**-r** setzt die Variable(n) auf 'nur Lesen' Status

**-x** markiert die Variable für automatischen Export in alle Subshellumgebungen

**-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 91).

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

### echo

Syntax: **echo** [−neE][*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

\\\\ das Zeichen '\\'

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

### enable

Syntax: **enable** [−n][−all] [*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.

Mit der Option '−all' werden alle eingebauten Shellkommandos aktivert und eine Liste ausgegeben.

### eval

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

Das eval Shellkommando fügt die Argumente zu einer Kommandozeile zusammen, die ausgeführt wird, ohne die Shell zu verdrängen. Sinn dieses Shellkommandos ist es, eine Kommandozeile mehrfach der Parameter–Erweiterung zu unterziehen.

Wenn in einem Shellscript Variablen eingesetzt werden müssen, die ihrerseits wieder Variable enthalten, oder wenn aufgrund der Reihenfolge ihrer Ausführung die gewünschte Erweiterung nicht erzielt wird, ist dieses Shellkommando das Mittel der Wahl.

### 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), das 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" Variablen und Funktionen enthalten sein können. Diese können vom Prozeß ausgewertet und benutzt werden.

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

Das export–Shellkommando schreibt bereits existierende Shellvariable in die Prozeßumgebung und macht sie so zu Umgebungsvariablen. Wenn die Option '−n' gesetzt ist, wird die Variable aus der Prozeßumgebung entfernt, innerhalb der Shell bleibt sie als Shellvariable erhalten. 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 Variablennamen beim Aufruf von export eine Zuweisung erfolgt, wird eine existierende Variable mit diesem Wert belegt oder eine neue mit diesem Wert erzeugt.

### fc

Syntax: **fc** [−e *Editor*] [−nlr] [*Anfang*] [*Ende*]
**fc** −s [*Muster*=*Ersatz*] [*Kommandozeile*]

Mit dem Shellkommando fc (fix command) 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 auch kein Anfang gesetzt ist, wird das letzte Kommando genommen.

Wenn die Option '`-l`' 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 umgekehrter Reihenfolge in den Editor geladen.

Wenn das Shellkommando fc in der zweiten Form aufgerufen wird, ersetzt es das *Muster* in der *Kommandozeile* durch *Ersatz*. Wenn kein Muster/Ersatz–Paar angegeben wird, kommt die Kommandozeile unverändert zur Ausführung.

### fg

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

Das Shellkommmando fg bringt einen (mit ^Z) angehaltenen Prozeß im Vordergrund zum Laufen. Wenn keine Jobspezifikation (→ Seite 68) angegeben ist, wird der zuletzt angehaltene Job im Vordergrund gestartet.

### getopts

Syntax: **getopts** *Optionen Variable* [*Argumente*]

Die getopts Shellfunktion kann in Shellscripten verwendet werden, um die Kommandozeile nach (konventionell) gültigen Optionen und Argumenten zu durchsuchen.

Wenn ein Shellscript als Kommando aufgerufen wird, kann es auf der Kommandozeile Optionen und Argumente übernehmen. Diese Parameter sind in den Positionsparametern (→ Seite 77) gespeichert und können so innerhalb des Scripts angesprochen und z. B. mit einer case Konstruktion verarbeitet werden.

Wenn die Kommandozeile den konventionellen Regeln entsprechend aufgebaut ist, kann sie einfacher mit der getopts Shellfunktion auseinander genommen werden (→ Seite 53).

In der Zeichenkette *Optionen* werden alle Schalter und Regler mit ihren Kennbuchstaben angegeben. Regler, die zusätzliche Argumente erhalten, werden von einem Doppelpunkt gefolgt.

Immer dann, wenn getopts aufgerufen wird, gibt es eine Option in der beim Aufruf bezeichneten *Variable* zurück. Wenn diese Variable nicht existiert, wird sie erzeugt. Wenn die Option ein Argument erwartet (also mit einem Doppelpunkt markiert ist), wird dieses Argument in der Shellvariablen OPTARG zurückgegeben.

Bei jedem Aufruf von getopts wird der Zeiger in der Shellvariablen OPTIND erhöht, damit bei einem weiteren Aufruf automatisch die nächste Option eingelesen wird.

Der OPTIND wird beim Start der Shell mit 1 initialisiert. Wenn eine Kommandozeile mehrfach eingelesen werden soll, muß der Index manuell zurückgesetzt werden.

Wenn nicht ausdrücklich eine *Argument* Zeichenkette beim Aufruf von getopts übergeben wird, nimmt das Shellkommando die Positionsparameter, also die Argumente von der Kommandozeile des Scripts.

Der Status von getopts ist 0 (wahr), wenn eine Option gefunden wurde und falsch, wenn das Ende der Kommandozeile bzw. der Argumente erreicht ist oder wenn ein Fehler aufgetreten ist.

Die Ausgabe von Fehlermeldungen in den Fehlerkanal kann durch Plazieren eines Doppelpunktes als erstes Zeichen der Optionen–Zeichenkette oder durch Belegung der Shellvariablen OPTERR=0 unterdrückt werden.

Zur weiteren Fehlerbehandlung kann bei der "stillen" Variante in der *Variablen* ein symbolischer Fehlercode und in der Shellvariablen OPTARG das zuletzt gelesene Token gefunden werden.

### 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 Hilfe–Text zu dem angegebenen Shellkommando oder, wenn kein Kommando angegeben ist, eine Übersicht aller Shellkommandos, zu denen Hilfe–Texte 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** [−lnp] [*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 vorhergehenden aktuellen Job (→ Seite 68).

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 68 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 80 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. Wenn die Shell als bash gestartet wurde, wird dabei das Shellscript '~/.bash_logout' abgearbeitet.

### 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 angegeben 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) Wörter den benannten Shellvariablen zu. Wenn mehr Wörter in der Zeile stehen als Namen angegeben sind, werden die verbleibenden Wörter alle in der zuletzt 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** [−npf] [*Name*]

Die Shellfunktion readonly 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.

Mit der '−n' Option können Variable mit "nur Lesen" Status wieder beschreibbar gemacht werden. Die einzige Ausnahme sind die reale und die effektive User–ID, die nicht direkt verändert werden können.

### 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** veranlaßt die Shell, alle neu erzeugten oder veränderten Variablen automatisch zu exportieren

**−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** veranlaß die Shell, Funktionen sofort bei ihrer Definition zu speichern, nicht erst bei ihrem Aufruf

**−i** zwingt eine Shell, interaktiv zu arbeiten; in diesem Fall wird immer die Initialisierungsdatei `~/.bashrc` interpretiert

**−k** veranlaßt die Shell, beim Aufruf eines Kommandos die komplette Kommandozeile in die Umgebung einer Funktion zu schreiben, 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 angegebene *Option*. Dabei sind folgende Optionen erlaubt:

- **allexport** das gleiche wie die Option `−a`
- **braceexpand** in geschweiften Klammern eingeschlossene, durch Kommata getrennte Listen von Wortteilen in der Kommandozeile werden durch mehrere Wörter 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
- **interactive-comments** veranlaßt die Shell, Kommentarzeichen '#' auch auf der Kommandozeile zu erkennen
- **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

**privileged** das gleiche wie die Option −p

**verbose** das gleiche wie die Option −v

**vi** schaltet den Kommandozeileneditor in den vi-Stil

**xtrace** das gleiche wie die Option −x

**−p** (privileged) veranlaßt die Shell, beim Start die ENV Initialisierungsdatei und die aus der Umgebung geerbten Funktionen zu ignorieren; dieser Schalter wird automatisch gesetzt, wenn die effektive User–ID nicht mit der realen übereinstimmt; beim Zurücksetzen des Schalters wird die effektive User–ID sofort mit der realen gleichgesetzt

**−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 so aus, 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 csh

**−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 der 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 Shellscript *Datei* abarbeiten. Dabei wird kein neuer Shell–Prozeß gestartet, sondern der Inhalt der Scriptdatei in den Eingabekanal der aktuellen Shell eingespeist. 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! (Siehe auch im Abschnitt über den Status der Kommandos auf Seite 64)

Ein Ausdruck kann einen einstelligen (unären) oder einen zweistelligen (binären) Operator enthalten. Einstellige Operatoren benötigen ein einziges Argument, zweistellige Operatoren stehen zwischen zwei Argumenten. Unäre Operatoren dienen oft zum Ermitteln des Zustandes einer Datei.

Außerdem können mehrere Ausdrücke noch durch spezielle Operatoren verknüpft werden.

Folgende Operationen können ausgeführt werden:

**–b *Datei*** ist wahr, wenn die *Datei* ein Blockdevice ist

**–c *Datei*** ist wahr, wenn die *Datei* ein Zeichendevice 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 (plain file) ist

**–g *Datei*** ist wahr, wenn bei der *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 ist

**–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*** (newer than) ist wahr, wenn die *Datei1* neuer ist als die *Datei2*

***Datei1* –ot *Datei2*** (older than) ist wahr, wenn die *Datei1* älter ist als die *Datei2*

***Datei1* –ef *Datei2*** (equal to file) 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, es wird also nicht der (eventuell numerische) Inhalt der Variablen getestet

***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 an.

### trap

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

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 die leere Zeichenkette 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 '`−l`' 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 "getrapten" Signale und der damit verbundenen Kommandos ausgegeben.

### type

Syntax: **type** [−atp] [−all] [−type | −path] [*Name*]

Die Shellfunktion `type` gibt an, wie der angegebene *Name* von der Shell interpretiert würde, wenn er in der Kommandozeile an der Position eines Kommandos stünde (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; die Veränderung solcher Grenzen innerhalb des durch "harte" Grenzen gegebenen Rahmens ist jederzeit möglich

**−H** setzt "harte" Grenzen; eine einmal gesetzte harte Grenze kann nicht nach oben erweitert werden

**−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** verbietet dem Anwender, Dateien über einer bestimmten Größe zu erzeugen; ist nur im Extended–2 Filesystem implementiert

**−s** schränkt den Stapelspeicher (Stack) auf eine bestimmte Größe ein

**−p** zeigt die Größe des Pipeline–Puffers (in 512 Byte Blöcken) an; dieser Wert kann nicht verändert werden

**−n** zeigt die maximale Anzahl offener Dateien an; dieser Wert kann nicht verändert werden

**−m** nicht implementiert; ist dafür vorgesehen, die Größe des residenten (nicht auszulagernden) Teiles der Prozesse einzuschränken

**−t** nicht implementiert; ist dafür vorgesehen, die verfügbare CPU-Zeit einzuschränken

**−u** nicht implementiert; ist dafür vorgesehen, die Anzahl der Prozesse je Benutzer einzuschränken

**−v**

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 an.

## umask

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

Die Shellfunktion umask setzt die Maske, mit der die Zugriffsrechte auf Dateien und Verzeichnisse unmittelbar nach ihrer Erzeugung durch einen von dieser Shell kontrollierten Prozeß[14] bestimmt werden. Die in der Maske gesetzten Bits werden bei den Zugriffsrechten für eine neue Datei (oder ein neues Verzeichnis) gelöscht (sie werden maskiert).

Die Maske kann als Oktalzahl oder in der beim Kommando chmod auf Seite 101 angegebenen Form angegeben werden. Wenn kein Wert angegeben ist, wird die aktuelle Maske angezeigt. Wenn die Option '-S' gesetzt ist, wird die aktuelle Maske in symbolischer Form ausgegeben.

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

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| - | r | w | x | r | w | x | r | w | x |
| | ⇓ | ⇓ | ⇓ | ⇓ | – | ⇓ | ⇓ | – | ⇓ |
| - | r | w | x | r | - | x | r | - | x |

Bei der Erzeugung einer Datei wird die creat(2) Funktion mit einem Wert für die Permissions aufgerufen, beispielsweise 0777. Durch die umask werden die Schreibrechte für Gruppe und Andere gelöscht.
Die übrigen Rechte kommen unverändert durch die Maske und erscheinen in der I-Node der frisch erzeugten Datei.

Abbildung 3.1: Maskierung der Zugriffsrechte durch umask

## 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.

[14] Jeder Prozeß kann mit der umask(2) Bibliotheksfunktion seine Dateierzeugungsmaske verändern, eine Garantie bietet die umask der Shell also nicht.

### unset

Syntax: **unset** [–fv] [*Name* ...]

Mit der Shellfunktion unset werden Shellvariable oder Shellfunktionen aus dem Speicher entfernt. Mit der Option '–f' wird die bezeichnete Funktion aus dem Speicher gelöscht, 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 wirklich aus dem Speicher 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 dem Arbeitsspeicher der Shell 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.

## 3.3.22 Login- und andere Shells

Wenn die bash mit einem '–' als erstem 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 Teminal (bzw. der Konsole) verbunden. Wenn die Shell mit der Option '–i' gestartet wird, arbeitet sie interaktiv.

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

- Wenn ein Shellscript `/etc/profile` existiert (und lesbar ist), werden die darin beschriebenen Einstellungen und Kommandos ausgeführt.
- Wenn 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, die im Falle ihrer Existenz ebenfalls gelesen und abgearbeitet werden.
- 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.

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

## 3.3.23 Optionen

Zusätzlich zu den beim Shellkommando **set** (→ Seite 92) beschriebenen Optionen und Schaltern, die auch beim Aufruf der bash in der Kommandozeile verwendet werden können, versteht die 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

**-noprofile** unterdrückt beim Aufruf als Loginshell die Bearbeitung der Initialisierungsdateien /etc/profile und ~/.bash_profile

**-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

### 3.3.24 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.

### 3.3.25 Dateien

**/bin/bash** ist das ausführbare Programm.

**/bin/sh** ist eigentlich die Standard–Bourne–Shell. Weil die aber keine Freie Software ist, wird unter Linux meistens ein Link auf bash gesetzt.

**/etc/profile** ist die Standard–Initialisierungsdatei für die Bourne–Shell und auch für die bash als Loginshell. Das folgende Beispiel zeigt eine /etc/profile Datei:

```
export OPENWINHOME=/usr/openwin
export DISPLAY=":0"

PATH="/bin:/usr/bin:/usr/local/bin:/usr/X386/bin:$OPENWINHOME/bin"
PS2='> '

umask 022
ulimit -c 0
```

Wenn die bash als Loginshell gestartet wird, versucht sie, eine zusätzliche Initialisierungsdatei im Heimatverzeichnis auszuführen.
Dabei wird in der Reihenfolge nach folgenden Dateien gesucht:
**~/.bash_profile**
**~/bash_login** und
**~/.profile**.
Die erste existierende Datei wird ausgeführt, die weiteren Dateien werden ignoriert. Folgendes Beispiel zeigt eine .bash_profile Datei:

```
PATH=$PATH:/usr/TeX/bin:/usr/local/scripts:~/bin:.
declare -r PATH

CDPATH=~:/usr/src:/usr
```

```
PS1='\w\n\u \$ '
MAIL=/var/spool/mail/she
export EDITOR=/usr/bin/vi


HISTSIZE=200
HISTFILESIZE=100
history_control=ignoredups

alias home=cd
alias l='v -a'
alias term='term < /dev/cua1 > /dev/cua1 2> /dev/null &'

source .bashrc
```

**~/.bashrc** ist die Initialisierungsdatei für die bash als interaktive Nicht–Loginshell. Das folgende Beispiel zeigt eine sinnvolle `.bashrc` Datei:

```
if [ $WINDOWID ]; then
  TERM=xterm
  export XDVIFONTS=/usr/TeX/lib/tex/fonts/%f.%d%p
else
  TERM=con100x40
fi
```

**~/.bash_logout** wird beim Verlassen der Login–Shell ausgeführt. Diese Datei kann beispielsweise folgende Zeilen enthalten:

```
sort .bash_history | uniq > .bash_history
clear
```

**~/.bash_history** ist die Sicherungsdatei für den Kommandozeilenspeicher. Der Name und die Größe können in den entsprechenden Shellvariablen (→ Seite 75) eingestellt werden.

**~/.inputrc** enthält die benutzerdefinierten Tastaturkommandobelegungen für den Kommandozeileneditor. Inhalt und Format sind auf Seite 62 beschrieben.

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

## 3.4 cat

### Funktion:

**cat** (con*cat*enate) 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 (→ Seite 66) 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 '|' (→ Seite 68) an den Filter weitergeleitet.

**Optionen:**

- **-b** alle nicht leeren Zeilen erhalten eine Zeilennummer
- **-e** das gleiche wie -vE
- **-n** sämtliche Zeilen werden numeriert
- **-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

**Beispiele:**

Mit dem cat–Kommando können Dateien ausgedruckt werden. Mit dem Kommando

```
$ cat Adressenliste > /dev/lp1
$ _
```

können Sie die Datei Adressenliste direkt der Gerätedatei /dev/lp1 zuführen, die den ersten parallelen Druckerport im System darstellt. Diese Methode ist aber sehr unelegant. Sie umgeht und blockiert den Druckerspooler lpd, der über die reine Auslieferung hinaus noch die Formatierung der Dokumente erledigen kann. Wenn ein Druckerdämon installiert ist, sollten Sie die "direkte" Methode nur in Ausnahmefällen benutzen.

Wenn Sie zwei Dateien durch Umlenkung der Ausgabe von cat verketten wollen, müssen Sie den Standardausgabekanal immer in eine dritte Datei zielen lassen.

```
$ cat foo bar > foo
cat: foo: input file is output file
$ _
```

führt NICHT zum erwünschten Ergebnis. Die Datei 'foo' wird von der Shell vor der Ausführung von cat gelöscht, um die Ausgabe des Kommandos dorthin umzulenken. cat erkennt noch, daß eine Eingabedatei mit der Ausgabedatei übereinstimmt, Ihre Daten sind aber bereits verloren.

Ein verwandtes Problem kann beim Anhängen von Daten an eine bestehende Datei entstehen. Der Befehl

```
$ cat foo bar >> foo
cat: foo: input file is output file
$ _
```

würde zu einem "Kurzschluß" führen, weil der schreibende Dateizeiger immer dem lesenden eine Dateilänge vorauseilen würde. Nicht unbedingt das gewünschte Ergebnis. cat erkennt diesen Fehler und verweigert die Ausführung. Die Datei foo bleibt also unverändert erhalten.

**Siehe auch:**

tac auf Seite 185

**Autor:** Torbjorn Granlund und Richard Stallman

## cd

**cd** ist ein eingebautes Shellkommando. Siehe auch im Abschnitt über die bash auf Seite 86.

## 3.5 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 der Superuserin (ruth) erlaubt. Sie können Ihre eigenen Dateien nur den Gruppen zuordnen, denen Sie selbst auch angehören.
Eine Aufstellung aller zulässigen Gruppen erhalten Sie mit dem Kommando `id -a` oder mit `groups`.

**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 wird ebenfalls geändert

**Siehe auch:**

chmod auf Seite 101 und chown auf Seite 195

**Autor:** David MacKenzie

## 3.6 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 der Systemverwalterin (ruth) 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 vier Ziffern angegeben werden, 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.

Das dritte Bit (1) schließlich hat unter Linux nur bei Verzeichnissen eine Bedeutung.[15]

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) ändert die effektive Benutzerkennung bei der Programmausführung. Das SGID Bit auf einem Verzeichnis sorgt dafür, daß alle Dateien, die in diesem Verzeichnis angelegt oder dorthin kopiert werden, Eigentum der entsprechenden Gruppe sind. Die Option **t** (text) schützt die Dateien eines beschreibbaren Verzeichnisses vor Löschung durch fremde Systembenutzer. 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.

**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

**Beispiel:**

| oktal | verbal | Anzeige ls -l | Bedeutung |
|---|---|---|---|
| 640 | u=rw,g=r,o= | `-rw-r-----` | Nur der Eigentümer kann diese Datei verändern. Die Gruppenmitglieder können die Datei lesen, alle anderen haben keinen Zugriff darauf. |
| 644 | a=r,u+w | `-rw-r--r--` | Alle Systembenutzer dürfen die Datei lesen, nur der Eigentümer darf sie verändern. |
| 4750 | a=,u=rwxs,g=rx | `-rwsr-x---` | Die ausführbare Datei kann vom Eigentümer gelesen, ausgeführt und verändert werden, die Gruppe darf sie lesen und mit der UID des Eigentümers ausführen, alle anderen haben keinen Zugriff auf die Datei. |
| 1777 | a=rwxt | `drwxrwxrwt` | Dieses Verzeichnis können alle Systembenutzer als aktuelles Verzeichnis wählen, sie können es auflisten und sie können Dateien darin anlegen. Das Löschen einer fremden Datei ist in diesem Verzeichnis nicht möglich. |

[15] Die Funktion des Stickybit, das das Textsegment eines Programms auch nach dessen Beendigung im Speicher hält, gibt es bei Linux nicht.

**Siehe auch:**

chgrp auf Seite 101 und chown auf Seite 195

**Autor:** David MacKenzie

## 3.7 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 der Systemverwalterin, zu verändern. Die Loginshell wird im letzten Feld des Benutzereintrags in der Paßwortdatei /etc/passwd festgelegt. Diese Datei kann nur mit Ruth's Privilegien 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 der Systemverwalterin gesetzt. Um die Systemsicherheit trotzdem zu gewährleisten, können nur Programme, die in der Datei /etc/shells eingetragen sind, als Loginshell benutzt werden.
Normalerweise können Sie nur Ihre eigene Loginshell ändern. Die Superuserin selbst kann das Programm aber auch für andere Benutzer anwenden, indem sie den Benutzernamen in der Kommandozeile angibt.

**Autor:** Peter Orbaek

## 3.8 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. Diese Prüfsummen können beispielsweise beim Verschicken per UUCP den Dateien mitgegeben werden, so daß der Empfänger die Möglichkeit hat, den vollständigen und korrekten Empfang der Dateien durch den Vergleich der Prüfsummen zu bestätigen.
Es gibt noch andere Verfahren zur Prüfsummenberechnung, die nicht dem POSIX.2 Standard entsprechen. Zwei davon werden durch das sum Kommando angeboten werden. Das cksum Programm ist nicht kompatibel zu sum.

**Autor:** Frank Q. Xia

**Siehe auch:**

sum auf Seite 184

## 3.9 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 und die Zeilennummer des ersten Bytes, in dem sich die Dateien unterscheiden. Wenn Sie anstelle eines der beiden Dateinamen ein Minuszeichen'–' angegeben, liest das cmp–Kommando die Vergleichsdaten von der Standardeingabe. 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 104

**Autor:** Torbjorn Granlund und David MacKenzie

## 3.10 comm

**Funktion:**

**comm** (common) 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) cmp auf Seite 104, sort auf Seite 177 und uniq auf Seite 190

**Autor:** Richard Stallman und David MacKenzie

## 3.11 compress

**Funktion:**

**compress** komprimiert Dateien

**Syntax:**

**compress** [−cdfrvV] [−b *Maxbits*] [*Datei* ...]

**Beschreibung:**

**compress** komprimiert Dateien mit LZW (einem veränderten Lempel–Ziv) Algorithmus. Bevor compress eine Datei durch die komprimierte Version ersetzt, überprüft es, ob die *Datei* nach der Kompression wirklich kleiner ist als vorher. Nur in diesem Fall wird die unkomprimierte *Datei* gelöscht. 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 verändert den Zeitstempel der Datei nicht.

Die Programme uncompress und zcat sind Links auf compress, bei denen bestimmte Optionen vorbelegt 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 kann als Standardpacker für Unix bezeichnet werden. Es wird im Zusammenhang mit tar auch für gepackte Dateiarchive verwendet. Das GNU tar bietet eine direkte Unterstützung von compress (→ Seite 186).

Allerdings wird in letzter Zeit dazu übergegangen, das neue gzip Programm zum Packen einzelner Dateien zu benutzen, weil es deutlich höhere Kompressionsraten erzielt. Im Bereich der Freien Software ist compress bereits überall durch gzip abgelöst.

**Optionen:**

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

**-d** (decompress) dekomprimiert die Datei; die komprimierte Datei wird dabei ersetzt

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

**-v** (verbose) gibt den Dateinamen und das Größenverhältnis aus

**-V** (Version) gibt die Versionsnummer aus

**-r** (recursiv) 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

## 3.12 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 Zugriffsrechte und Eigentümer des Originals (nicht die SUID und SGID Bits)

**-r** (recursive) kopiert rekursiv alle Dateien der Unterverzeichnisse (auch Devices und Links) wie normale Dateien

**-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) kopiert alle Unterverzeichnisse rekursiv; Spezialdateien bleiben erhalten

**-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 numeriert

**existing** Backups werden nur für Dateien numeriert, die bereits numerierte Backups haben.

**simple** es werden immer einfache Backups gemacht

**Autor:** Torbjorn Granlund, David MacKenzie und Jim Meyering

## 3.13 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:

Mit **csplit** können Sie beliebige Textdateien an bestimmten kontextabhängigen Stellen zerteilen. Aus einer Eingabedatei (oder der Standardeingabe) werden mehrere Ausgabedateien erzeugt, denen Inhalt von einem *Suchmuster* abhängig gemacht werden kann. Die erste Zeile, in der das *Muster* vorkommt, wird zur ersten Zeile der nächsten Datei. Das *Muster* muß folgendermaßen angegeben werden:

***/Ausdruck/***[*Offset*] ␣{***Anzahl***} erzeugt eine Ausgabedatei, die alle Zeilen der Eingabe bis (ausschließlich) zu 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 ingoriert 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** (silent) die Ausgabe der Ausgabedateigröße wird unterdrückt

**-k** (keep) 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*

### Beispiel:

Wenn Sie beispielsweise Ihre Mailbox (das ist die Datei, in der all ihre persönlichen eMails aneinandergehängt sind) auseinandernehmen wollen, können Sie das folgende Kommando versuchen:

```
$ csplit -k /usr/spool/mail/$LOGNAME /^From / {100}
812
770
695
2279
2279
3201
975
```

```
1964
2946
csplit: '/^From /': match not found on repetition 9
1188
$ _
```

Ihre Mailbox mit ankommender Mail befindet sich standardmäßig im Verzeichnis /usr/spool/mail und trägt Ihren Benutzernamen. Die einzelnen Mails werden durch sogenannte Header — also Briefköpfe — eingeleitet. Die Details zum Aufbau dieser Briefköpfe sind installationsabhängig. Die Suche nach einer Zeile, die mit `From` und einem darauffolgenden Leerzeichen beginnt, sollte aber mehr oder weniger exakt auf den Anfang des Mailheaders treffen.

Weil Sie wahrscheinlich nicht genau wissen, wie viele Mails in Ihrer Mailbox liegen, können Sie durch Angabe einer relativ hohen Zahl zusammen mit der `-k` Option erreichen, daß alle Mails getrennt und nach dem vorzeitigen Scheitern (hier nach dem 9. Durchgang) die bereits erzeugten Dateien nicht wieder gelöscht werden.

Die Dateien mit den einzelnen Mails werden im aktuellen Verzeichnis angelegt (Schreibberechtigung vorausgesetzt) und heißen xx00 bis xx09.

**Siehe auch:**

split auf Seite 178

**Autor:** Stuart Kemp und David MacKenzie

## 3.14 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 Standdardeingabe 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 durch Kommata getrennte 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

**Beispiel:**

Mit dem Kommando

```
$ cut -d : -f 1,5 /etc/passwd
ruth:Systemverwalterin
root:der traditionelle Superuser
daemon:der unbekannte Daemon
bin:
adm:
uucp:
news:Netnews Administrator
she:Sebastian Hetze
sync:
$ _
```

können Sie sich alle Benutzernamen (das 1. Feld) und die Realnamen (das 5. Feld) aller in `/etc/passwd` eingetragenen Accounts anzeigen lassen.

**Autor:** David M. Ihnat und David MacKenzie

## 3.15 date

**Funktion:**

**date** schreibt oder setzt die Systemzeit

**Syntax:**

**date** [−u] [−s *Datum*] [+*Formatstring*] [*MMDDhhmm*[[*CC*]*YY*][.*ss*]]

**Beschreibung:**

Das date–Kommando liest oder setzt die Linux Systemzeit.

Weniger zur Datumsausgabe auf der Kommandozeile als vielmehr zur Benutzung in Shellscripts kann date das Datum in vielfältiger Weise formatiert ausgeben. Es arbeitet dabei im Prizip als Front–End zur strftime(3) C–Bibliotheksfunktion. Ein durch '+' eingeleiteter Formatstring wird durch entsprechende Datumsangaben ergänzt und auf dem Standardausgabekanal angezeigt.

In einem volltändig eingerichteten Linux System passt sich date automatisch an die lokale Zeitzone an. Insbesondere findet der Wechsel zwischen Mitteleuropäischer Zeit (MET) und Sommerzeit (Daylight Saving Time, DST) automatisch statt. Die Zeitzone kann mit der Umgebungsvariablen TZ verändert werden.

Die vom POSIX Standard vorgesehen "Locales" zur länder- und sprachspezifischen Ausgabe des Datums sind bei Linux (noch) nicht implementiert.

In dem durch ein '+' eingeleiteten Formatstring werden die folgenden Parameter 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

### Die Systemzeit einstellen

Die Ein- bzw. Umstellung der Systemzeit ist nur der Superuserin erlaubt.
Die Angabe des neuen Datums kann entweder mit der `-s` Option im Format der normalen Datumsausgabe erfolgen, oder sie erfolgt in Form einer einzigen Zahl. 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)

Die Systemzeit von Linux ist nicht identisch mit der in der CMOS–Uhr fortgeschriebenen Zeit. Mit dem clock–Utility kann die Systemzeit aus der CMOS–Uhr gesetzt werden oder umgekehrt die CMOS–Uhr mit der aktuellen Systemzeit geladen werden. Damit die automatische Behandlung der Zeitzonen korrekt funktioniert, sollte die CMOS–Uhr auf GMT eingestellt werden. Die Systemzeit wird dann mit dem Kommando `clock -u -s` (beispielsweise beim Systemstart in `rc.local`) aus der CMOS–Uhr geladen.

Für das korrekte Handling der Zeitzonen müssen die entsprechenden Steuerdateien im Verzeichnis `/usr/lib/zoneinfo` installiert sein. Die Dateien `posixrules` und `localtime` müssen Kopien oder Links von `MET` sein. Die Umrechnung der GMT in die lokale Zonenzeit erfolgt durch die normalen Zeitfunktionen der Standardbibliothek. Deshelb wird bei **jeder** Datumsanzeige automatisch die Mitteleuropäische Zeit (bzw. die in der TZ Umgebungsvariablen festgelegte Zonenzeit) angezeigt.

### Optionen:

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

**-u** (universal) zeigt (oder setzt) die Zeit als UCT (Universal Coordinated Time, auch bekannt als Greenwich Mean Time)

### Umgebung:

In der TZ Umgebungsvariablen kann eine andere als die von der Systemverwalterin vorgegebene Zeitzone eingestellt werden. Wenn keine nach der Zeitzone benannte Datei in `/usr/lib/zoneinfo` existiert, wird automatisch die GMT benutzt.

### Beispiel:

Mit dem Kommando

```
$ date "+Es ist %H Stunden und %M Minuten nach Mitternacht."
Es ist 16 Stunden und 03 Minuten nach Mitternacht.
$ _
```

können Sie auf einfache Art eine umständliche Uhrzeit anzeigen lassen.

Wenn Sie Superuserrechte haben, können Sie die Systemzeit einstellen:

```
# date -s "Thu Sep 16 15:09:07 GMT 1993"
Thu Sep 16 17:09:07 MET DST 1993
# _
```

Das Datum wird intern durch eine vorzeichenbehaftete vier Byte Integer Variable repräsentiert. Der "Nullpunkt" ist am 1.1.1970 um 0 Uhr. Dieses Datum wird auch als Epoche bezeichnet. Der Bereich zulässiger Systemzeiten beginnt am 13.12.1901 um 20:45:52 und endet am 19.01.2038 um 03:14:07 Universal Coordinated Time.

**Autor:** David MacKenzie

## 3.16 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.

**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 Eingabeblöcke bis zur Größe von ibs mit Nullen

**Beispiel:**

Das Kommando

```
$ dd bs=8192 if=zImage of=/dev/fd0
26+1 records in
26+1 records out
$ _
```

können Sie benutzen, um die fertig übersetzte Kerneldatei (zImage) auf eine formatierte Diskette zu schreiben und so eine Bootdiskette zu erzeugen.
Mit dem Kommando

```
# dd if=/dev/hda of=/dev/fd0 bs=512 count=1
1+0 records in
1+0 records out
# _
```

kann die Superuserin (Ruth) eine Kopie des Festplattenbootsektors auf einer Diskette anlegen. Mit dieser Diskette kann die Festplatte gebootet werden, wenn der Festplattenbootsektor zerstört wurde.

**Autor:** Paul Rubin, David MacKenzie und Stuart Kemp

## 3.17 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 Festplattenplatz für das Dateisystem, 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

**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 156 nachzulesen.

## 3.18 dirname

### Funktion:

**dirname** gibt den Pfadanteil eines vollständigen Dateinamens aus

### Syntax:

**dirname** *Datei*

**Autor:** David MacKenzie und Jim Meyering

## 3.19 doshell

**Funktion:**

**doshell** startet eine Shell auf einem (virtuellen) Terminal ohne login

**Syntax:**

**doshell** *Terminal Shell*

**Beschreibung:**

**doshell** ermöglicht es, auch auf den (virtuellen) Terminals, auf denen von init keine getty–Prozesse gestartet wurden 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

## 3.20 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 (in Kilobyte). Wenn die Umgebungsvariable POSIXLY_CORRECT gesetzt ist, 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 GNU–Shellutility ist identisch mit dem echo–Shellkommando, das auf der Seite 87 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 115 im Handbuch erklärt.

## 3.21 elvis

**Funktion:**

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

**Syntax:**

**elvis** [−reviR] [−t *tag*] [−m [*Datei*]] [−w *Fenstergröße*] [−c *Befehl*] [+*Befehl*]

**Beschreibung:**

elvis bietet sowohl im 'visual mode', als auch im 'colon mode' nahezu alle Möglichkeiten des 'Vorbilds' ex/vi.

elvis kann unter verschiedenen Namen aufgerufen werden. Wird elvis als "vi" aufgerufen, so verhält er sich wie elvis. Unter dem Namen "view" läßt elvis keine Änderungen an der Datei zu und schützt sie so vor versehntlichem Überschreiben. Als "ex" startet elvis im "colon mode" (wie -e, und als "input" aufgerufen, befindet man sich sofort im "input mode", als sei die Option "-i" gesetzt.

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 (→ Seite 135).

elvis ist ein Public-Domain Programm und unterliegt keinen Beschränkungen in der Verwendung.

**Optionen:**

**-r** Bei dem originalen ex/vi bedeutet die "-r" Option, daß ein durch einen Absturz oder Stromausfall unterbrochener Editiervorgang wiederhergestellt werden soll. elvis verwendet hierfür ein separates Programm namens "elvrec" (→ S. 135) und gibt deshalb nur einen Hinweis auf dieses Programm aus.

**-R** Diese Option setzt das readonly Flag, so daß eine Datei nicht versehentlich überschrieben werden kann.

**-t *Tagname*** elvis positioniert den Cursor nach dem Starten auf den angegebenen Referenzpunkt ***Tagname***.

**-m [*Datei*]** elvis durchsucht die angegebene Datei nach Compilerfehlermeldungen, und platziert den Cursor in der Datei auf der Zeile, in der der Fehler aufgetreten ist. Wird keine Datei angegeben, so sucht elvis in der Datei "errlist".

**-e** elvis startet im "colon mode".

**-v** elvis startet im "visual command mode".

**-i** elvis startet im "input mode".

**-w *Fenstergröße*** Setzt die "window" Option auf den Wert von *Fenstergröße*.

**-c *Befehl* oder +*Befehl*** Nachdem elvis die erste Datei geladen hat, führt elvis *Befehl* als "ex" Kommando aus. Ein typisches Beispiel hierfür ist "elvis +75 *Datei*", um den Cursor automatisch auf Zeile 75 der *Datei* zu platzieren.

### 3.21.1 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'.

### 3.21.2 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. 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 meistens 1.

***Taste*** Einige Kommandos erfordern zwei Tastendrücke. Der erste ist immer der Befehl, der zweite ein Parameter zum Befehl. So bedeutet m*a* 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 Wörter. 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.

**Kommandos zum Positionieren des Cursors:**

| | **Befehl** | | **Beschreibung** |
|---|---|---|---|
| *Anzahl* | h | | Cursor rechts *Anzahl* Spalten (BEW) |
| *Anzahl* | ^H | | Cursor rechts (wie h) *Anzahl* Spalten (BEW) |
| *Anzahl* | l | | Cursor links *Anzahl* Spalten (BEW) |
| *Anzahl* | SPC | | Cursor links *Anzahl* Spalten (wie l) (BEW) |
| *Anzahl* | ^X | | Cursor zur Spalte *Anzahl* (BEW) (ERW) |
| *Anzahl* | \| | | Cursor zur Spalte *Anzahl* (BEW) |
| *Anzahl* | t | *Taste* | Cursor nach rechts **vor** das Zeichen *Taste* (BEW) |
| *Anzahl* | f | *Taste* | Cursor nach rechts **auf** das Zeichen *Taste* (BEW) |
| *Anzahl* | T | *Taste* | Cursor nach links **vor** das Zeichen *Taste* (BEW) |
| *Anzahl* | F | *Taste* | Cursor nach links **auf** das Zeichen *Taste* (BEW) |
| *Anzahl* | j | | Cursor abwärts *Anzahl* Zeilen (BEW) |
| *Anzahl* | ^J | | Cursor abwärts (wie j) *Anzahl* Zeilen (BEW) |
| *Anzahl* | ^N | | Cursor abwärts *Anzahl* Zeilen (BEW) |
| *Anzahl* | k | | Cursor aufwärts *Anzahl* Zeilen (BEW) |
| *Anzahl* | ^P | | Cursor aufwärts *Anzahl* Zeilen (BEW) |
| *Anzahl* | w | | Cursor rechts *Anzahl* Wörter (BEW) |
| *Anzahl* | W | | Cursor rechts *Anzahl* Wörter (BEW) |
| *Anzahl* | b | | Cursor links *Anzahl* Wörter (BEW) |
| *Anzahl* | B | | Cursor links *Anzahl* Wörter (BEW) |
| *Anzahl* | e | | Cursor zum Wortende *Anzahl* Wörter (BEW) |
| *Anzahl* | E | | Cursor zum Wortende *Anzahl* Wörter (BEW) |
| | ^ | | Cursor zum Zeilenanfang (BEW) |
| | 0 | | Cursor zum Zeilenanfang, wenn nicht Teil einer Wiederholungsangabe |
| *Anzahl* | _ | | Cursor zum Zeilenanfang oder *Anzahl* Zeilen abwärts (BEW) |
| | $ | | Cursor zum Zeilenende (BEW) |
| *Anzahl* | ^M | | Cursor abwärts zum Zeilenanfang *Anzahl* Zeilen (BEW) |
| *Anzahl* | + | | Cursor abwärts zum Zeilenanfang *Anzahl* Zeilen (BEW) |
| *Anzahl* | - | | Cursor aufwärts zum Zeilenanfang *Anzahl* Zeilen (BEW) |
| *Anzahl* | ) | | Cursor *Anzahl* Sätze vorwärts (BEW) |
| *Anzahl* | ( | | Cursor *Anzahl* Sätze zurück (BEW) |
| *Anzahl* | } | | Cursor *Anzahl* Absätze vorwärts (BEW) |
| *Anzahl* | { | | Cursor *Anzahl* Absätze zurück (BEW) |
| *Anzahl* | ^D | | bewegt den Cursor Richtung Dateiende um *Anzahl* Zeilen (Standard: 1/2 Bildschirmseite) |
| *Anzahl* | ^U | | bewegt den Cursor Richtung Dateianfang um *Anzahl* Zeilen (Standard: 1/2 Bildschirmseite) |

| | | | |
|---|---|---|---|
| | ^F | | bewegt den Cursor seitenweise zum Dateiende (BEW) |
| | ^B | | bewegt den Cursor seitenweise zum Dateianfang (BEW |
| ) | | | |
| *Anzahl* | ^E | | scrollt in Richtung Dateiende *Anzahl* Zeilen |
| *Anzahl* | ^Y | | scrollt in Richtung Dateiende *Anzahl* Zeilen |
| | \` | *Taste* | Cursor zu dem mit *Taste* markierten Zeichen (BEW) |
| | ' | *Taste* | Cursor zu der mit *Taste* markierten Zeile (BEW) |
| *Anzahl* | G | | Cursor zur Zeile *Anzahl* (Standard: zur letzten Zeile) (BEW) |
| *Anzahl* | H | | Cursor zur *Anzahl*-ten Bildschirmzeile (Standard: oberste Zeile) (BEW) |
| | M | | positioniert den Cursor auf den Anfang der mittleren Bildschirmzeile (BEW) |
| *Anzahl* | L | | positioniert den Cursor auf den Anfang der letzten Bildschirmzeile (BEW) |
| | z | *Taste* | bewegt die aktuelle Zeile zum Anfang(+) Ende(−) oder zur Mitte(.) des Bildschirms (BEW) |

**Kommandos zum Ändern von Text:**

| | **Befehl** | | **Beschreibung** |
|---|---|---|---|
| | d | *Bereich* | löscht den mit *Bereich* angegebenen Text (EDIT) |
| | D | | löscht den Text bis Zeilenende (EDIT) |
| *Anzahl* | x | | löscht das Zeichen, **auf** dem sich der Cursor befindet (EDIT) |
| *Anzahl* | X | | löscht *Anzahl* Zeichen links vom Cursor (EDIT) |
| | > | *Bereich* | schiebt den Text nach rechts (EDIT) |
| | < | *Bereich* | schiebt den Text nach links (EDIT) |
| *Anzahl* | i | *eing* | fügt Text vor dem Cursor ein (EDIT) |
| *Anzahl* | a | *eing* | fügt Text hinter dem Cursor ein (EDIT) |
| *Anzahl* | I | *eing* | fügt Text am Zeilenanfang ein (EDIT) |
| *Anzahl* | A | *eing* | hängt Text an das Zeilenende an (EDIT) |
| *Anzahl* | o | *eing* | eröffnet unter der aktuellen Zeile eine neue und fügt Text ein (EDIT) |
| *Anzahl* | O | *eing* | eröffnet über der aktuellen Zeile *Anzahl* neue und fügt Text ein (EDIT) |
| | R | *eing* | überschreibt vorhandenen Text mit *eing* (EDIT) |
| *Anzahl* | r | *Taste* | ersetzt *Anzahl* Zeichen durch *Taste* (EDIT) |
| *Anzahl* | s | *eing* | ersetzt *Anzahl* Zeichen durch *eing* (EDIT) |
| | c | *Bereich* | ändert den Text im Bereich *Bereich* (EDIT) |
| | C | *eing* | ändert den Text bis Zeilenende in *eing* (EDIT) |
| *Anzahl* | S | *eing* | ändert *Anzahl* Zeilen (wie *Anzahl* cc) (EDIT) |
| *Anzahl* | i | *eing* | fügt Text vor dem Cursor ein (EDIT) |
| *Anzahl* | a | *eing* | fügt Text hinter dem Cursor ein (EDIT) |
| *Anzahl* | I | *eing* | fügt Text am Zeilenanfang ein (EDIT) |
| *Anzahl* | A | *eing* | hängt Text an das Zeilenende an (EDIT) |
| | \ | *Bereich* | öffnet ein Pop-Up Menü zum Ändern von Text (ERW) |
| | u | | nimmt das letzte EDIT Kommando zurück |
| | U | | nimmt alle letzten Änderungen der momentanen Zeile zurück |
| *Anzahl* | J | | hängt die nachfolgenden *Anzahl* Zeilen an die momentane an (EDIT) |

**Arbeiten mit Puffern und Marken:**

| | **Befehl** | | **Beschreibung** |
|---|---|---|---|
| | y | *Bereich* | kopiert den mit *Bereich* angegebenen Text in einen Puffer |
| *Anzahl* | Y | | kopiert die aktuelle und *Anzahl* weitere Zeilen in einen Puffer |
| | “ | *Taste* | legt fest, welcher Zwischenpuffer als nächstes verwendet wird |
| | p | | fügt ausgeschnittenen Text hinter dem Cursor ein (EDIT) |
| | P | | fügt ausgeschnittenen Text vor dem Cursor ein (EDIT) |
| | @ | *Taste* | interpretiert den Inhalt eines Zwischenspeichers als ‘vi’ Befehle und führt sie aus |
| | m | *Taste* | markiert eine Zeile oder ein Zeichen |
| | v | | setzt den Anfang einer Markierung für c, d, y, <, >, !, \ (ERW) |
| | V | | setzt den Anfang einer Zeilenmarkierung für c, d, y, <, >, !,\ (ERW) |

**Kommandos zum Suchen nach Ausdrücken und Wörtern:**

| | **Befehl** | | **Beschreibung** |
|---|---|---|---|
| | ^A | | sucht nach dem nächsten Vorkommen des Wortes, auf dem der Cursor steht (BEW) (ERW) |
| *Anzahl* | % | | plaziert den Cursor auf die zugehörige ( ) { } [ ], oder plaziert den Cursor auf *Anzahl* Prozent der Datei (BEW) (ERW) |
| | / | *Text* | sucht vorwärts nach einem regulären Ausdruck *Text* (BEW) |
| | ? | *Text* | sucht rückwärts nach dem regulären Ausdruck *Text* (BEW) |

**Wiederholungs Kommandos:**

| | | | |
|---|---|---|---|
| *Anzahl* | . | | wiederholt das vorhergehende EDIT Kommando |
| | n | | wiederholt das letzte Suchen (BEW) |
| | N | | wiederholt das letzte Suchen in umgekehrter Richtung (BEW) |
| *Anzahl* | ; | | wiederholt das vorhergehende f, F, t, T Kommando (BEW) |
| *Anzahl* | , | | wiederholt das vorhergehende f, F, t, T Kommando in umgekehrter Richtung (BEW) |
| *Anzahl* | & | | wiederholt den letzten :s// Befehl an der momentanen Position (EDIT) |

**Sonstige Kommandos:**

| | **Befehl** | | **Beschreibung** |
|---|---|---|---|
| | ^G | | zeigt den Dateistatus und die momentane Zeilennummer |
| | Z | Z | speichert die Datei und beendet elvis |
| | ^L | | erneuert den Bildschirm (wie ^R) |
| | ^R | | erneuert den Bildschirm (wie ^L) |
| | ! | *Bereich* | führt die selektierten Zeilen einem externen Filter zu, der nach dem Kommando angegeben werden muß |
| | = | *Bereich* | formatiert *Bereich* neu |
| | ^^ | | zur vorherigen Datei |
| *Anzahl* | # | + | erhöht eine Zahl (EDIT) (ERW) |
| | : | *Text* | führt ein einzelnes ‘ex’ Kommando aus |
| | Q | | schaltet in den ‘ex’ Modus |
| | * | | bewegt den Cursor zum nächsten Fehler in der Fehlerliste (ERW) |
| | K | | dient in Verbindung mit dem Programm ctags dazu, Schlüsselworte nachzuschlagen |
| | ^] | | wenn sich der Cursor auf einem tag-Namen befindet, gehe dorthin |

### 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 elvis |
| ^[ | (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 Zeichen 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 X11-Window 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.

### 3.21.3 Der 'colon mode'

#### 'Colon mode' Befehle

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

| | | |
|---|---|---|
| | se[t] | [*Option*] |
| | so[urce] | [*Datei*] |
| [*Zeile*][,*Zeile*] | s[ubstitute] | */regAusdruck/Ersatz/*[p][g][c] |
| | ta[g][!] | *Tagname* |
| | una[bbr] | [*Abkürzung*] |
| | u[ndo] | |
| | unm[ap][!] | *Taste* |
| | ve[rsion] | |
| [*Zeile*][,*Zeile*] | v[global] | */regAusdruck/Befehl* |
| | vi[sual] | *Dateiname* |
| | wq | |
| [*Zeile*][,*Zeile*] | w[rite][!] | [[>>]*Datei*] |
| | x[it][!] | |
| [*Zeile*][,*Zeile*] | y[ank] | *x* |
| [*Zeile*][,*Zeile*] | ! | *externerBefehl* |
| [*Zeile*][,*Zeile*] | < | |
| [*Zeile*][,*Zeile*] | = | |
| [*Zeile*][,*Zeile*] | > | |
| [*Zeile*][,*Zeile*] | & | |
| | @ | *x* |

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 aus 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:

| | |
|---|---|
| :p | gibt die aktuelle Zeile aus |
| :*37*p | gibt Zeile 37 aus |
| :'*g*p | gibt die Zeile, in der die Marke *g* gesetzt ist aus |
| :*/foo/*p | gibt die nächste Zeile, die *foo* enthält aus |
| :$p | gibt die letzte Zeile des Textes aus |
| :*20,30*p | gibt die Zeilen *20—30* aus |
| :*1*,$p | gibt den gesamten Text aus |
| :%p | gibt ebenfalls den gesamten Text aus |
| :*/foo/–2,+4*p | gibt 5 Zeilen um das nächste Vorkommen von *foo* 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 angegeben 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 angegeben 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.

### Kommandos zum Editieren einzelner Zeilen

[*Zeile*][,*Zeile*] **join[!]**
[*Zeile*][,*Zeile*] **!** *progamm*
[*Zeile*][,*Zeile*] <

[*Zeile*][,*Zeile*] **>**
[*Zeile*][,*Zeile*] **substitute**/*regAusruck*/*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 Ausruck und ersetzt ihn durch *Ersatz* Die p Option gibt die geänderten Zeilen aus, die c Option fragt vor jedem Ersetzen, 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 Wörtern. Im 'input mode' ersetzt elvis *Kurzform* durch *erweiterteForm*, sobald nach *Kurzform* ein nicht alphanumerisches 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 names '.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 Markierungen) 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) schalten 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 plaziert 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 plaziert den Cursor in der entsprechenden 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 Pogrammnamen 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.

### 3.21.4 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. Kal*ende*r) 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 Orginaltextes ein

~ (Tilde) fügt eine Kopie des vorhergehenden Ersetzungstextes ein

\x fügt eine Kopie des Orginaltextes 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. Solange 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*/

### 3.21.5 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 Name, 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 gegenüber ex/vi. 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 Leerraum 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, indem 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 Standardwert 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 beinflußt 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 '!rm -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 in der Zeichenkette angegebenen Zeichen werden als Paare 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 beim 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 nach wie vor 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 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. h. Tabulatorzeichen werden zu einer bestimmten Anzahl Leerzeichen umgewandelt. Wird die list Option gesetzt, so zeigt elvis für jedes Tabulatorzeichen `^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.

**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.

**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 '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 paragraphs Option besteht aus einer Zeichenkette, in der jedes Zeichenpaar eine Befehlskombination des Absatzkommandos des Formatierers darstellt.

**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 eine 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 Betriebsystem nicht alle Änderungen sofort auf die Festplatte geschrieben. Über die sync Option läßt sich festlegen, ob elvis die Speicherung der Änderungen dem Betriebsystem 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.

**taglenght (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.

**warpmargin (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 dazuführen, daß alle Zeilen auf höchstens 70 Zeichen pro Zeile umgebrochen werden.

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

**writeany (wr)** Das Auschalten 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.

### 3.21.6 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 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. :*20,30*y *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. "*a*p 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 angegeben 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, das der Pufferinhalt Zeichen für Zeichen interpretiert wird. Jedes Zeichen wird als Tastendruck betrachtet. Ist der Pufferinhalt z. B. mit "*z*dd 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"*z*D ausgeschnitten werden.
Obwohl in Zwichenspeichern 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

## 3.22 elvrec

#### Funktion:

**elvrec** — Stellt die geänderte Version einer Datei wieder her, falls während des Arbeitens mit elvis ein Systemabsturz aufgetreten ist.

#### Syntax:

**elvrec** [***Datei*** [***neueDatei***]]

#### Beschreibung:

Ist während des Editierens mit elvis ein Systemabsturz oder Stromausfall aufgetreten, können oft große Teile der letzten Änderungen mit Hilfe der Temporärdateien, die elvis während des Editierens anlegt, wiederhergestellt werden.
Beim Neustart des Systems wird die editierte Datei nicht automatisch überschrieben, sondern elvprsv (→ S. 196) sichert die noch vorhandenen Änderungen in /usr/preserve und erstellt einen Index aller geretteten Dateien..
elvrec sucht nach den geretteten Änderungen und aktualisiert die *Datei*, bzw. speichert die aktualisierte Datei unter *neueDatei* ab.
Um eine Liste aller wiederherstellbaren Dateien zu bekommen muß elvrec ohne Parameter aufgerufen werden.

**Autor:** Steve Kirkendall

## 3.23 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 dem Ausdruck *Name*=*Wert* laßt sich die Umgebungsvariable *Name* 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 97). 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 87 zu finden.

## exec

**exec** ist eine eingebaute Shellfunktion. Eine Beschreibung ist im Abschnitt über die bash auf Seite 88 zu finden.

## exit

**exit** ist eine eingebaute Shellfunktion. Eine Beschreibung ist im Abschnitt über die bash auf Seite 88 zu finden.

## 3.24 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 88 zu finden.

## 3.25 expr

### Funktion:

**expr** (expression) bearbeitet einen Ausdruck

### Syntax:

**expr** *Ausdruck* …

### Beschreibung:

**expr** bewertet oder berechnet einen oder mehrere *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. 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

## 3.26 fdformat

### Funktion:

**fdformat** formatiert eine Diskette (low–level)

**Syntax:**

**fdformat** [−n] *Gerätedatei*

**Beschreibung:**

**fdformat** formatiert eine Diskette. Das Diskettenformat, die Kapazität und der Laufwerkstyp werden durch die entsprechende Gerätedatei ausgewählt. In der folgenden Tabelle sind die Gerätedateien mit den dazugehörenden Diskettenformaten aufgeführt.

| Gerätedatei | Minor | Floppy | Sec | Hds | Track | Zoll | KiloByte | Typ |
|---|---|---|---|---|---|---|---|---|
| /dev/fd0d360 | 4 | A | 9 | 2 | 40 | 5 1/4 | 360 | DD |
| /dev/fd0h360 | 20 | A | 9 | 2 | 40 | 5 1/4 | 360 | HD |
| /dev/fd0h720 | 24 | A | 9 | 2 | 80 | 5 1/4 | 720 | HD |
| /dev/fd0h1200 | 8 | A | 15 | 2 | 80 | 5 1/4 | 1200 | HD |
| /dev/fd0D360 | 12 | A | 9 | 2 | 40 | 3,5 | 360 | DD |
| /dev/fd0H360 | 12 | A | 9 | 2 | 40 | 3,5 | 360 | HD |
| /dev/fd0D720 | 16 | A | 9 | 2 | 80 | 3,5 | 720 | DD |
| /dev/fd0H720 | 16 | A | 9 | 2 | 80 | 3,5 | 720 | HD |
| /dev/fd0H1440 | 28 | A | 18 | 2 | 80 | 3,5 | 1440 | HD |
| /dev/fd1d360 | 5 | B | 9 | 2 | 40 | 5 1/4 | 360 | DD |
| /dev/fd1h360 | 21 | B | 9 | 2 | 40 | 5 1/4 | 360 | HD |
| /dev/fd1h720 | 25 | B | 2 | 2 | 80 | 5 1/4 | 720 | HD |
| /dev/fd1h1200 | 9 | B | 15 | 2 | 80 | 5 1/4 | 1200 | HD |
| /dev/fd1D360 | 13 | B | 9 | 2 | 40 | 3,5 | 360 | DD |
| /dev/fd1H360 | 13 | B | 9 | 2 | 40 | 3,5 | 360 | HD |
| /dev/fd1D720 | 17 | B | 9 | 2 | 80 | 3,5 | 720 | DD |
| /dev/fd1H720 | 17 | B | 9 | 2 | 80 | 3,5 | 720 | HD |
| /dev/fd1H1440 | 29 | B | 18 | 2 | 80 | 3,5 | 1440 | HD |

Die Typbezeichnung in der letzten Spalte bezieht sich nicht auf den Diskettentyp, sondern auf das Laufwerk. Die alten DD (40 Track) Laufwerke wurden vor allem in XT Rechnern eingesetzt.

Auf der Diskette werden nur leere Blöcke geschrieben und kein Dateisystem eingerichtet. Dazu stehen die Kommandos mkfs mkxfs und mke2fs (→ Seite 204ff) für Linux–Dateisysteme und mformat (→ Seite 159) 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

## 3.27 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 Schlüsselwörter '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 Schlüsselwörter benannt.

Der Dateisystemtest wird mit Hilfe des stat(2) 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 Schlüsselwörtern 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

**Autor:** Ian F. Darwin

## 3.28 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 zu Ende sind.

Das erste Argument, das mit einem '−', einer Klammer '(', ')', einem Komma ',' oder einem Ausrufezeichen '!' beginnt, wird als Anfang einer Option oder eines Tests 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 Linux-Verzeichnis 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.

Die Tests auf die erweiterten Zeitmarken (Zugriff und Erstellung) werden nur in solchen Verzeichnissen korrekt behandelt, die auf einem der neuen Linux–Dateisysteme angesiedelt sind (e2fs, xiafs, new minix). Auf den anderen Dateisystemen wird nur das Datum der letzten Änderung zuverlässig getestet. Das Ergebnis der anderen Tests hängt davon ab, ob der letzte Zugriff bzw. die letzte Änderung so kurz zurückliegen, daß die veränderte I–Node noch im Arbeitsspeicher (Cache) ist. Dann können auch für die Dateien der alten Dateissysteme alle drei Zeitmarken unterschieden werden.[16]

**-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[17]

[16] Die interne I–Node enthält für alle Dateisysteme die drei Zeitmarken. Erst bei dem Versuch diese Marken auf der Festplatte zu sichern, gehen die zusätzlichen Daten verloren.

[17] Der Name ctime legt den falschen Schluß nahe, es handele sich um die creation time, also das Datum der ersten Erstellung dieser Datei. Ein solches Datum wird nirgends gespeichert. Die ctime Marke wird bei jeder Änderung der I–Node einer Datei gesetzt. Damit ist die ctime immer jünger als die mtime.

**-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 entprechen exakt dem *Modus*; der *Modus* kann als Oktalzahl oder mit den bei chmod auf Seite 101 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[18]

[18] 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; das Semikolon kann nicht weggelassen werden, und es muß durch mindestens ein Whitespace von der letzen Option getrennt werden (damit es vor der Shell geschützt ist, muß es zusätzlich Quotiert werden); 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 65)

**-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[19]

**-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, von einem Nullbyte abgeschlossen, auf die Standardausgabe; 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 Backslash, 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 ctime–Funktion
- **%A***k* die Zeit des letzten Zugriffs auf die Datei, in dem von *k* bestimmte Format; k hat dabei das gleiche Format wie der entprechende 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)

[19] Die Gerätedateien /dev/stdout und /dev/stderr existieren nicht, sie können nur im Zusammenhang mit den Optionen des find–Kommandos benutzt werden.

- **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

**%C*k*** 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

**%T*k*** 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; wenn *Ausdruck1* wahr ist, wird *Ausdruck2* bewertet (ausgeführt)

***Ausdruck1* -a *Ausdruck2*** auch eine UND Verknüpfung

***Ausdruck1* -and *Ausdruck2*** auch eine UND Verknüpfung

***Ausdruck1* -o *Ausdruck2*** ODER Verknüpfung; *Ausdruck2* wird bewertet (ausgeführt), wenn *Ausdruck1* falsch ist

***Ausdruck1* -or *Ausdruck2*** auch eine ODER Verknüpfung

***Ausdruck1* , *Ausdruck2*** Liste; beide Ausdrücke werden immer bewertet (ausgeführt); der Wahrheitswert des gesamten Ausdrucks entspricht dem von Ausdruck2

**Beispiel:**

Die folgenden Beispiele zeigen typische Anwendungen des find–Kommandos aus dem Aufgabenbereich der Systemverwalterin.
Im ersten Beispiel wird das gesamte Dateisystem nach core–Files durchsucht. Diese Speicherabzüge bleiben häufig nach Programmabstürzen zurück. Um den dadurch unnütz belegten Festplattenplatz wieder nutzbar zu machen, sollten diese Dateien in regelmäßigen Abständen gelöscht werden. In dem Beispiel werden nur solche Dateien zum Löschen vorgeschlagen, die länger als 5 Tage nicht geöffnet wurden. Auf diese Weise wird verhindert, daß ein möglicherweise gerade zum debuggen eines Programms benutztes core–File dem Eigentümer "unter der Hand weg" gelöscht wird.

```
# find / -atime -5 -name "core*" -ok rm {} \;
< rm ... /home/she/core > ? n
< rm ... /usr/src/util-etc-2.1/core > ? y
< rm ... /usr/openwin/include/images/core_eye.icon > ? n
< rm ... /usr/openwin/include/images/coredoc.icon > ? n
# _
```

Im zweiten Beispiel wird das gesamte Dateisystem nach Dateien durchsucht, die mit den Privilegien des Eigentümers ausgeführt werden. Solche Programme sind immer ein potentielles Sicherheitsrisiko. Deshalb sollte in einem öffentlichen System regelmäßig der Bestand an solchen Dateien durchgesehen und auf Veränderungen geprüft werden.

```
# find / -type f -perm -4000 -exec ls -l {} \;
-r-sr-sr-x  1 daemon  daemon    12328 Aug 12 22:58 /usr/bin/lpq
-r-sr-sr-x  1 ruth    daemon    14048 Aug 12 22:58 /usr/bin/lpr
-r-sr-xr-x  1 uucp    daemon   123908 Mar 24  1993 /usr/bin/cu
-r-sr-xr-x  1 uucp    daemon    87044 Mar 24  1993 /usr/bin/uux
-r-sr-xr-x  1 uucp    daemon    87044 Mar 24  1993 /usr/bin/uucp
-r-sr-xr-x  1 uucp    daemon    37892 Mar 24  1993 /usr/bin/uuname
-r-sr-xr-x  1 uucp    daemon    99332 Mar 24  1993 /usr/bin/uustat
-rws--x--x  1 ruth    ruth      24352 Jan 16  1993 /bin/login
-rws--x--x  1 ruth    ruth      16464 Jan 16  1993 /bin/su
-rws--x--x  1 ruth    ruth      16864 Jan 16  1993 /bin/passwd
-rws--x--x  1 ruth    ruth      16292 Jan 16  1993 /bin/gpasswd
-rws--x--x  2 ruth    ruth      11176 Jan 16  1993 /bin/newgrp
-rws--x--x  1 ruth    ruth       9356 Jan 16  1993 /bin/chfn
-rws--x--x  1 ruth    ruth       8232 Jan 16  1993 /bin/chsh
-rws--x--x  1 ruth    ruth      16264 Jan 16  1993 /bin/chage
-rws--x--x  2 ruth    ruth      11176 Jan 16  1993 /bin/sg
---s--x--x  1 ruth    ruth      13316 Aug 13 07:14 /etc/traceroute
```

**Autor:** Eric Decker, David MacKenzie, Jay Plett und Tim Wood

## 3.29 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, so daß Tabulatoren korrekt behandelt werden. Ein Backspace erniedrigt die Zeichenzahl entsprechend, und ein Wagenrücklauf (carriage return) setzt die Zahl auf Null zurück.

### 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

## 3.30 free

**Funktion:**

**free** zeigt den freien Speicherplatz im Arbeitsspeicher und auf dem Swapdevice an

**Syntax:**

**free** [−msribpc] [−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 169 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

## 3.31 grep

**Funktion:**

**grep** (get regular expression print) gibt alle Zeilen aus, in denen ein bestimmter regulärer Ausdruck gefunden wird

**Syntax:**

**grep** [−CVbchilnsvwx] [−*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:[20]

***c*** ein einzelner Buchstabe paßt auf sich selbst

[20] Die Steuerzeichen müssen in Hochkomma oder Anführungszeichen eingeschlossen werden, damit sie nicht von der Shell bearbeitet werden.

**.** 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 einmal oder öfter vorkommen

**\|** die durch den Operator '\|' verbundenden Argumente werden **oder** verknüpft

**∧** (Caret) paßt auf den Zeilenanfang

**$** paßt auf das Zeilenende

**\<** paßt auf den Wortanfang[21]

**\>** paßt auf das Wortende[21]

**[*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.

**** jedes der Sonderzeichen kann, durch einen '\\' (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 gefundenden Zeile aus

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

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

**-*Anzahl*** gibt *Anzahl* Zeilen Kontext **vor und nach** jeder gefundenden 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

[21] Der Wortanfang bzw. das Wortende ist die Stelle zwischen dem ersten bzw. letzten Buchstaben und den das Wort einschließenden Leerzeichen.

**Beispiel:**

Beispielsweise können Sie mit dem Kommando

```
$ grep "[Pp]rozess[^eio]" Handbuch.tex
werden, erscheinen in der neuen Prozeßumgebung automatisch die
Die {\sf free} Version, die mit dem Prozeßdateisystem arbeitet,
natürlich in der Regel mehr als einen lauffähigen Prozeß. Und der
dem {\sf ps} Programm, das mit dem Prozeßdateisystem arbeitet,
$ _
```

feststellen, ob (noch immer) mehr als die in diesem Beispiel vorgestellten unkorrekten Schreibweisen des Wortes `Prozeß` in diesem Handbuch vorkommen.

**Autor:** Mike Haertel, James A. Woods und David Olson

## 3.32 groff

**Funktion:**

**groff** ist ein reiner Textfresser. Es ernährt sich von den weitverbreiteten '.1', '.n', '.an' und '.ms' Texten. 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 X11 Ausgabe mit 75dpi

**X100** für X11 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 Ausgabe*Format*; 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 im *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 '−m*Name*'

**/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*

**Beispiel:**

In den meisten Programmpaketen sind unformatierte Manualpages enthalten. Wenn Sie sich einen solchen Hilfstext ansehen wollen, ohne ihn gleich in einem Verzeichnis im MANPATH zu installieren, können Sie sich die Manualpage "von Hand" formatieren und mit einem Pager anzeigen lassen:

```
$ groff -mandoc -Tascii foobar.1 > cat.foobar.1
$ less cat.foobar.1
```

**Siehe auch:**

grog(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

## 3.33 groups

**Funktion:**

**groups** zeigt alle Gruppen, denen der Benutzer angehört

**Syntax:**

**groups** [*Benutzer* ...]

**Beschreibung:**

Mit dem **groups**-Kommando können Sie sich eine Liste aller Benutzergruppen ausgeben lassen, denen Sie fest angehören. Mit dem newgrp–Kommando können Sie jede dieser Gruppen zu Ihrer aktuell aktiven Benutzergruppe machen.

**Siehe auch:**

groups ist ein Shellscript und benutzt id

## 3.34 gzip

**Funktion:**

**gzip** komprimiert Dateien

**Syntax:**

**gzip** [−cdfhlLnNqrtvV19] [−S *Endung* [*Datei* ...]

**Beschreibung:**

**gzip** komprimiert Dateien mit dem LZ77 Lempel–Ziv Algorithmus. gzip erzielt erheblich 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 oder pack gepackt wurden. Archive, die mit zip gepackt wurden, können mit gzip nur ausgepackt werden, wenn sie eine einzige Datei enthalten und mit der "*deflation*" Methode 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 '.gz' anhängt. Wenn der Dateiname durch Anhängen der Endung unzulässig lang würde, verkürzt gzip automatisch den Namen um die erforderliche Anzahl Zeichen.

Die Zeitmarke der Datei und die Zugriffsrechte bleiben beim Komprimieren erhalten. Um die Wiederherstellung der Zeitmarke und des Dateinamens sicherzustellen, werden diese Daten mit eingepackt und können beim entkomprimieren verwendet werden. Außerdem wird eine CRC–Checksumme mit eingepackt, mit der beim Auspacken automatisch die Integrität der Daten geprüft wird.

Wenn gzip ohne Dateinamen aufgerufen wird, liest es von der Standardeingabe und schreibt auf die Standardausgabe. Der gleiche Effekt wird erzielt, wenn anstelle einer Eingabedatei ein Minuszeichen '−' angegeben wird.

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 der oben aufgeführten Formate aus. gunzip erwartet die Endung '.gz', '`-gz`', '`.tgz`', '`.taz`', '`.z`', '`-z`', '`-z`' oder '`.Z`' an dem Dateinamen. Außerdem wird die Datei auf eine "magische Zahl" überprüft, die mit gzip komprimierte Dateien identifiziert. Nach dem Auspacken bleiben die Zugriffsrechte und das Erstellungsdatum der Datei erhalten.

**zcat** arbeitet wie `gzip -dc`, schreibt also die entkomprimierte Datei auf die Standardausgabe und läßt die komprimierte Datei unberührt. Wenn die Eingabedatei die korrekte magische Zahl enthält, wird sie ausgepackt, egal welche Endung der Dateiname hat.

**Optionen:**

**-c** schreibt die (ent)komprimierte *Datei* auf die Standardausgabe, anstatt die Datei zu ersetzen

**-d** (decompress) dekomprimiert die *Datei*

**-f** (force) ersetzt bestehende Dateien mit Endung '.gz'; normalerweise fragt gzip vor dem Überschreiben solcher Dateien nach

**-h** (help) gibt eine Kurzhilfe zum Programm aus

**-l** (list) zeigt den in einer mit gzip komprimierten Datei gespeicherten originalen Dateiname, sowie die originale und die gepackte Größe an; wenn die `-v`–Option gesetzt ist, wird zusätzlich die Zeitmarke und die Checksumme ausgegeben

**-L** (license) gibt eine Kurzfassung des Lizenztextes aus

**-n** (no–name) unterdrückt beim Einpacken das Speichern des Dateinamen und der Zeitmarke (nur wenn der Name nicht gekürzt werden muß); beim Auspacken wird die Wiederherstellung des originalen Namens mit der Zeitmarke unterdrückt; diese Option ist Voreinstellung zum Entpacken

**-N** (name) veranlaßt beim Einpacken die Sicherung des originalen Namen und der Zeitmarke in der gepackten Datei und beim Auspacken die Wiederherstellung dieser Daten an der dekomprimierten Datei; diese Option ist Voreinstellung beim Einpacken

**-q** (quiet) unterdrückt alle Warnungen

**-r** (recursive) packt alle Dateien in den angegebenen Unterverzeichnissen

**-S** ***Endung*** veranlaßt die Verwendung der neuen *Endung* anstelle von '.gz'

**-t** (test) prüft die Integrität der angegebenen *Datei*

**-v** (verbose) gibt den Namen und 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 105 und tar auf Seite 186

**Autor:** Jean–Loup Gailly

## 3.35 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** [−*nr*bcklmqv] [*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 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** (quiet) unterdrückt die Ausgabe des Dateinamen

**-v** (verbose) gibt die Dateinamen immer aus

**Siehe auch:**

tail auf Seite 186

**Autor:** David MacKenzie

## 3.36 hostname

### Funktion:

**hostname** zeigt oder setzt den Netzwerknamen des Systems

### Syntax:

**hostname** [*Name*]

### Beschreibung:

Das hostname–Kommando wird benutzt, um den Netzwerknamen des lokalen Rechners anzuzeigen bzw. um ihn zu setzen. Um den Namen des Rechners im lokalen Netz zu ändern, benötigt man selbstverständlich Superuserrechte. Normalerweise geschieht das automatisch in der `rc.local` Systeminitialisierungsdatei.

Das hier vorgestellte hostname–Kommando ist nicht dafür gedacht, den vollständigen Domainnamen (Fully Qualified Domain Name) auszugeben. In der `utsname` Kerneldatenstruktur, die den Rechnernamen (`nodename`) enthält, ist ein gesonderter Eintrag für den `domainname` enthalten. Um diesen Eintrag anzuzeigen bzw. zu ändern, gibt es das domainname–Kommando.

Möglicherweise wird sich in Zukunft ein anderes hostname Kommando aus dem TCP/IP–Paket von Fred van Kempen durchsetzen, das den Hostnamen im BSD–Stil als vollständigen Domainnamen behandelt. Wenn Sie in Ihren Newsartikeln oder in den von Ihnen verschickten Mails doppelte Domainkennungen (im Stil von `soho.lunetix.de.lunetix.de`) in der Absenderadresse finden, sollten Sie nur den bloßen Rechnernamen ohne Domain als hostname benutzen.

### Siehe auch:

uname auf Seite 190 und domainname(1)

**Autor:** Peter Orbaek

## 3.37 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

## 3.38 join

**Funktion:**

**join** verknüpft zwei Dateien nach Schlüsselfeldern

**Syntax:**

**join** [−a 1|2] [−v 1|2] [−e *Zeichenkette*] [-o *Feldliste* ...] [-t *Buchstabe*] [-j[1|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

## 3.39 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) oder um Signale an bestimmte Prozesse zu senden.

kill sendet das Signal *Signr* an den Prozeß *Prozeßnr*. Standardwert ist SIGTERM (15) zum Terminieren des Prozesses. Weil das Signal SIGTERM nicht von allen Programmen abgefangen wird, wird ein Prozeß manchmal erst mit dem Signal SIGKILL (9) vom Kernel beendet. Die "normale" Terminierung mit SIGTERM ist aber vorzuziehen, weil dadurch dem Prozeß noch die Möglichkeit gegeben wird, die Bühne geordnet zu verlassen. Es können nur an die eigenen Prozesse Signale geschickt 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 90)

**Optionen:**

-***Signr*** sendet *Signr* anstelle von SIGTERM (15)

**Autor:** Peter MacDonald

## 3.40 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 (mit Option `-f`)

**-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 numerierte Backups
- **existing** macht numerierte Backups nur für bereits numerierte Dateien
- **simple** macht immer nur einfache Backups

**Autor:** Mike Parker und David MacKenzie

## 3.41 logname

**Funktion:**

**logname** zeigt den Benutzernamen

**Syntax:**

**logname**

**Beschreibung:**

Das Kommando **logname** zeigt den Benutzernamen, wie er von getty 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.

Weil nach dem getty bis zum Ausloggen des Benutzers kein Programm den Eintrag für ein bestimmtes Terminal in der `/etc/utmp` Datenbasis verändert, wird mit dem logname–Kommando immer der Name angezeigt, unter dem sich ein Systembenutzer ursprünglich im System angebmeldet hat, unabhängig davon ob er aktuell zum Beispiel nach einem su–Kommando unter einer anderen Benutzerkennung arbeitet.

## 3.42 ls

**Funktion:**

**ls** (list) zeigt den Inhalt eines Verzeichnisses

**Syntax:**

**ls** [−abcdgiklmnpqrstuxABCFLNQRSUX1] [−w *Spalten*] [−T *Spalten*] [−I *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** zeigt den Inhalt der Verzeichnisse des Dateisystems an.

Das Standardausgabeformat von ls hängt vom Typ der Ausgabedatei 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 nicht druckbaren 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 (zusammen mit Option −t)

**-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 sockets

alles 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

## 3.43 man

**Funktion:**

**man** zeigt die Manualpages 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 Thema *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, in der Umgebungsvariablen PAGER kann aber ein anderes Programm bestimmt werden.
Die Manualpages werden in verschiedene Kapitel unterteilt:

1. die Benutzerkommandos
2. die Systemaufrufe
3. die C–Bibliotheksfunktionen
4. die Beschreibungen der Gerätedateien
5. die Dateiformate
6. Spiele
7. die Makropakete für die Textformatierer
8. die Kommandos für die Systemverwalterin
9. für die Kernelroutinen

Außerdem werden noch die Sektionen '`l`' (lokale) '`p`' (private) '`n`' (new) und '`o`' (old) unterstützt. Außerdem hat LunetIX noch eine Sektion **g** für die deutschen Manualpages eingeführt.

Die Reihenfolge, in der eine Manualpage in den Sektionen gesucht wird ist wie folgt:

g, 1, n, l, 8, 3, 2, 9, 4, 5, 6, 7, p, o

In der MANSEC Umgebungsvariablen kann eine andere Reihenfolge bestimmt werden.

Die Verzeichnisse, in denen man nach Manualpages sucht, werden von der Systemverwalterin in der Datei /usr/lib/manpath.config bestimmt. In der MANPATH Umgebungsvariablen können andere Verzeichnisse angegeben werden.

#### 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 zum man–Kommando 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 90

**Autor:** John W. Eaton

## 3.44 mformat

#### Funktion:

**mformat** richtet ein MS–DOS Dateisystem ein

#### Syntax:

**mformat** [ –t *Spuren*] [ –h *Köpfe*] [ –s *Sektoren*] [ –l *Label*] *Laufwerk*:

#### Beschreibung:

Das **mformat**–Kommando gehört zu den mtools zur Bearbeitung von MS–DOS Disketten und richtet auf einer roh (low–level) formatierten Diskette ein MS–DOS Dateisystem ein. So eine Diskette kann entweder mit den übrigen mtools bearbeitet werden oder mit dem mount–Kommando in das Linux–Dateisystem eingebunden werden.

Die Standardparameter für Spuren/Köpfe/Sektoren werden aus der Datei /etc/mtools gelesen, sie können aber auch auf der Kommandozeile eingegeben werden.

### 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 Diskette

### Siehe auch:

fdformat auf Seite 137 und mtools auf Seite 162

## 3.45 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 101); der Standard, und damit der Ausgangswert für relative Modes, ist 0777 minus der Bits von umask (siehe bei umask auf Seite 96)

**-p** wenn ein Unterverzeichnis in einem nicht existierenden Verzeichnis angelegt werden soll, werden alle fehlenden Verzeichnisse angelegt

### Siehe auch:

ln auf Seite 155 und rmdir auf Seite 173

**Autor:** David MacKenzie

## 3.46 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 dem Wert von umask (→ Seite 96) 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

## 3.47 more

**Funktion:**

**more** zeigt Dateien seitenweise

**Syntax:**

**more** [ −cdflsu ] [ −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 Umgebungsvariablen MORE können Kommandozeilenoptionen 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.

***N*** LEERZEICHEN 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

***N*z** das gleiche wie Leerzeichen; die Zahl *N* wird die neue Anzahl Zeilen pro Bildschirm

***N*s** überspringt die nächsten *N* Zeilen und zeigt die darauffolgenden Zeilen an

***N*f** überspringt die nächsten *N* Bildschirme und zeigt die darauffolgenden Zeilen an

***N*b** springt *N* Bildschirmseiten rückwärts

***N*** CTRL-**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 Hilfekommando; gibt eine Übersicht über die more–Kommandos

***N*/*Ausdruck*** sucht das *N*-te Auftreten des Ausdrucks vom aktuellen Bilschirm an vorwärts

***N* n** sucht das *N*te aufteten 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*

**Autor:** Eric Shienbrood, Geoff Peck, John Foderaro

## 3.48 mtools

**Funktion:**

**mtools** ist eine Sammlung von Werkzeugen zur Bearbeitung von MS–DOS Dateisystemen auf Diskette oder Festplatte.

**Beschreibung:**

Die mtools von Emmet P. Gray ermöglichen alle Datei- und Verzeichnisoperationen auf MS–DOS Disketten. Von besonderer Bedeutung ist dabei, daß die Disketten nicht in das Dateisystem eingebunden werden, also kein mount–Kommando ausgeführt werden muß.

Unter der Sammelbezeichnung mtools verbergen sich dreizehn verschiedenene Programme (Tools), von denen jedes ein MS–DOS Vorbild hat.

| | | |
|---|---|---|
| mattrib | [{+,-}ahrs *Datei*] | ändert die MS-DOS Dateiattribute. |
| mcd | *Verzeichnis* | wechselt das Arbeitsverzeichnis auf der DOS Diskette. |
| mcopy | [-tnvm] *Quelle Ziel* | kopiert MS–DOS Dateien von/nach Linux. Besonders interessant ist die automatische Konvertierung von CR/LF in LF mit der Option -t. |
| mdel | [-v] *Datei* | löscht eine MS–DOS Datei. |
| mdir | [-w] *Datei* | zeigt den Inhalt eines DOS Verzeichnisses an. |
| mformat | → Seite 159 | legt ein DOS Dateisystem auf einer Low–Level formatierten Diskette an. |
| mlabel | [-v] *Laufwerk:* | erzeugt ein Volume–Label für eine MS–DOS Diskette. |
| mmd | [-v] *Verzeichnis* | legt ein Verzeichnis auf einer DOS Diskette an. |
| mrd | *Verzeichnis* | löscht ein Unterverzeichnis auf einer DOS Diskette. |
| mread | [-tnm] *Quelle Ziel* | liest (kopiert) eine Datei "roh" von DOS nach Linux. |
| mren | [-v] *Alt Neu* | benennt eine existierende DOS Datei um. |
| mtype | [-ts] *Datei* | gibt den Inhalt einer DOS Datei aus. |
| mwrite | [-tnvm] *Quelle Ziel* | schreibt (kopiert) eine Datei "roh" von Linux auf eine DOS Diskette. |

Genaue Beschreibungen der mtools können Sie in den Manualpages zu den Programmen nachlesen.

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 '~/.mcwd' 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

## 3.49 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 Zieldateien rücksichtslos

**-i** erwartet interaktiv eine Bestätigung vor dem Überschreiben existierender Zieldateien

**-u** verschiebt Dateien nur, wenn sie neuer sind als die gleichnamigen Zieldateien

**-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 106 erklärt

**Siehe auch:**

ln auf Seite 155 und cp auf Seite 106

**Autor:** Mike Parker und David MacKenzie

## 3.50 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

## 3.51 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 'Scheduler' der dafür sorgt, daß die Prozessorzeit möglichst optimal zugeteilt wird. Prozesse, die auf das Ergebnis eines anderen Prozesses, ein Interrupt oder ein Signal warten, können beispielsweise so lange 'schlafen', bis das erwartete Ereignis eingetreten ist. Der Kernel 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 Scheduler 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 der Superuserin **Ruth** erlaubt.

Wenn kein Wert angegeben ist, wird die Nettigkeit um 10 Punkte erhöht. Die maximale Nettigkeit ist 19 Punkte. Nach unten kann die Superuserin ihre Nettigkeit bis –20 abkühlen.

**Optionen:**

**-n *Wert***

**-*Wert*** setzt nice auf *Wert*

**Siehe auch:**

nohup auf Seite 166

**Autor:** David MacKenzie

## 3.52 nl

**Funktion**

**nl** numeriert 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 numeriert werden. Jeder Teil kann auch leer sein. Wenn vor dem ersten Kopfteil bereits Zeilen vorhanden sind, werden diese Zeilen wie ein Seitenkörper numeriert.

Die Numerierung beginnt auf jeder Seite neu. Mehrere Dateien werden als ein einziges Dokument betrachtet und die Zeilennummer wird 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 Zeilennumerierung für die Kopfzeile; das Nummerntrennzeichen wird auch den nicht numerierten Zeilen vorangestellt; als *Stil* werden folgende Zeichen erkannt

- **a** alle Zeilen werden numeriert
- **t** die leeren Zeilen werden nicht numeriert (Voreinstellung für den Körper)
- **n** die Zeilen werden nicht numeriert (Voreinstellung für Kopf und Fuß)
- **p** ***Ausdruck*** nur die Zeilen, in denen der reguläre *Ausdruck* vorkommt, werden numeriert

**-b** ***Stil*** bestimmt die Art der Zeilennumerierung für den Körper

**-f** ***Stil*** bestimmt die Art der Zeilennumerierung für den Fuß

**-p** die Zeilen aller Seiten werden fortlaufend numeriert

**-v** ***Nummer*** die erste Zeile jeder logischen Seite bekommt die angegebene Nummer

**-i** ***Nummer*** die Schrittweite für die Numerierung

**-l** ***Nummer*** die angegebene Anzahl aufeinanderfolgender Leerzeilen werden als eine Zeile angesehen, und die letzte Zeile wird numeriert; wenn weniger Leerzeilen in Folge auftreten, werden sie nicht numeriert; 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 168

**Autor:** Scott Bartram, David MacKenzie

## 3.53 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 Schedulerpriorität 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 148

## 3.54 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 Ruths Privilegien und darf in die Datei schreiben.

Bei einigen Linux–Installationen 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 103 und newgrp auf Seite 164

**Autor:** Peter Orbaek

## 3.55 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

## 3.56 pr

**Funktion:**

**pr** formatiert Textdateien zur Druckerausgabe

**Syntax:**

**pr** [+SEITE] [−SPALTEN] [−abcdfFmrtv] [−e[*Eintab*[*Schritt*]]] [−h *Kopf*][−i[*Austab*[*Schritt*]]] [−l *Seitenlänge*] [−n[*Separator*[*Stellen*]]] [−o *Lrand*] [−s[*Separator*]] [−w *Breite*] [*Datei* ...]

**Beschreibung:**

**pr** produziert seitenweise numerierte 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 letze 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

## 3.57 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

## 3.58 ps

**Funktion:**

**ps** (process status) zeigt die Prozesse mit ihrem Status an

**Syntax:**

**ps** −acehjlmnrsuvwxS −t*xx* *Systempfad* *Swappfad*

**Beschreibung:**

Mit **ps** lassen sich Daten über die Prozesse in der Prozeßtabelle anzeigen.

Es gibt zwei unterschiedliche Versionen von ps. Das eine ps greift direkt auf den Kernelspeicher zu, aus dem es die Prozeßtabelle und andere Daten ausliest. Dazu braucht ps 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[22] für alle Prozesse des Systems, in deren Unterverzeichnissen und Dateien alle für ps relevanten Daten zu finden sind. Das ps–Kommando bereitet diese Daten auf und zeigt sie dem Anwender in der gleichen Weise an wie die andere Version. Der Vorteil der Methode mit dem Prozeßdateisystem besteht in der Unabhängigkeit von der Kernelversion.

Die beiden Versionen von ps unterscheiden sich in verschiedenen Details bei den Ausgabeformaten und in den Kommandozeilenoptionen. Die Beschreibung hier bezieht sich hauptsächlich auf das procps. Die Optionen und Ausgabeformate des kernelabhängigen ps werden nur am Rande und unvollständig dokumentiert.

Die Prozeßtabelle wird mit einer Titelzeile ausgegeben. Die Spalten bedeuten folgendes:

**COMMAND** der Name des Kommandos; Prozesse, die komplett in den Swapbereich ausgelagert sind, werden in Klammern angezeigt

**TIME** die verbrauchte Rechenzeit (Summe User- und Kernelmodus) im Format MM:SS

**TT** die Nummer des kontrollierenden Terminals

**UID** die Benutzer–ID des Eigentümers dieses Prozesses

**PID** die Prozeßnummer dieses Prozesses

**PPID** die Prozeßnummer des Elternprozesses

**PGID** die Prozeßgruppe dieses Prozesses

**TPGID** die Prozeßgruppe, der z. Z. das kontrollierende Terminal zu diesem Prozeß gehört

**SID** die Session–ID des Prozesses (ID der Loginshell)

**STAT** ist der Status des Prozesses; folgende Symbole sind möglich:

- **R** lauffähig
- **S** schlafend
- **D** nicht störbarer Schlaf
- **T** angehalten oder verfolgt
- **Z** Zombie

Die folgenden Statusinformationen werden vom procps nicht angeboten.

- **W** der Prozeß belegt keine Seiten im Arbeitsspeicher
- **I** (idle) der Prozeß läuft leer
- **P** der Prozeß wird gerade in den Swapbereich ausgelagert
- **x** der Prozeß wird mit dem Debugger verfolgt
- **X** die System–Calls werden verfolgt
- **s** der Prozeß führt eine neue Session an
- **+** der Prozeß ist in einer Prozeßgruppe im Vordergrund
- **<** kennzeichnet die Prozesse mit höherer Priorität
- **N** kennzeichnet die Prozesse mit verminderter Priorität

**F** die Flags des Prozesses

- **04** Prozeß hat eine Floating–Point Operation ausgeführt; weil bei der Programminitialisierung durch crt0.s immer eine FPOp durchgeführt wird, ist dieses Flag bei C–Programmen immer gesetzt und wird aus diesem Grund nicht angezeigt
- **10** der Prozeß wird verfolgt (traced); das ist z. B. im Debugger der Fall

[22] Das Verzeichnis, auf dem das Prozeßdateisystem selbst aufgesetzt ist, kann in der aktuellen Version des procps nicht angegeben werden. Es erwartet das Prozeßdateisystem unter dem Verzeichnis '/proc'.

**PRI** das aktuelle Maximum an Rechenzeit (in Millisekunden), das dem Prozeß zugeteilt wird; wenn der Prozeß gerade läuft, ist das der Rest von der Zeitscheibe

**NI** ist der Nicewert des Prozesses; dieser Wert kann die Geschwindigkeit erhöhen oder verringern, in der die Zeitscheibe eines Prozesses verbraucht wird

**MAJFLT** (auch **PAGEIN**) Anzahl der "major page faults" (das sind die Versuche, auf eine ausgelagerte Seite zuzugreifen)

**MINFLT** Anzahl der "minor page faults"; diese Zugriffe hatten keine Hardwareaktion zur Folge

**TSIZ** (Textsize) die Größe des Textsegmentes der ausführbaren Datei

**DSIZ** (Datasize) die Differenz aus `vsize` (virtuelle Größe des Prozesses) und `TSIZ`

**RSS** (Resident Set Size) ist die Größe des Programms im Arbeitsspeicher; dieser Wert wird aus dem task_struct errechnet und ist nicht mit dem RSS Feld in `statm` identisch

**SIZE** der "virtuelle" Speicherbereich des Prozesses

**STACK** zeigt auf die Basis des nach unten wachsenden Stack

**LIM** zeigt den mit `ulimit -m` eingestellten Grenzwert für den Resident Stack Size

**%MEM**

**ESP** (Extended Stack Pointer) zeigt auf die Spitze des nach unten wachsenden Stack

**EIP** (Extended Instruction Pointer) zeigt auf den aktuellen Maschinenbefehl im Programmtext (¡0x60000000) oder in den Shared Libraries (¿0x60000000)

**TMOUT** zeigt den Wert eines eventuell gesetzten Timeouts

**ALARM** zeigt den Wert eines eventuell gesetzten Alarmtimers (z. B. von sleep)

**SIGNAL** zeigt ein eventuell gerade anliegendes Signal

**BLOCKED** Bitmaske der blockierten Signale

**IGNORED** Bitmaske der Signale, die ignoriert werden

**CATCHED** Bitmaske für Signale, die abgefangen werden

**WCHAN** ist der Name der Kernelfunktion, in der der Prozeß schläft[23]

Die folgenden Werte werden mit der `-m` Option angezeigt. Sie geben detailliert Auskunft über die Speicherauslastung durch die einzelnen Prozesse. Die Werte für `SIZE` und `RSS` werden auch mit anderen Optionen angezeigt, die mit der `-m` Option ausgegebenen Werte werden auf eine andere Weise berechnet.

**TRS** (Text Resident Size) Größe des Textsegments (enthält keine shared Libraries)

**DRS** (Data Resident Size, auch **DSIZ**) Größe des Datensegments (enthält benutzte Libraryseiten)

**SIZE** die Speicherbelegung des Prozesses; Text, Daten und Stack unabhängig davon, ob sie sich im physikalischen Speicher befinden

**SWAP** ausgelagerte Speicherseiten in Kilobyte (oder Seiten mit -p); ist einfach die Differenz aus `SIZE` und `RSS`

**RSS** (Resident Set Size)

**SHRD** die Größe des mit mindestens einem anderen Prozeß gemeinsam benutzten Speicherbereiches

**LIB** (Library Resdident Size) die gesamte Größe der vom Prozeß benutzten Libraryseiten im Arbeitsspeicher

**DT** (Dirty) benutzte Libraryseiten in Kilobyte (oder Seiten mit -p)

[23] Damit das procps die Kernelfunktionen benennen kann, braucht es wie das "normale" ps die Datei `/etc/psdatabase`. Diese Datei kann mit dem psupdate–Kommando aus der `/usr/src/linux/tools/system` Datei destiliert werden. Die `/etc/psdatabase` muß für jede Kernelversion neu gemacht werden. Das psupdate–Kommando ist Teil des procps Paketes.

**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** „breite" Ausgabe, kann bis zu drei mal angegeben werden; die Ausgabezeilenlänge wird auf 132 Spalten, 264 Spalten und unbegrenzt erhöht

**-x** zeigt Prozesse, die von keinem Terminal kontrolliert werden

**-S** addiert die Prozessorzeit der Kindprozesse zu den Eltern

**t** *xx* zeigt nur die Prozesse, die von Terminal *xx* kontrolliert werden

Folgende Optionen werden nur vom kernelabhängigen ps zusätzlich angeboten:

**-0** der Prozeß Nummer 0 (scheduler) wird mit angezeigt

**-f** (forest) zeigt die Prozeßfamilien mit ihren Eltern–Kind–Beziehungen als Bäume

**-H** gibt einen kurzen Hilfstext zum ps–Kommando aus

**-p** veranlaßt ps die Speicherbelegung in Seiten und nicht in Kilobytes anzuzueigen

**-U** aktualisiert die Datei /etc/psdatebase, die den Zugang zu den Kerneldaten 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[24]

**-y** zeigt den Syscall, in dem der Prozeß gerade schläft oder den er gerade verlassen hat

**Autor:** Branko Lankester, Michael K. Johnson, Rick Sladkey

## 3.59 pwd

**Funktion:**

**pwd** gibt den Namen des aktuellen Verzeichnisses aus

**Syntax:**

**pwd**

[24] Stattdessen gibt es das psupdate–Kommando, das ebenfalls eine /etc/psdatabase für das procps erzeugt.

**Beschreibung:**

**pwd** ist in vielen Shells ein integriertes Kommando (z. B. in der bash, Seite 91). Wenn in der Distribution kein binäres pwd–Kommando enthalten ist, kann als Notbehelf beispielsweise mit der tcsh, bei der kein pwd–Kommando integriert ist, das folgende Shellscript benutzt werden:

```
#!/bin/bash
pwd
```

## read

**read** ist ein eingebautes Shellkommando zum Einlesen von Daten von der Tastatur in ein Shellscript. Siehe dazu die Kommandobeschreibung auf Seite 91.

## 3.60 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 die Superuserin **Ruth**); 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

## 3.61 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 Verzeichnis im Verzeichnis gelöscht ist)

**Autor:** David MacKenzie

## 3.62 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 von einander 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 mit jeder Zeile 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 146). Darüber hinaus 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 Arbeisspeichers in die benannte *Datei*

**x** vertauscht den Inhalt des Puffers mit dem Arbeitsspeicher

**y/*Zeichenkette1*/*Zeichenkette2*/** vertauscht jedes auftretende 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 auf Seite 92 zu finden

## 3.63 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

## 3.64 sort

### Funktion:

**sort** sortiert die Zeilen einer Textdatei

### Syntax:

**sort** [−cmus] [−t *Separator*] [−o *Ausgabedatei*] [−bdfiMnr] [+*POS1* [−*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 sortieren 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 Mbdfinr 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 ist/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 nicht druckbaren 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***] bestimmt die Zeichen zwischen *POS1* und *POS2* zum Sortierschlüssel; wenn *POS2* fehlt, werden alle Zeichen bis zum Zeilenende zum Schlüssel; Positionen der Felder und Buchstaben zählen von 0

**-k *POS1*** [ **-*POS2***] bestimmt die Zeichen zwischen *POS1* und *POS2* zum Sortierschlüssel; wird das Schlüsselfeld so spezifiziert, zählen die Felder und Buchstaben von 1

Eine Position hat die Form ***feld.buchstabe***

**Autor:** Mike Haertel

## 3.65 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 Kilobyte (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 107

**Autor:** tege@sics.se

## 3.66 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, für das aktuelle Terminal aus.

Alle Anzeigen und Einstellungen beziehen sich auf die Standardeingabe. Um also ein anderes als das aktuelle Terminal zu überprüfen oder einzustellen, muß die Standardeingabe von dem entsprechenden Device umgelenkt werden.

Mit den folgenden Argumenten lassen sich die Eigenschaften des Terminals ändern. Wird einem Argument ein '−' vorangestellt, so wird die entsprechende Eigenschaft abgeschaltet bzw. in ihr Gegenteil verkehrt.

Um alle Einstellungen in ihrer vollen Bedeutung zu verstehen, muß man sich mit der Funktionsweise der asynchronen, zeichenorientierten Gerätetreiber im Kernel im allgemeinen und der seriellen Schnittstelle im speziellen befassen. Dieses Thema führt aber weit über den Rahmen dieses Buches hinaus.[25]

## Allgemeine Einstellungen:

**parenb** schickt und erwartet ein Paritätsbit bei der Datenübertragung auf der seriellen Schnittstelle

**parodd** setzt ungerade Parität (gerade mit `−parodd`)

**cs5 cs6 cs7 cs8** setzt die Zeichengröße auf 5, 6, 7 oder 8 Bits

**hupcl | hup** wenn der letzte Prozeß die Gerätedatei schließt, werden DTR und RTS zurückgesetzt und darurch ein Modem zum Unterbrechen einer eventuell noch bestehenden Telefonverbindung veranlaßt (hangup on close)

**cstopb** jedes auf der seriellen Schnittstelle übertragene Zeichen wird durch zwei Stopbits abgeschlossen (ein Stopbit mit `−cstopb`)

**cread** sollte den "Receiver" der seriellen Schnittstelle abschalten; wird von Linux ignoriert

**clocal** schaltet den "soft carrier" für die serielle Schnittstelle ein, das CD Modemkontrollsignal wird vorgetäuscht; serielle Terminals brauchen normalerweise dieses Flag, auf Modemleitungen muß dagegen der "hard carrier" eingeschaltet sein, damit das Modem die Schnittstelle kontrollieren kann

**crtscts** schaltet RTS/CTS Hardware–Handshaking zur Datenflußkontrolle auf der seriellen Schnittstelle ein

## Einstellungen für die Terminaleingabe:

**ignbrk** der Terminaltreiber ignoriert BREAK Zeichen

**brkint** der serielle Treiber erzeugt ein SIGINT wenn vom Modem ein BREAK gesendet wird

**inpck** schaltet die Paritätsprüfung in der seriellen Schnittstelle ein

**ignpar** erkannte Paritätsfehler werden bei der Terminalausgabe ignoriert

**parmrk** erkannte Paritätsfehler werden bei der Ausgabe mit einer 255–0–Zeichen Sequenz markiert

**istrip** löscht das 8. Bit der ankommenden Zeichen

**inlcr** übersetzt ankommende NEWLINE (Zeilenvorschub, `^J`) in CARRIAGE RETURN (Wagenrücklauf, `^M`)

[25] Vielleicht nur so viel:
Die asynchronen Gerätetreiber bestehen aus mehreren Schichten, von denen die unterste durch externe Ereignisse — die Hardwareinterrupts von der Tastatur, der seriellen Schnittstelle etc. — unabhängig vom aktuellen Programm, also asynchron gesteuert wird. Die oberste Ebene wird synchron mit dem Programmtext von irgendwelchen normalen Programmen benutzt. Auf beiden Ebenen können allerlei Parameter eingestellt und so die Arbeitsweise des Gerätes insgesamt verändert werden. Die einzelnen Parameter in der Kommandobeschreibung von stty sind thematisch, nicht Programmtechnisch sortiert. Das führt dazu, daß spezielle Einstellungen für den Treiber der seriellen Schnittstelle unmittelbar neben allgemeinen Einstellungen für alle Terminaltreiber beschrieben werden. Im Zweifelsfall sind die Kernelsourcen selbst immer die beste Informationsquelle.

**igncr** ignoriert ankommende CARRIAGE RETURN

**icrnl** übersetzt ankommende Wagenrücklauf in Zeilenvorschub

**iuclc** alle ankommenden Großbuchstaben werden in Kleinbuchstaben umgewandelt

**ixon** der Terminaltreiber reagiert auf XON/XOFF Flowcontrol

**ixoff | tandem** schaltet die automatische XON/XOFF Datenflußkontrolle für die seriellen Schnittstellen ein; sobald der freie Platz im Eingabedatenpuffer des Terminals die "Niedrigwassermarke" von 16 Zeichen unterschreitet, wird die STOP_CHAR(tty)–Funktion ausgelöst

**ixany** erlaubt jedes Zeichen als Startzeichen nach dem Einfrieren des Datenflusses beim XON/XOFF Flowcontrol (nur `^Q` mit `-ixany`)

**imaxbel** wird vom Linux–Kernel ignoriert

## Einstellungen für die Terminalausgabe:

**opost** schaltet die "Nachbehandlung" der ankommenden Zeichen im Terminaltreiber ein; die Nachbearbeitung der Ausgabe kann mit `-opost` komplett abgeschaltet werden

**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** Zeilenvorschub erzeugt zusätzlichen Wagenrücklauf

Die Einstellungen für verschiedenste Ausgabeverzögerungen, die im `stty`–Kommando erlaubt werden, stammen aus der Zeit, als noch mechanische Ausgabegreäte mit trägen Druckköpfen als Terminals benutzt wurden. Die Ausgabeverzögerungen verschaffen dem Drucker genügend Zeit, die Mechanik korrekt zu positionieren, bevor das nächste Zeichen gesendet wird. Die Parameter haben bei den modernen Ausgabegeräten unter Linux keine Bedeutung mehr und werden von den Terminaltreibern im Kernel nicht benutzt.

## Lokale Einstellungen:

**isig** ermöglicht Signalzeichen für Softwareinterrupts; die Zuordnung bestimmter Signale zu Eingabezeichen ist auf der nächsten Seite erklärt

**icanon** schaltet die primitiven Zeileneditorfunktionen des Terminaltreibers ein, siehe nächste Seite unter Spezialzeichen

**iexten** 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; auch diese Funktion wird vom aktuellen Linux–Kernel nicht benutzt

**echonl** ein NEWLINE wird im kanonischen Modus reflektiert, auch wenn das Echo für andere Zeichen abgeschaltet ist

**noflsh** unterdrückt das Löschen des Eingabepuffers nach einer Unterbrechung

**xcase** wird von den Treibern im Linux–Kernel ignoriert

**tostop** hält Hintergrundprozesse an, die auf das Terminal schreiben wollen

**echoctl | ctlecho** gibt Controlzeichen als Caret–Sequenz aus (`^C` für Control-C)

**echoprt | prterase** wird vom Linux–Kernel nicht bearbeitet

**echoke | crtkill** wird vom Linux–Kernel ebenfalls ignoriert

## Kombinationen von Einstellungen:

Die folgendezimen Parameter für stty werden nicht direkt an den Kernel weitergegeben. Sie sind vielmehr Zusammenfassungen häufig gebrauchter Kombinationen der bisher vorgestellten Einstellungen.

**evenp | parity** das gleiche wie `parenb −parodd cs7` (mit einem '−' wie `−parenb cs8`)

**oddp** das gleiche wie `parenb parodd c7` (mit '−' wie `−parenb cs8`)

**nl** das gleiche wie icrnl (mit '−' wie `−icrnl −inlcr −igncr`)

**ek** setzt die Löschzeichen auf ihre voreingestellten Werte

**sane** setzt alle Eistellungen 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 isigicanon iexten echo echoe echok −echonl −noflsh −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, Wörter 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 −xcasemin 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 '^−'[26] 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 Dateiendezeichen (beendet die Eingabe) (`^D`)

**eol** Zeilenende (undef)

[26] Der Ausdruck steht für die Tastenkombination CONTROL-MINUS

**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. Diese Sonderzeichen müssen selbst durch das Sonderzeichen `^V` eingeleitet werden, damit sie unverändert von der Shell an das Terminal gegeben werden.

## Spezielle Einstellungen:

**min** $N$ bestimmt die minimale Zeichenzahl zum Abbrechen eines Lesezyklus im `cbreak` Modus

**time** $N$ bestimmt die Zeit in Zehntelsekunden, nach der ein Lesezyklus im `cbreak` Modus automatisch beendet wird, auch wenn die bei '`min`' angegebene Anzahl Zeichen noch nicht gelesen ist

**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 Bit pro Sekunde); *Geschwindigkteit* 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 '`clocal`' gesetzt ist; um höhere Übertragungsraten als 38400 bps zu erzielen, muß die serielle Schittstelle mit dem setserial–Kommando umgestellt werden

**ispeed** $N$ setzt die Eingabegeschwindigkeit auf $N$ Zeichen pro Sekunde

**ospeed** $N$ setzt die Ausgabegeschwindigkeit auf $N$ Zeichen pro Sekunde

### Optionen:

**-a** zeigt alle Einstellungen in lesbarer Form

**-g** gibt alle Einstellungen in einer Form, die beim Zurücklesen (als Kommandozeilenargument) die gleichen Einstellungen reproduziert

### Beispiel:

Das folgende Kommando gibt alle Einstellungen der seriellen Schnittstelle /dev/ttyS1 aus.

```
$ stty -a < /dev/ttyS1
speed 9600 baud; rows 0; columns 0; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
eol = <undef>; eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z;
rprnt = ^R; werase = ^W; lnext = ^V; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr -icrnl
-ixon -ixoff -iuclc -ixany -imaxbel
```

```
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab3
bs0 vt0 ff0 -isig -icanon -iexten -echo -echoe -echok -echonl
-noflsh -xcase -tostop -echoprt -echoctl -echoke
$ _
```

Ein weiteres Beispiel zeigt, wie das aktuelle Terminal, das zuerst durch eine offenbar fehlerhafte Eingabe[27] durcheinandergebracht worden ist, wieder "saniert" werden kann:

```
$ monster -d terminal
$ pwd
     /usr/local/lib
                   $ cd
                       $ stty sane
                                  $ pwd
/home/she
$ _
```

**Autor:** David MacKenzie

## 3.67 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 ist, wird zur ID 0 gewechselt, man ändert die ID also in die der "Superuserin". Allein sie kann mit dem su–Kommando die Identität jedes beliebigen Benutzers 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.

Ohne weitere Optionen wechselt das su–Kommando das Arbeitsverzeichnis nicht, setzt aber die Umgebungsvariablen HOME und SHELL auf die neuen Werte aus /etc/passwd. Wenn die neue Identität nicht die der Superuserin ist, wird auch die LOGNAME Umgebungsvariable entsprechend verändert.

In keinem Fall wird von su ein Eintrag in der Datenbank /etc/utmp angelegt. Das bedeutet, daß ein "Login" mit su nicht mit last oder who angezeigt wird. Insbesondere geben auch das loginname–Kommando und die getlogin(3) Bibliotheksfunktion weiterhin den ursprünglichen Login–Namen aus.

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

[27] Das in dem Beispiel vorgestellte monster-Kommando kann beispielsweise ein `stty -onlcr` sein.

**-l** (login) die Shell wird aufgerufen wie bei einem login als *Name*; es werden alle Umgebungsvariablen außer TERM, HOME und SHELL entfernt, der Pfad auf einen einkompilierten Wert gesetzt und ein '–' als erstes Zeichen in die Kommandozeile beim Aufruf der Shell geschrieben; daraufhin startet die Shell als Loginshell mit den entsprechenen Initalisierungen; die LOGNAME Variable wird auch für die Superuserin neu gesetzt

**-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 164

**Autor:** David MacKenzie

## 3.68 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

**Siehe auch:** cksum auf Seite 103

**Autor:** Kayvan Aghaiepour und David MacKenzie

## 3.69 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 automatisch[28] 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.

## 3.70 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* zeilenweise 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 99

**Autor:** Jay Lepreau, David MacKenzie

[28] Beim simpleinit wird zu diesem Zweck das update Programm im Hintergrund gestartet. Das sysvinit führt die Synchronisation selbst durch, wenn ein entsprechender Befehl in der Datei /etc/inittab (→ Seite 37) steht

## 3.71 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** [{−,+}Nbcfklmqv] [*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

## 3.72 tar

### Funktion:

**tar** (tape archiver) verwaltet Dateiarchive

### Syntax:

**tar** [−Acdrtux] [−−delete] [−b $N$] [−BgGhiklmMoOpPPsSvwWz] [−C *Verzeichnis*] [−f *Datei*] [−F *Datei*][−K *Datei*] [−L *Länge*] [−N *Datum*] [−T *Datei*] [−V *Name*] [−X *Datei*] [0−7] [{lmh}]

**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. Das GNU tar kann aber die Ein- und Ausgabe durch einen Kompressor leiten. Die neuen Versionen (ab 1.11.2) unterstützen sowohl compress als auch gzip.

Linux kann alle SCSI Streamer und QIC–02 Streamer mit eigener Karte betreiben. Die Entwicklung von Treibern für die verbreiteten Floppystreamer macht Fortschritte, zum gegenwärtigen Zeitpunkt sind diese Geräte zur zuverlässigen Sicherung kostbarer Daten jedoch noch nicht gegeignet.

Wenn keine Datei und kein Gerät angegeben ist, versucht tar auf einen Streamer zuzugreifen. Je nach Konfiguration ist das meist /dev/tape oder /dev/rmt0.

**Optionen:**

**-A** hängt ein komplettes Archiv an ein anderes Archiv an

**-c** erzeugt ein neues Archiv

**-d** vergleicht das Archiv mit dem Dateisystem

**- -delete *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 archiviert, 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 Shellscript *Datei* 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 mitarchiviert

**-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) Label *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 gzip komprimiertes Archiv

**-Z** erzeugt ein mit compress komprimiertes Archiv

-{**0..7**}{**lmh**} spezifiziert das Gerät und die Dichte des Speichermediums (für Diskettenarchive ohne Bedeutung); 0 ist der erste Streamer, 1 der zweite und so weiter; die Dichte bestimmt den Bandtyp[29]

**Beispiel:**

**Siehe auch:**

cpio(1), mt(1), shar(1)

**Autor:** John Gilmore

## 3.73 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 compilieren 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

[29] Durch unterschiedliche Dichten wird der Streamer über verschiedene Minor Device Nummern angesprochen. Der QIC–02 Treiber unterscheidet dadurch die verschiedenen Bandformate. Minor Null ist für Streamer, die die Bandsorte selbst erkennen, 2 für QIC–11, 4 für QIC–24, 6 für QIC–120 und 8 für QIC–150.

**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 identisch mit dem test–Shellkommando (→ 94).

## 3.74 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 worden 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

**Autoren:** Paul Rubin, Arnold Robbins,
Jim Kingdon, David MacKenzie und Randy Smith

## 3.75 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 (→ Seite 96).

## 3.76 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

## 3.77 uniq

**Funktion:**

**uniq** löscht doppelte Zeilen aus einer sortierten Datei

**Syntax:**

**uniq** [−cdu] [−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 (→ Seite 156).

## vi, view

**vi** ist ein Link auf elvis, den freien vi–Clone. Die Beschreibung ist ab Seite 115 zu finden.

**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 115 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 97 des Handbuchs zu finden.

## 3.78 wc

**Funktion:**

**wc** zählt die Anzahl von Zeichen, Wörtern oder Zeilen

**Syntax:**

**wc** [−clw] [−−bytes] [−−chars] [−−lines] [−−words] [*Datei* ...]

**Beschreibung:**

**wc** zählt in jeder *Datei* oder in der Standardeingabe die Zeilen, die Wörter 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 Wörter aus

**-c** gibt nur die Anzahl der Zeichen aus

## 3.79 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 eMail 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

## 3.80 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 'mesg no' kann das eigene Terminal vor fremdem Beschreiben geschützt werden.

**Autor:** Peter Orbaek

# Kapitel 4

# Die Kommandos für root

## 4.1 chown

**Funktion:**

**chown** (change owner) ändert den Eigentümer der Datei(en)

**Syntax:**

**chown** [−Rcfv] [−−recursive] [−−show-changes] [−−silent] [−−quiet]
[−−verbose] [*Besitzerin*][:.][*Gruppe*] *Datei* ...

**Beschreibung:**

Normalerweise ist die Erzeugerin einer Datei bzw. eines Verzeichnisses auch deren/dessen Eigentümerin (die besondere Bedeutung der Eigentumsrechte ist auf Seite 230 erklärt). Mit chown kann die Systemverwalterin (ruth) die Eigentümerin und die Gruppe einer Datei oder eines Verzeichnisses ändern. *Besitzerin* und *Gruppe* können als Name oder Kennzahl angegeben werden. Name und Zahl müssen mit den entsprechenden Einträgen in `/etc/passwd` und `/etc/group` übereinstimmen.

Das chown–Kommando kann auch von Benutzerinnen ohne Privilegien ausgeführt werden, allerdings kann dann nur die Gruppe geändert werden. Weil es zu diesem speziellen Zweck das separate Kommando chgrp gibt, kann die Ausführung des chown Kommandos der Superuserin vorbehalten bleiben.

**Optionen:**

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

**Siehe auch:**

chgrp auf Seite 101 und chmod auf Seite 101

**Autor:** David MacKenzie

## 4.2 elvprsv

**Funktion:**

**elvprsv** — Rettet die Temporärdateien, die elvis in `/tmp` erstellt.

**Syntax:**

**elvprsv** [*"warum elvis abstürzte"*] **/tmp/*Dateiname...***
**elvprsv -R /tmp/*Dateiname...***

**Beschreibung:**

elvprsv sichert die letzten Änderungen in den Temporärdateien, die nach einem gewaltsamen Tod von elvis (→ S. 115) in `/tmp` verbleiben, nach `/usr/preserve` und verwaltet dort eine Indexdatei, in der die absoluten Pfadnamen der Dateien stehen, die von elvis' Absturz betroffen sind.

Auf UNIX-ähnlichen Systemen sollte es unnötig sein, elvprsv von der Kommandozeile aus zu starten. elvis startet das Programm selbst, wenn er im Begriff steht abzustürzen. Außerdem sollte das Programm bei jedem Systemstart von der Datei `/etc/rc` aus aufgerufen werden.

elvprsv überschreibt die Originaldateien nicht automatisch, sondern schickt an Benutzer, die von dem Absturz betroffen waren, eine Mail.

**Autor:** Steve Kirkendall

## 4.3 fdisk

**Funktion:**

**fdisk** ist der Partitionstabelleneditor für Linux

**Syntax:**

**fdisk** [−lv] −s *Partition*] [*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 16–Bit–RLL– oder MFM–Controller unterstützt. Mit entsprechender Kernelkonfiguration kann auch ein 8–Bit–XT–Controller verwendet werden.

**Optionen:**

**-v** gibt die Versionsnummer aus

**-l** gibt eine Liste der Partitionstabellen für alle AT–Bus– und SCSI–Festplatten aus

**-s *Partition*** wenn die angegebene *Partition* keine DOS–Partition ist, wird die Größe dieser Partition ausgegeben (dieser Wert kann beispielsweise als Eingabe für das Programm mkfs.minix verwendet werden)

**Siehe auch:**

Eine ausführliche Beschreibung des fdisk–Kommandos ist im Kapitel "Die Festplatte partitionieren" ab Seite 275 zu finden.

**Autor:** A. V. Le Blanc, Rik Faith

## 4.4 fsck (Front-End)

**Funktion:**

**fsck** steuert als Front–End die Prüfung und Reparatur aller Linux–Dateisysteme

**Syntax:**

**fsck** [–AV] [–t *Typ*] [*Optionen*] *Dateisystem*

**Beschreibung:**

**fsck** ist das von den verschiedenen Linux–Dateisystemen unabhängige Front–End zum Prüfen und Reparieren der Dateisystem–Struktur.

Als Front–End übernimmt fsck die Arbeit am konkreten Dateisystem nicht selbst, sondern ruft das spezifische File–System–Check–Programm für das angegebene Dateisystem auf. Die Zuordnung eines Dateisystemtyps zu einer bestimmten Partition findet über die Datei `/etc/fstab` oder durch eine Kommandozeilenoption statt.

Als Kommandozeilenargument kann das zu prüfende Dateisystem entweder als Gerätedatei oder durch seinen Aufsetzpunkt (mount point) angegeben werden. Im Fall der Angabe durch den Aufsetzpunkt muß die Verknüpfung mit einer bestimmten Partition in der Datei `/etc/fstab` festgelegt sein.

Nach den eigenen Optionen erlaubt das fsck–Front–End die Übergabe weiterer Optionen, die an das spezifische File–System–Check–Programm weitergegeben werden. Die Optionen `-a`, `-l`, `-r`, `-s` und `-v` werden von den meisten dieser Programme unterstützt, Einzelheiten finden Sie bei den entsprechenden Kommandobeschreibungen.

Um sie automatisch aufrufen zu können, erwartet das fsck–Front–End die spezifischen File–System–Checker unter Namen der Art `fsck.`*Typ*, also beispielsweise `fsck.ext2` für Extended–2–Dateisysteme oder `fsck.minix` für "alte" Minix–Dateisysteme. Der File–System–Standard sieht das Verzeichnis `/sbin` als Installationsverzeichnis für all diese Programme vor. Das fsck–Front–End sucht aber auch in den Verzeichnissen `/etc/fs` und `/etc` nach den Programmdateien.

Der Name fsck für das Front–End leitet sich Unix–typisch aus der von Vokalen befreiten Abkürzung für *file system check* ab. Zu Beginn der Linux–Entwicklung gab es als einziges das von Minix übernommene Dateisystem, deshalb hat das Kommando zur Prüfung eines Minix–Dateisystems ursprünglich den Namen fsck gehabt. Mit der Einführung des Front–Ends muß das alte fsck seinen angestammten Namen zugunsten der einheitlichen Namenskonvention abgeben.

Um die Zusammenarbeit mit anderen Programmen zu erleichtern, gibt das fsck–Front–End über den Exit–Status verschiedene Informationen zum Zustand des Dateisystems und zur Programmausführung weiter:

**0** Es wurde kein Fehler im Dateisystem gefunden.

**1** Es wurden Fehler im Dateisystem gefunden und korrigiert.

**2** Es wurden schwerwiegende Fehler im Dateisystem gefunden und korrigiert. Das System sollte rebootet werden.

**4** Es wurden Fehler im Dateisystem gefunden, aber nicht korrigiert.

**8** Es ist ein Fehler bei der Programmausführung aufgetreten.

**16** Falsche Benutzung (Fehler in der Kommandozeile?).

**128** Es ist ein Fehler in einer Funktion der Shared–Librarys aufgetreten.

Diese Fehlercodes können beispielsweise in einem `rc`–Shellscript zur Systeminitialisierung abgefragt werden, um damit den weiteren Ablauf des Programms zu bestimmen. Wenn mit der Option `-A` mehrere Dateisysteme geprüft werden, ist der Status des fsck–Front–Ends die logische ODER–Verknüpfung aller Rückgabewerte der einzelnen File–System–Checker.

**Optionen:**

**-A** (All) veranlaßt fsck, alle in der Datei `/etc/fstab` aufgeführten Linux–Dateisysteme zu prüfen; diese Option schließt die Option `-t` zur direkten Angabe eines Dateisystemtyps aus, stattdessen werden die Typen aus der `fstab` gelesen

**-V** veranlaßt fsck, Informationen über den Programmverlauf auszugeben; wenn diese Option mehr als einmal angegeben ist, werden die spezifischen File–System–Checker nicht wirklich ausgeführt

**-t** ***Typ*** spezifiziert den Dateisystemtyp der zu prüfenden Partition; wenn diese Option nicht angegeben ist, versucht das fsck–Front–End, den Typ aus der Datei `/etc/fstab` zu bestimmen

**Siehe auch:**

mkfs Seite 203, fsck.minix Seite 199, fsck.ext2 Seite 198, fsck.xiafs Seite 200

**Autor:** David Engel, Fred van Kempen

## 4.5 fsck.ext2 (e2fsck)

**Funktion:**

**e2fsck** überprüft und repariert ein ext2fs–Dateisystem

**Syntax:**

**e2fsck** [−acdflmrstv] [−B *Nummer*] [−b *Datei*] *Gerätedatei*

**Beschreibung:**

**e2fsck** ist das Wartungsprogramm für Extended–2–Dateisysteme. Es prüft die Konsistenz des Dateisystems und repariert Fehler. Für die Zusammenarbeit mit dem neuen fsck–Front–End muß das Programm unter dem Namen fsck.ext2 installiert sein.[1] In der Kommandobeschreibung steht e2fsck synonym für fsck.ext2.

Die *Gerätedatei* gibt die Partition an, auf der das Dateisystem geprüft werden soll. Wenn diese Partition kein ext2fs enthält, bricht das Kommando automatisch ab.

Wenn das valid–Flag der ext2fs–Partition gesetzt ist, wird das Dateisystem normalerweise nicht geprüft (→ Seite 295).

Dateien, deren Inodes in keinem Verzeichnis eingetragen sind, werden von e2fsck im Verzeichnis `lost+found` mit der Inode–Nummer eingetragen und können so "gerettet" werden.[2]

Um die Zusammenarbeit mit anderen Programmen zu erleichtern, gibt e2fsck über den Exit–Status verschiedene Informationen weiter:

**0** Kein Fehler im Dateisystem gefunden.

[1] In Distributionen, die dem File–System–Standard nicht angepaßt sind, ist das Programm in `/etc` oder `/etc/fs` installiert, sonst in `/sbin`.

[2] Das Verzeichnis `lost+found` wird von mke2fs (→ Seite 204) automatisch angelegt. Wenn dieses Verzeichnis versehentlich gelöscht wurde, muß es mit mklost+found wieder angelegt werden.

**1** Fehler im Dateisystem gefunden und korrigiert.

**2** Schwerwiegende Fehler im Dateisystem gefunden und korrigiert. das System sollte rebootet werden, wenn das Dateisystem aufgesetzt war.

**4** Fehler im Dateisystem gefunden und unkorrigiert gelassen.

**8** Fehler bei der Programmausführung aufgetreten.

**16** Falsche Benutzung (Fehler in der Kommandozeile?).

**128** Fehler in den Shared–Librarys.

Diese Fehlercodes können beispielsweise in einem Shellscript abgefragt werden, um damit den weiteren Ablauf des Programms zu bestimmen.

**Optionen:**

**-a** (automatic) repariert alle gefundenen Fehler automatisch (ohne Nachfrage, daher mit Vorsicht anzuwenden)

**-B** *Nummer* liest den Superblock aus dem mit der *Nummer* bezeichneten Partitionsblock; beim ext2fs werden Sicherungskopien des Superblocks normalerweise alle 8192 Blöcke angelegt, sinnvolle Optionsargumente sind also `-B 8193` oder `-B 16385`

**-b** ***Datei*** liest die Liste schlechter Blöcke (*bad blocks*) aus der angegebenen *Datei*; diese Blöcke werden automatisch als benutzt markiert

**-c** (check) durchsucht das Dateisystem nach schlechten Blöcken

**-d** (debug) gibt zusätzliche Information über den Status des Programms auf die Standardfehlerausgabe

**-f** (force) erzwingt die Überprüfung eines Dateisystems, auch wenn das valid–Flag gesetzt ist (→ Seite 295)

**-l** (list) gibt alle Dateinamen aus dem Dateisystem aus

**-m** (mounted) verhindert die Überprüfung eines aufgesetzten Dateisystems

**-r** (repair) repariert ein ext2fs interaktiv (jede Aktion muß bestätigt werden)

**-s** (superblock) gibt den Superblock aus, bevor das Dateisystem geprüft wird

**-t** (test) durchsucht das Dateisystem nach schlechten Blöcken (wie –c)

**-v** (verbose) gibt Information über den Status des Programms und über das Dateisystem aus

**Siehe auch:**

mkfs.ext2 Seite 204, fsck Seite 197

**Autor:** Remy Card, Linus Torvalds und Wayne Davidson

## 4.6 fsck.minix (fsck)

**Funktion:**

**fsck.minix** prüft die Konsistenz eines Minix–Dateisystems

**Syntax:**

**fsck.minix** [-larvsm] *Gerätedatei*

**Beschreibung:**

fsck.minix überprüft ein Minix–Dateisystem auf Blöcke, die belegt, aber keiner Datei zugeordnet sind, und auf korrekte Vernetzung der Inodes. Wenn ein Fehler festgestellt wird, kann er mit fsck.minix auch repariert werden.

Fehler im Dateisystem entstehen leicht durch unkontrolliertes Abschalten des Rechners. Dabei gehen die letzten Veränderungen des Dateisystems verloren, die aus dem Speicher nur bei einem sync auf die Festplatte geschrieben werden.

fsck.minix kann die Integrität des Dateisystems meistens wiederherstellen, allerdings gehen dabei die Daten der betroffenen Dateien verloren. Damit die Veränderungen am Dateisystem auch tatsächlich wirksam werden, darf fsck.minix mit den Optionen `-a` und `-r` nur auf abgesetzte Dateisysteme angewandt werden. Anderenfalls würden die reparierten Strukturen bei der nächsten Synchronisierung wieder durch die defekten aus dem Blockdepot im Arbeitsspeicher ersetzt werden.

Das fsck.minix–Programm gibt einige Informationen über den Zustand des Dateisystems als Exit–Status an das aufrufende Programm zurück:

**0** Kein Fehler im Dateisystem gefunden.

**3** Fehler im Dateisystem gefunden und korrigiert. Das System sollte rebootet werden, wenn das Dateisystem aufgesetzt war.

**4** Fehler im Dateisystem gefunden und unkorrigiert gelassen.

**8** Fehler bei der Programmausführung aufgetreten.

**16** Falsche Benutzung (Fehler in der Kommandozeile?).

Mit diesen Statuswerten können beispielsweise in einem Shellscript die weiteren Programmschritte gesteuert 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 aus

**-s** (superblock) zeigt den Superblock des Dateisystems

**-m** (mode) gibt "Mode not cleared"–Warnungen aus, wozu auch immer

**Autor:** Linus Torvalds

## 4.7 fsck.xiafs (xfsck)

**Funktion:**

**fsck.xiafs** testet und repariert das xiafs Dateisystem

**Syntax:**

**fsck.xiafs** [-akrs] *Gerätedatei*

**Beschreibung:**

**fsck.xiafs** (**xfsck**) checkt das xiafs. Das Programm findet Fehler im Dateisystem und kann sie reparieren. Es werden fehlerhafte Verzeichniseinträge erkannt und gelöscht, doppelt belegte Blöcke gefunden und fehlerhafte Linknummern in Inodes korrigiert.

Um mit dem fsck–Front–End zusammenarbeiten zu können, muß das ursprünglich unter dem Namen xfsck verbreitete Kommando (auch) unter dem Namen fsck.xiafs installiert sein. Das Verhalten des Programms ändert sich durch die Umbenennung nicht.

**Optionen:**

**-a** (automatic) repariert die gefundenen Fehler automatisch

**-k** (kernel image) liest das Kernelimage aus dem reservierten Systembereich und schreibt es in die Standardausgabe

**-r** (repair) repariert das Dateisystem interaktiv; jede Aktion muß einzeln bestätigt werden

**-s** (super block) prüft den Superblock und zeigt dessen Inhalt an

**Siehe auch:**

fsck Seite 197 und e2fsck Seite 198

**Autor:** Frank Q. Xia

## 4.8 insmod

**Funktion:**

**insmod** fügt zur Laufzeit Kernel–Module ein

**Syntax:**

**insmod** *Objektdatei*

**Beschreibung:**

**insmod** fügt in den Linux–Kernel zur Laufzeit Module ein. Mit diesen Laufzeitmodulen können einzelne Gerätetreiber in den Kernel integriert werden, ohne dafür den ganzen Kernel neu zu linken (oder gar neu zu übersetzen). Die in den Modulen definierten Kernelfunktionen können wie die fest in den Kernel gelinkten Funktionen benutzt werden, insbesondere werden sie im Kernel–Modus ausgeführt.

Die Kernelmodule sind normale Objektdateien, wie sie vom C–Compiler mit der −**c**–Option erzeugt werden. In der Objektdatei müssen die beiden Funktionen

```
int init_module(void);
void cleanup_module(void);
```

definiert werden, mit denen das Modul initialisiert bzw. aufgeräumt/entfernt werden kann. Den Laufzeitmodulen stehen alle globalen Datenstrukturen und Funktionen des Kernels zur Verfügung.

Wenn Sie eigene Module schreiben wollen, sollten Sie sich die Quelltexte zum Floppy–Tape–Treiber als Beispiel ansehen.

**Siehe auch:**

rmmod auf S. 211

**Autor:** Anonym

## 4.9 mkboot

### Funktion:

**mkboot** schreibt ein Kernelimage in den reservierten Systembereich einer xiafs–Partition und macht sie bootfähig

### Syntax:

**mkboot** [–fr *Rootpartition*] *Bootpartition*
**mkboot** `-M` *Festplatte* `[`–*Nummer Betriebssystemname*`]`

### Beschreibung:

**mkboot** installiert ein Kernelimage in dem reservierten Systembereich einer xiafs Partition und macht sie bootfähig. Das Kernelimage wird von der Standardeingabe gelesen.

Das xiafs kann den Bereich am Anfang einer Partiton als Systembereich reservieren. Mit dem mkboot–-Kommando kann ein Kernelimage in diesen Bereich geschrieben werden. Außerdem wird von mkboot ein Bootloader in den Bootsektor dieser Partition geschrieben. Damit ist diese Partition bootfähig. Um sie aktivieren zu können, muß es sich um eine primäre Partition der ersten Festplatte handeln.

Die Bootpartition muß nicht mit der Rootpartition übereinstimmen. Es ist sogar möglich, eine minimal kleine Partition als (primäre) Bootpartition einzurichten, ohne daß ein Dateisystem darauf eingerichtet wird.

Das mkboot–Programm kann auch den Masterbootrecord (MBR) der Festplatte verändern. Der MBR enthält neben der Partitionstabelle für die vier primären Partitionen einen Loader, der vom BIOS automatisch bei jedem Systemstart aufgerufen wird (wenn nicht von Diskette gebootet wird). Der vom MS–DOS bekannte MBR sucht in der Partitionstabelle die aktive Partition und startet den Bootloader, der seinerseits das Betriebssystem lädt. Mit mkboot kann anstelle dieses MBR ein Bootselector installiert werden, der sich beim Systemstart mit einem kleinen Menü meldet und die Auswahl zwischen mehreren Bootpartitionen ermöglicht.

### Optionen:

**-f** ***Rootpartition Bootpartition*** installiert den Kernel auf der Bootpartition (mit xiafs); der Kernel benutzt die angegebene *Rootpartition* als Rootfilesystem

**-r** ***Rootpartition Bootpartition*** schreibt den Kernel auf die rohe *Bootpartition*(ohne xiafs); der Kernel benutzt die *Rootpartition* als Rootfilesystem

**-M** ***Festplatte*** [ ***-Partitionsnummer Name***. . .] schreibt einen neuen Masterbootrecord (MBR) auf die *Festplatte*; die *Partitionsnummer* bezeichnet die (primäre) Partition, der *Name* ist der Eintrag im Bootmenü

Das mkboot–Kommando kann auch den normalen DOS MBR schreiben. Dazu muß mit der Option `-M` nur die Festplatte ohne Partitionsnummer und Name angegeben werden.

### Beispiel:

Mit den Kommandos

```
# mkxfs -k 512 /dev/hda2 81920
# mkboot /dev/hda2 < zImage
# _
```

wird auf der 80 Megabyte großen zweiten Partition der ersten Festplatte (`/dev/hda2`) ein xiafs–Dateisystem eingerichtet und ein Systembereich von 512 Kilobytes reserviert. Danach wird die Kerneldatei `zImage` in diesem Systembereich installiert und die Partition bootfähig gemacht (aber nicht aktiviert).

Mit dem Kommando

```
# mkboot -M /dev/hda  -2 LINUX -1 MS-DOS
# _
```

wird der Masterbootrecord der ersten Festplatte ersetzt. Der Bootselector von Frank Q. Xia kann danach sowohl die zweite Partition mit Linux als auch die erste Partition mit MS–DOS booten. Wenn der Rechner das nächste Mal von Festplatte bootet, erscheint ein Menü mit diesen Einträgen. Wenn nach 15 Sekunden keine Auswahl getroffen wurde, benutzt der Loader automatisch den ersten Eintrag, in diesem Fall also LINUX auf der zweiten Partition.

**Siehe auch:**

Die Installation von LILO, Seite 216

**Autor:** Frank Q. Xia

## 4.10 mkfs (Front-End)

**Funktion:**

**mkfs** steuert als Front–End die Einrichtung von Linux–Dateisystemen

**Syntax:**

**mkfs** [-V] [-t *Typ*] [*Optionen*] *Dateisystem*

**Beschreibung:**

**mkfs** ist das von dem konkreten Dateisystemtyp unabhängige Front–End zur Erzeugung eines Dateisystems auf einer Festplattenpartition oder auf einer formatierten Diskette.

Als Front–End übernimmt mkfs die tatsächliche Erzeugung des Dateisystems nicht selbst, sondern ruft das spezifische Programm zur Einrichtung des geforderten Dateisystems auf. Die Zuordnung eines Dateisystemtyps zu einer Festplattenpartition findet über eine Kommandozeilenoption oder durch die Datei `/etc/fstab` statt.

Als Kommandozeilenargument kann das zu prüfende Dateisystem entweder als Gerätedatei oder durch seinen Aufsetzpunkt (mount point) angegeben werden. Im Fall der Angabe durch den Aufsetzpunkt muß die Verknüpfung mit einer bestimmten Partition in der Datei `/etc/fstab` festgelegt sein.

Nach den eigenen Optionen erlaubt das mkfs–Front–End noch die Angabe weiter Optionen, die an das spezifische Programm zur Erzeugung des Dateisystems weitergegeben werden. Die Optionen `-c`, `-l` und `-v` werden von den meisten dieser Programme unterstützt, Einzelheiten finden Sie bei den entsprechenden Kommandobeschreibungen.

Um sie automatisch aufrufen zu können, erwartet das mkfs–Front–End die spezifischen Programme zur Erzeugung der Dateisysteme unter Namen der Art `mkfs.`*Typ*, also beispielsweise `mkfs.ext2` für Extended–2–Dateisysteme oder `mkfs.minix` für "alte" Minix–Dateisysteme. Der File–System–Standard sieht das Verzeichnis `/sbin` als Installationsverzeichnis für all diese Programme vor. Das fsck–Front–End sucht aber auch in den Verzeichnissen `/etc/fs` und `/etc` nach den Programmdateien.

Der Name mkfs für das Front–End leitet sich Unix–typisch aus der von Vokalen befreiten Abkürzung für *make file system* ab. Zu Beginn der Linux–Entwicklung gab es als einziges das von Minix übernommene Dateisystem, deshalb hat das Kommando zur Erzeugung eines Minix–Dateisystems ursprünglich den Namen mkfs gehabt. Mit der Einführung des Front–Ends muß das alte mkfs seinen angestammten Namen zugunsten der einheitlichen Namenskonvention abgeben.

Um die Zusammenarbeit mit anderen Programmen zu erleichtern, gibt das mkfs–Front–End über den Exit–Status verschiedene Informationen zum Zustand des Dateisystems und zur Programmausführung weiter:

**0** Die Erzeugung des Dateisystems konnte ohne Fehler durchgeführt werden.

**8** Es ist ein Fehler bei der Programmausführung aufgetreten.

**16** Es ist ein Fehler bei der Benutzung aufgetreten (Fehler in der Kommandozeile?).

**128** Es ist ein Fehler in einer Funktion der Shared–Librarys aufgetreten.

### Optionen:

**-V** veranlaßt mkfs, Informationen über den Programmverlauf auszugeben; wenn diese Option mehr als einmal angegeben ist, wird das spezifische Programm zur Erzeugung eines Dateisystems nicht wirklich ausgeführt

**-t** ***Typ*** spezifiziert den Dateisystemtyp des zu erzeugenden Dateisystems; wenn diese Option nicht angegeben ist, versucht das fsck–Front–End, den Typ aus der Datei `/etc/fstab` zu bestimmen; wenn kein spezifisches Dateisystem bestimmt werden kann, wird ein Minix–Dateisystem erzeugt

### Siehe auch:

mkfs.minix Seite 205, mkfs.ext2 Seite 204, mkfs.xiafs Seite 206

**Autor:** David Engel, Fred van Kempen

## 4.11 mkfs.ext2 (mke2fs)

### Funktion:

**mke2fs** erzeugt ein neues, erweitertes (ext2fs) Dateisystem auf einem formatierten Datenträger[3]

### Syntax:

**mke2fs** [−ctv] [−b *Größe*] [−f *Größe*] [−i *Anzahl*] [−l *Datei*] [−m *Verhältnis*] *Gerätedatei* [*Größe*]

### Beschreibung:

Das **mke2fs**–Kommando legt auf einer Festplattenpartition ein Dateisystem an. Für die Zusammenarbeit mit dem neuen mkfs–Front–End muß das Programm (auch) unter dem Namen mkfs.ext2 installiert sein. [4] In der Kommandobeschreibung steht mke2fs synonym für mkfs.ext2, das Verhalten des Programms ändert sich mit dem Namen nicht.

Der Unterschied zum Minix–Dateisystem besteht für den Anwender vor allem darin, daß das ext2fs Partitionen bis zu 2 Gigabytes und Dateinamen mit einer Länge von bis zu 255 Zeichen zuläßt. Eine genauere Beschreibung des ext2fs ist im Anhang zu finden (→ Seite 294).

In zukünftigen Versionen des erweiterten Dateisystems werden Blockgrößen von mehr als 1 Kilobyte wählbar sein; in der aktuellen Version enthält ein Block immer 1024 Bytes. Ebenso wird die geplante Fragmentierung der Blöcke noch nicht unterstützt.

[3] Das ext2fs hat das extfs abgelöst.

[4] In Distributionen, die dem File–System–Standard nicht angepaßt sind, ist das Programm in `/etc` oder `/etc/fs` installiert, sonst in `/sbin`.

**Optionen:**

**-b** ***Größe*** (blocks) bestimmt die Größe der Datenzonen (wird noch nicht unterstützt)

**-c** (check) überprüft die Partition auf defekte Blöcke

**-f** ***Fragmentgröße*** bestimmt die *Fragmentgröße* zur Fragmentierung der Datenzonen (diese Option wird noch nicht unterstützt)

**-g** ***Größe*** ändert die *Größe* der Datengruppen auf den angegebenen Wert (Voreinstellung sind 8196 Blöcke pro Gruppe)

**-i** ***Anzahl*** setzt die Anzahl Bytes pro Inode fest; für jeweils *Anzahl* Bytes Plattenplatz wird eine Inode erzeugt (Mindestwert 1024, Voreinstellung 4096)

**-l** ***Datei*** liest die Nummern der defekten Blöcke aus der *Datei*

**-m** ***Anteil*** setzt den *Anteil* der Datenblöcke, die für den Superuser reserviert sind (Voreinstellung 5%); diese Einstellung kann auch nachträglich mit dem tune2fs–Programm verändert werden

**-t** (test) Überprüft die Partition auf defekte Blöcke

**-v** (verbose) das Programm gibt sich informativ

**Autor:** Linus Torvalds, Remy Card und Adam Richter

## 4.12 mkfs.minix (mkfs)

**Funktion:**

**mkfs.minix** erzeugt ein Dateisystem im Minix–Format auf einem formatierten Datenträger

**Syntax:**

**mkfs.minix** [−c] [−l *Datei*] [−n*Zahl*] *Gerätedatei Größe*

**Beschreibung:**

Mit Hilfe von **mkfs** wird auf leeren, formatierten Datenträgern (Disketten oder Festplatten) eine Struktur zur Verwaltung von Dateien und Verzeichnissen – das Minix–Dateisystem – angelegt.

Das Format des von mkfs angelegten Dateisystems ist identisch mit dem Dateisystem von A. Tanenbaum's Lehrbetriebssystem. Deshalb ist die Typbezeichnung weiterhin 'minix'. Eine genaue Beschreibung der Struktur dieses Dateisystems ist im Anhang ab Seite 287 zu finden.

Die *Gerätedatei* ist der Eintrag der entsprechenden Partition im Verzeichnis `/dev`. Der Name setzt sich aus der Bezeichnung für die gesamte (rohe) Festplatte (wie er beim fdisk–Kommando auf Seite 196 beschrieben ist) und der Partitionsnummer zusammen. Die zweite Partition der ersten AT–Bus–Platte wird beispielsweise als `/dev/hda2` angesprochen; die erste Partition der zweiten SCSI–Festplatte heißt entsprechend `/dev/sdb1`.

Die *Größe* wird in Blöcken angegeben. Jeder Block ist ein Kilobyte groß, die genaue Zahl kann der von fdisk angezeigten Partitionstabelle entnommen werden.

Das mkfs–Kommando legt einen Superblock, Bitmaps für Inodes und Datenzonen 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 Sektoren; jeder Block wird einmal beschrieben, wieder gelesen und verglichen; fehlerhafte Blöcke werden automatisch von einer `.badblocks`–Datei belegt

**-l** ***Datei*** veranlaßt mkfs.minix, die Nummern der defekten Blöcke aus der *Datei* zu lesen

**-n***Zahl* [5] stellt die maximale Länge für Dateinamen in dem erzeugten Dateisystem auf die angegebene Zahl; zulässige Werte sind nur 14 oder 30

**Autor:** Linus Torvalds

## 4.13 mkfs.xiafs (mkxfs)

**Funktion:**

**mkxfs** legt ein xiafs auf einer Diskette oder Festplattenpartition an

**Syntax:**

**mkxfs** [−c | −l *Datei*] [−k *Größe*] [−z *Größe*] *Gerät Größe*

**Beschreibung:**

**mkxfs** legt auf einer formatierten Diskette oder auf einer Festplattenpartition *Gerät* ein xiafs (Xia–Filesystem) an. Die *Größe* muß in Kilobytes angegeben werden. Für die Zusammenarbeit mit dem neuen mkfs–Front–End muß das Programm (auch) unter dem Namen mkfs.xiafs installiert sein. In der Kommandobeschreibung steht mkxfs synonym für mkfs.xiafs, das Verhalten des Programms ändert sich mit dem Namen nicht.

Das Verhältnis von Inodes zu Datenzonen ist mit 1:4 unveränderlich festgelegt.

Eine genauere Beschreibung des xiafs ist im Anhang auf Seite 297 zu finden.

**Optionen:**

**-c** (check) sucht nach defekten Blöcken auf dem Datenträger und kennzeichnet sie automatisch als belegt

**-l** ***Datei*** liest eine Liste defekter Blöcke aus der *Datei*, um sie als belegt zu markieren

**-k** ***Größe*** reserviert einen Systembereich der angegebenen *Größe* für den Kernel; mit dem mkboot–Kommando kann eine so eingerichtete xiafs–Partition bootfähig gemacht werden

**-z** ***Größe*** richtet Datenzonen der angegebenen *Größe* ein (diese Option wird noch nicht unterstützt)

**Siehe auch:**

mkfs Seite 203, mke2fs Seite 204, mkboot Seite 202, xfsck Seite 200

**Autor:** Frank Q. Xia

## 4.14 mknod

**Funktion:**

**mknod** erzeugt eine Spezialdatei

[5] In Abweichung vom POSIX–Standard für die Interpretation der Kommandozeilenoptionen wird diese Option nur dann korrekt behandelt, wenn kein Leerzeichen zwischen dem Buchstaben und der Zahl steht.

**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** unbenutzt
**8** SCSI–Festplatte
**9** (SCSI–) Bandlaufwerk (character)
**10** Busmaus
**11** (SCSI–) CD–ROM
**12** Mitsumi CD–ROM (block); QIC–02 Tape (character)
**13** XT Festplatte mit 8–Bit Controller
**14** Soundkarte
**15** Joystick
**16** Socket
**17** AF_UNIX
**18** AF_INET
**19** WE–driver
**20** DP8390–driver
**21** Sony CD–ROM
**22** zweiter IDE Controller

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 aktuellen umask des aufrufenden Prozesses gebildet.
Der erste Buchstabe nach dem *Namen* gibt den Typ der Datei an:

**p** (pipe) erzeugt eine FIFO Spezialdatei (wie mkfifo, → Seite 160)
**b** (block) erzeugt eine Gerätedatei für ein (gepuffertes) blockorientiertes Gerät
**c** (character) erzeugt eine ungepufferte Gerätedatei für ein zeichenorientiertes Gerät
**u** (unbuffered) das Gleiche wie c

**Optionen:**

**-m** ***Modus*** setzt die Rechte der Dateien auf *Modus* wie bei chmod (→ Seite 101)

**Autor:** David MacKenzie

## 4.15 mkswap

**Funktion:**

**mkswap** bereitet eine (Geräte–) Datei für die Auslagerung von Prozessen vor

**Syntax:**

**mkswap** [−c] *Datei Blöcke*

**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 *Blöcke* ist die Größe in 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.[6]

**Beispiel:**

```
# dd bs=1024 if=/dev/zero of=/dev/swapfile count=5120
5120+0 records in
5120+0 records out
# mkswap -c /dev/swapfile 5120
Setting up swapspace, size = 5238784 bytes
# sync
# swapon /dev/swapfile
# _
```

Diese Kommandofolge erzeugt eine Swapdatei von 5 Megabyte im Verzeichnis `/dev`.

Die Speicherverwaltung 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** (check) überprüft die Partition auf defekte Blöcke

**Autor:** Linus Torvalds

[6] Es ist nicht ratsam, eine Swapdatei beispielsweise durch Kopieren aus einer existierenden normalen Datei zu erzeugen. Solche Dateien können Löcher (holes) enthalten, die zwar bei der Dateigröße angezeigt werden, die aber keine Blöcke auf der Festplatte belegen. In so einem Fall würde der Versuch, Speicherseiten auf diese Adressen auszulagern, zwangsläufig zu einem Systemfehler führen.

## 4.16 mount

### Funktion:

**mount** setzt das Dateisystem zusammen

### Syntax:

**mount** [−afnprwuv] [−t *Typ*] [−o *Schalter*] [*Gerätedatei*] [*Verzeichnis*]

### Beschreibung:

**mount** fügt die einzelnen Dateisysteme auf den verschiedenen Massenspeichern zu einem einzigen Dateisystembaum zusammen. Linux unterstützt verschiedene Arten von Dateisystemen.

**ext** das alte erweiterte Dateisystem

**ext2** das neue erweiterte Dateisystem

**hpfs** das OS/2 Dateisystem (Read–Only)

**iso9660** das CD–Dateisystem (mit Rockridge–Extensions)

**minix** der Klassiker in leichter Variation

**msdos** das Feilsüstem für Dosen

**nfs** Network–File–System

**proc** Prozeßdateisystem

**swap** Swap–Space (auch Dateien)

**xiafs** das Dateisystem von Frank Xia

Weitere Information finden Sie bei der Beschreibung der Datei `/etc/fstab` auf Seite 34 und im Kapitel über Dateisysteme, ab Seite 287.

Die einzelnen Partitionen und Laufwerke sind als Gerätedateien (special–files) im Ordner `/dev` abgebildet. Die auf den Partitionen gespeicherten Dateisysteme werden durch den mount Befehl auf beliebige leere Verzeichnisse aufgesetzt.

mount erstellt eine Liste der aufgesetzten Dateisysteme in `/etc/mtab`.[7] Wenn es ohne Kommandozeilenargumente aufgerufen wird, zeigt es den Inhalt dieser Liste an. Wenn ein mount–Befehl unvollständig angegeben wird, werden die fehlenden Parameter automatisch aus der Datei `/etc/fstab` ergänzt.

Es gibt (mindestens) zwei verschiedene Varianten des mount–Kommandos für Linux. Das andere, ursprünglich von Doug Quale beigesteuerte, ist vielfach ergänzt und verbessert worden und findet zusammen mit dem Boot–Utility–Paket von Stephen Tweedie große Verbreitung. Das andere, von Peter Orbaek geschriebene, ist auch als usermount–Kommando bekannt, weil es Benutzerinnen ohne Privilegien das Auf– und Absetzen von Dateisystemen erlaubt.

### Optionen:

**-a** alle Dateisysteme werden automatisch wie in `/etc/fstab` beschrieben zusammengefügt

**-f** (fake) führt alle Schritte des Befehls mit Ausnahme des eigentlichen Systemaufrufs aus; diese Option ist sinnvoll im Zusammenhang mit der Option -v

[7] Es gibt eine Erweiterung von Fred Baumgarten (→ iX 12/93, S. 170) für das Prozeßdateisystem, mit der eine Datei `/proc/mtab` erzeugt werden kann. Indem ein symbolischer Link von `/etc/mtab` auf die Datei in `/proc` angelegt wird, kann mit einem leicht veränderten mount–Kommando das Root–Filesystem Read–Only gemountet werden. Dazu müssen noch alle anderen Dateien aus dem Root–Filesystem, in die eventuell während des Betriebs geschrieben werden kann, in den `/var`–Ast des Verzeichnisbaums verlegt werden. Das dazugehörende Dateisystem muß dann unmittelbar nach dem Systemstart aufgesetzt werden.

**-n** unterdrückt den Eintrag des Dateisystems in die Datei `/etc/mtab` (nur usermount)

**-o** ***Schalter*** ... bestimmt mit einer durch Komma getrennten Liste von *Schaltern* die folgenden Eigenschaften des Dateisystems:

**noexec** verbietet die Ausführung von Programmen aus diesem Dateisystem

**nosuid** unterdrückt die Wirkung der SUID und SGID–Bits

**nodev** die zeichenorientierten und blockorientierten Gerätedateien werdennicht angesprochen

**remount** veranlaßt das mount–Kommando, den Status eines bereits aufgesetzten Dateisystems zu ändern; der einzige Parameter, der mit dieser Option verändert werden kann, ist das READ–ONLY–Flag

**sync** alle Lese/Schreibaktionen werden ungepuffert (synchron) ausgeführt (nur beim ext2fs implementiert)

**-p** (print) der Inhalt der Mounttabelle `/etc/mtab` wird angezeigt

**-r** (read–only) veranlaßt das mount–Kommando, das Dateisystem Read–Only aufzusetzen

**-t** ***Typ*** spezifiziert den Typ des Dateisystems, das aufgesetzt werden soll; zusammen mit der `-a` Option können auch bestimmte Dateisystemtypen ausgeschlossen werden, indem anstelle des Typs die Angabe `no`*Typ* (z.B. `nomsdos` oder `nonfs`) gemacht wird

**-u** (usermount) anstelle der Datei `/etc/fstab` wird `/etc/ufstab` benutzt; wenn das usermount–Kommando nicht von der Systemverwalterin aufgerufen wird, ist das die Voreinstellung

**-v** (verbose) veranlaßt das mount–Kommando, Informationen zum Programmverlauf auf den Bildschirm zu schreiben

**-w** (writable) veranlaßt das mount–Kommando, das Dateisystem mit Read–Write Erlaubnis aufzusetzen (nur bei Doug Quale's mount)

**Autor:** Doug Quale oder Peter Orbaek

## 4.17 rdev

### Funktion:

**rdev** (root device) zeigt und verändert einige Kernelkonfigurationen

### Syntax:

**rdev** [–help] [–R *Image* {0,1}] [–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 Rootfilesystem neu festzulegen, ohne den Kernel neu zu übersetzen.

Die durch das rdev–Kommando veränderbaren Parameter werden in der Kerneldatei an einer bestimmten Stelle gespeichert. Die genaue Position kann durch einen *Offset* manipuliert werden. Die Voreinstellung ist 508 Bytes. In den auf diese Dateiposition folgenden Bytepaaren befinden sich die veränderbaren Parameter.

Die Kerneldatei kann sich entweder in einem Dateisystem befinden (zum Beispiel in der Datei `/Image`, `/zImage`, `/vmlinux`, `/usr/src/linux/Image`), oder auf einer rohen Bootdiskette (zum Beispiel `/dev/fd0`)

**Optionen:**

**-R** ***Image*** {**0,1**} ändert das "Root-Flag" im Kernelimage; wenn das Flag auf 1 gesetzt ist, wird das Root–Filesystem Read–Only aufgesetzt

**-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 die RAM–Disk–Größe oder setzt sie auf den angegebenen Wert

Durch Links auf die Kommandonamen **ramsize** und **vidmode** können die Optionen `-r` beziehungsweise `-v` voreingestellt werden.

**Autor:** Werner Almesberger und Peter MacDonald

## 4.18 rmmod

**Funktion:**

**rmmod** entfernt ein Laufzeitmodul aus dem Kernel

**Syntax:**

**rmmod** *Modul*

**Beschreibung:**

**rmmod** führt einen sys_delete_module(9) Systemaufruf für das angegebene Modul aus. Auf diese Weise wird das Laufzeitmoduls gelöscht. Aus dem Arbeitsspeicher wird es erst entfernt, wenn die Anzahl der Prozesse, die das Modul benutzen 0 ist.
Als Kommandozeilenargument muß dem rmmod–Kommando der Modulname, nicht der Name der Objektdatei übergeben werden.

**Siehe auch:**

insmod auf S. 201

**Autor:** Anonym

## 4.19 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[8] 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, wenn keine Zeitspanne auf der Kommandozeile angegeben wird.

**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

Neben dem hier beschriebenen shutdown–Programm von Peter Orbaek gibt es noch ein shutdown–Kommando von Miquel van Smoorenburg, das als Front–End das separate Programm halt oder reboot vom gleichen Autor aufruft. Eine Besonderheit des shutdown–Front–Ends ist der Aufruf des Programms (Shellscripts) brc zum "Aufräumen" des Systems und zum Abbauen des Dateisystems.

**Optionen:**

**-h** hält das System an, ohne neu zu booten

**-r** startet das System nach dem Herunterfahren sofort neu

**-f** erzeugt die Datei `/etc/fastboot`, die in der Systeminitialisierungsdatei gesucht werden kann, um gegebenenfalls die Prüfung der Dateisysteme zu überspringen; diese Datei wird nicht automatisch gelöscht

**-q** unterdrückt die Frage nach einer Begründung für die Unterbrechung des Rechnerbetriebes (Meldung für wall)

**-s** erzeugt die Datei `/etc/singleboot`, die vom simpleinit ausgewertet wird und zum Systemstart im Einbenutzermodus führt

**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

## 4.20 umount

**Funktion:**

**umount** setzt ein aufgesetztes Dateisystem ab

**Syntax:**

**umount** [−a] [−t minix | ext| msdos]
**umount** *Gerätedatei* | *Verzeichnis*

[8] Seit der Linux–Version 0.99.10 kann auch das Root–Filesystem abgesetzt werden.

### Beschreibung:

**umount** setzt das aufgesetzte Dateisystem des mit der *Gerätedatei* verbundenen Gerätes von dem *Verzeichnis* ab.

Beispielsweise muß eine Diskette mit Dateisystem 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 so lange 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. Andernfalls wird eine Meldung der Form ...`device busy` ausgegeben.

### Optionen:

**-a** setzt alle in `/etc/fstab` aufgeführten Devices ab (auch das Root–Filesystem)

**-t** ***Typ*** setzt nur Dateisysteme vom *Typ* ab. Beschreibung des Parameters *Typ* ist beim Kommando mount zu finden.

# Kapitel 5

# Systemverwaltung

## 5.1 Das System starten

### 5.1.1 Von Diskette booten

Wenn im SETUP des Rechners nichts anderes eingestellt wurde, versucht das BIOS (*Basic Input/Output System*) 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 Boot–Loader gesteuert wird, der sich im ersten Sektor der Bootdiskette befindet. Das fertige Kernelimage beginnt immer mit einem solchen Bootprogramm, so daß ein solcher Kernel einfach nur "roh" auf eine formatierte Diskette kopiert werden muß, um eine Bootdiskette zu erhalten.[1]

Wenn also beim Einschalten des Rechners eine Linux–Bootdiskette im Laufwerk A: liegt, wird der darauf enthaltene Kernel in den Arbeitsspeicher geladen. Dabei 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.

### 5.1.2 Von Festplatte booten

Beim Booten von Festplatte ist der Vorgang etwas komplizierter. Die Festplatten für PC können in maximal vier primäre Partitionen unterteilt werden, auf denen jeweils ein anderes Betriebssystem installiert sein kann. Diese Partitionen sind in einer Partitionstabelle beschrieben, die im ersten Block der Festplatte enthalten ist. Um ein bestimmtes Betriebssystem von einer der Partitionen laden zu können, befindet sich in diesem Block auch der Primary–Boot–Loader, der Block als Ganzes wird als Master–Boot–Record (MBR) bezeichnet.

Weil die unterschiedlichen Ansprüche der verschiedenen für PC denkbaren Betriebssysteme nicht mit einem einzigen Primary–Boot–Loader erfüllt werden können, lädt dieser einen betriebssystemspezifischen Secondary–Boot–Loader vom ersten Block einer Partition.[2] Mit Ausnahme einer erweiterten Partition enthalten die primären Partionen selbst keine Partitionstabellen. Trotzdem wird bei allen Dateisystemen der erste Block für den Boot–Loader freigehalten.

Der erste Block reicht natürlich nicht für das komplette Betriebssystem, die darauffolgenden Blöcke der Linux–Dateisysteme sind für spezielle Verwaltungsdaten vorgesehen, können also den Kernel auch nicht aufnehmen. Allein das Xia–Dateisystem kann so eingerichtet werden, daß die ersten Blöcke der Partition für das Betriebssystem frei bleiben. Mit dem mkboot–Programm (→ Seite 202) von Frank Xia kann auf so einer primären Partition ein bootfähiger Linux–Kernel installiert werden.

---

[1] Es ist möglich, aus einem normalen Diskettendateisystem mit Kernelimage eine bootfähige Diskette zu machen, indem mit dem LILO Paket ein entsprechender Boot–Loader für dieses Kernelimage installiert wird. Einzelheiten dazu finden Sie in der ausführlichen LILO–Dokumentation von Werner Almesberger.

[2] Bei dem üblichen MS–DOS–MBR kann genau eine Partition aktiviert werden, von der dann das Betriebssystem geladen wird.

### LILO

Für beliebige Linux–Partitionen geeignet ist der *Generic boot loader for Linux*, kurz LILO von Werner Almesberger. Dessen Vielseitigkeit beruht vor allem darauf, daß er mit einer als *map* bezeichneten Tabelle das Kernel–Image und andere Daten von beliebigen Plattenblöcken, sogar von beliebigen Festplatten holen kann. Einzige Voraussetzung ist, daß der Bootsektor von LILO auf einer primären Partition der ersten Festplatte installiert werden kann.[3]

Hier soll nur die Verwendung von LILO zum Booten von einer primären Linux–Partition auf der ersten Festplatte `/dev/hda` beschrieben werden. Für Einzelheiten und Spezialfragen (beispielsweise wenn Sie von einer SCSI–Festplatte booten wollen) sollten Sie sich die ausgezeichnete Dokumentation durchlesen, die zusammen mit den LILO–Sourcen verteilt wird.

### Installation

In den meisten Linux–Distributionen ist eine installationsfähige Version des LILO–Pakets enthalten. Meistens werden allerdings nicht die Quelltexte verteilt, und — das ist ein besonderer Nachteil — die Dokumentation im LaTeX–Format fehlt.

Je nachdem, ob sich die Distributoren Ihres Linux–Paketes an die Empfehlungen des neuen File–System–Standards halten oder nicht, befinden sich die Dateien des LILO–Pakets zusammen im Verzeichnis `/etc/lilo` (alte Konfiguration), oder sie sind ihren Funktionen entsprechend auf die Verzeichnisse `/boot`, `/sbin` und `/etc` aufgeteilt.

Um Ihre Linux–Partition bootfähig zu machen, müssen Sie die LILO–Konfigurationsdatei `/etc/lilo.conf` editieren/erstellen und das lilo–Kommando zur Installation des Boot–Loaders und der Map aufrufen.[4]

Das folgende Beispiel zeigt eine solche Konfigurationsdatei für ein System, bei dem auf der ersten Partition einer AT–Bus–Festplatte (`/dev/hda1`) MS–DOS und auf der zweiten (primären) Partition (`dev/hda2`) Linux installiert ist.

```
boot=/dev/hda2
install=/boot/boot.b
map=/boot/map
vga=3
delay=5
image=/vmlinuz
        label=linux
        root=/dev/hda2
        read-only
other=/dev/hda1
        label=mess-dos
        table=/dev/hda
```

In der Regel sollte es ausreichen, die Bezeichnungen der Gerätedateien sowie den Namen des Kernel–Images dem konkreten System anzupassen. Sie müssen darauf achten, daß als Bootdevice nicht die rohe Festplatte angegeben wird (`boot=/dev/hda`).

Bevor Sie den Boot–Loader der Konfiguration entsprechend durch Aufruf des lilo–Kommandos installieren, sollten Sie auf jeden Fall eine Sicherungskopie Ihres alten Bootsektors machen, damit Sie ihn nötigenfalls restaurieren können. Das folgende Beispiel zeigt, wie Sie den Bootsektor von `/dev/hda2` in die Datei `/boot/bootsector` schreiben können:

[3] Dazu ist jede Partition mit einem Linux–Dateisystem geeignet (nicht aber die Swap–Partition), zur Not geht auch die erweiterte Partition. Wenn gar nichts hilft, kann LILO auch als Primary–Boot–Loader im Master–Boot–Record installiert werden.

[4] In älteren Versionen von LILO hieß die Konfigurationsdatei `/etc/lilo/config` und das Shellscript `/etc/lilo/install` hatte die Aufgabe, das eigentliche lilo–Programm mit den passendenden Optionen aufzurufen. Bei den neueren Distributionen existiert die `install`–Datei nur noch als Link auf `lilo`, das ohne Parameter direkt aufgerufen werden kann, um die in `/etc/lilo.conf` beschriebene Konfiguration zu aktivieren.

```
# dd if=/dev/hda2 of=/boot/bootsector bs=512 count=1
1+0 records in
1+0 records out
# _
```

Das Input–File muß mit dem bei der `boot`–Variablen in der Konfigurationsdatei angegebenen Device übereinstimmen. Sie sollten die Sicherungsdatei außerdem auf einer Diskette speichern.

### Konfigurationsdatei

Das oben gezeigte Beispiel kann meistens mit leichten Veränderungen übernommen werden. Die folgende Liste erklärt Ihnen die Funktion der einzelnen Einträge in der oben gezeigten Konfigurationsdatei. Der LILO–Boot–Loader erlaubt noch wesentlich mehr Einstellungen. Eine vollständige Liste finden Sie im Dokument von W. Almesberger.

Die Konfigurationsdatei enthält Schalter und Variable. Variablen wird durch eine Gleichung ein Wert zugewiesen.

**boot=*Partition*** Diese Variable bestimmt die Partition, auf die der Boot–Loader geschrieben werden soll. Wenn diese Angabe in der Konfigurationsdatei fehlt, wird der Bootsektor auf die aktuelle Rootpartition geschrieben. Um mit dem normalen Master–Boot–Record (MBR) vom MS–DOS davon booten zu können, muß es sich um eine primäre Partition der ersten Festplatte handeln. Für kompliziertere Systemkonstellationen gibt es die Möglichkeit, den Boot–Loader auf dem Master–Boot–Record oder auf der erweiterten (primären) Partition zu installieren. Bevor Sie sich für eine dieser Varianten entscheiden, sollten Sie die Dokumentation zum LILO–Paket sorgfältig durchlesen.

**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 zur Ausgabe des Boot–Prompts wartet. Wenn diese Zeit verstrichen ist, wird automatisch das erste in der Konfigurationsdatei definierte Kernelimage geladen.

**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 der Systemverwalterin 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 210 beschrieben. Wenn dieser Eintrag in der Konfigurationsdatei fehlt, wird der in der Kerneldatei gespeicherte Videomodus beibehalten.
- **ramdisk = *Kilobytes*** Es wird eine RAM–Disk in der angegebenen Größe eingerichtet, unabhängig davon, ob dies beim Übersetzen des Kernels so bestimmt wurde. Siehe auch beim rdev–Kommando auf Seite 210.

**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.

Sie können mit LILO auch ein anderes Betriebssystem, beispielsweise MS–DOS, booten. Dazu dienen die folgenden zusätzlichen Variablen:

**other =** ***Device*** Die Variable `other` leitet einen speziellen Teil der Konfigurationsdatei ein, wie `image`. Die hier angegebene Gerätedatei muß den Secondary–Boot–Loader für das andere Betriebssystem enthalten. Beispielsweise muß beim MS–DOS hier die primäre DOS–Partition angegeben werden.

**loader =** ***chain_loader*** Innerhalb des von `other` eingeleiteten speziellen Teils wird mit der Variablen `loader` ein spezielles Programm bestimmt, das bestimmte Vorbereitungen zum Umschalten der Betriebssysteme treffen kann. Das Standardprogramm (Voreinstellung) `chain.b` übergibt einfach die Kontrolle an den vom LILO–Secondary–Boot–Loader geladenen Secondary–Boot–Loader des fremden Betriebssystems.

**table =** ***Device*** In der Variablen `table` kann die Gerätedatei angegeben werden, aus der die Partitionstabelle für das fremde Betriebssystem gelesen werden kann. Im Fall von MS–DOS kann diese Angabe unterbleiben, weil eine Art interner Partitionstabelle im DOS–Dateisystem gespeichert ist.

### Der LILO–Prompt

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 und wartet die in der Konfigurationsdatei unter `timeout` bestimmte Zeitspanne auf eine Kommandozeile.

Auf dieser Kommandozeile können zuerst einmal Parameter an den Boot–Loader übergeben werden. Zusätzlich kann der Boot–Loader aber auch einen Teil der Kommandozeile an den Kernel weitergeben.

Als Kommandozeilenargument für den Boot–Loader selbst kommt vor allem der als `label` bezeichnete Name des zu ladenden Kernel–Image bzw. das `label` eines anderen Betriebssystems in Frage. Im Beispiel oben sind das `linux` oder `mess-dos`. Eine Liste aller möglichen Labels können Sie durch Eingabe eines TAB oder eines Unterstrichs (das ist auf der US–Tastatur das Fragezeichen) ausgeben lassen.

Alle weiteren Kommandozeilenparameter werden an den Linux–Kernel weitergegeben. Dazu muß natürlich als erstes Kommandozeilenargument das Label für ein gültiges Kernel–Image angegeben werden.

Eine erste Gruppe von Argumenten überlagert die im Kernel–Makefile bzw. mit dem `rdev`–Kommando eingestellten Parameter.

**root=*****Root-Partition*** veranlaßt den Kernel, die angegebene Root–Partition zu mounten.

**read-only (ro)** veranlaßt den Kernel, die Root–Partition Read–Only zu mounten.

**rw** veranlaßt den Kernel, die Root–Partition mit Schreibberechtigung zu mounten, auch wenn in der Kerneldatei oder in der LILO–Konfigurationsdatei etwas anderes festgelegt ist.

**vga=*****Modus*** stellt den Videomodus für den Textbildschirm ein (die Modi entsprechen den beim `rdev`–Kommando auf S. 210 beschriebenen).

Eine andere Gruppe von Argumenten dient der Konfiguration bestimmter Gerätetreiber. Wenn eines dieser Argumente auf der Kommandozeile auftaucht, wird die Setup–Funktion für den entsprechenden Gerätetreiber mit den hier angegebenen Parametern aufgerufen. Besonders für die generischen Kernel–Images auf den Bootdisketten der verschiedenen Distributionen ist dies oft die einzige Möglichkeit, eine vom Standard abweichende Konfiguration zum Laufen zu bringen. Die mit der Setup–Funktion dem Gerätetreiber übergebenen Parameter verdecken bei der anschließenden Initialisierung des Gerätes die Defaultwerte aus der Kerneldatei.

**reserve=*Adresse*,*Größe*[,*Adresse*,*Größe*...]** markiert bis zu fünf Adreßbereiche für IO–Ports als belegt und schützt sie so vor dem Zugriff beliebiger Gerätetreiber beim Auto–Probing. Diese Reservierung ist beispielsweise für den IO–Bereich der NE2000–Ethernet–Karten sinnvoll, die nach einem unkontrollierten Zugriff durch einen wilden Gerätetreiber den Rechner blockieren können.

**ether=*IRQ*,*IO-Port*,*Mem-Start*,*Mem-End*,*Device*** konfiguriert den allgemeinen Treiber für die Ethernet–Karten mit den angegebenen Parametern. Die Werte für den Beginn und das Ende des Shared–Memory–Bereiches können für WD–80x3–Karten wichtig sein, allerdings werden diese Werte in der Regel automatisch korrekt erkannt, wenn sie hier mit 0 angegeben werden.

**hd=*Cyl*,*Head*,*Sect*** konfiguriert den Festplattentreiber für die normale (AT–Bus–) Festplatte `/dev/hda` mit den angegebenen Parametern.

**bmouse=*IRQ*** konfiguriert den Logitech–Busmaustreiber auf den angegebenen Interrupt.

**st0x=*IO-Port*,*IRQ*** konfiguriert die Seagate st01/st02 SCSI Host–Adapter mit den angegebenen Parametern.

**tmc8xx=*IO-Port*,*IRQ*** konfiguriert die Future Domain SCSI Host–Adapter TMC–885 TMC–950.

**t128=*IO-Port*,*IRQ*** konfiguriert die SCSI Host–Adapter T128/T128F/T228 von Trantor.

**ncr5380=*IO-Port*,*IRQ*,*DMA*** konfiguriert den generischen SCSI–Treiber für Host–Adapter mit Chips der NCR 5380 Familie.

**aha152x=*IO-Port*,*IRQ*,*SCSI-ID*,*Reconnect*** konfiguriert den Gerätetreiber für den Adaptec 1520/1522 SCSI Host–Adapter.

**xd=*Typ*,*IRQ*,*IO-Port*,*DMA*** konfiguriert den Gerätetreiber für den 8–Bit– XT–Festplattencontroller

**mcd=*IO-Port*,*IRQ*** konfiguriert den Treiber für ein Mitsumi CD–ROM–Laufwerk.

**sound=*Zahl*[,*Zahl*...]** konfiguriert den oder die Soundkarten. Die Zahl enthält verschlüsselt die Konfigurationsdaten: die niedrigsten vier Bits den DMA–Channel, das 5. bis 8. Bit den IRQ, das 9.–20. Bit den IO–Port und das 21. bis 28. Bit den Kartentyp.

Die folgenden beiden Kommandozeilenargumente schalten bestimmte Kernelfunktionen an bzw. ab:

**debug** schaltet die Ausgaberoutine für Kernelmeldungen in den Debug–Modus.

**no387** schaltet den mathematischen Koprozessor ab. Damit der Kernel trotzdem startet, müssen die Routinen der Mathe–Emulation in den Kernel eingebunden sein.

Außer den hier aufgeführten Argumenten können noch weitere Optionen (Schalter) angegeben werden, die der Kernel an das init–Programm weiterreicht, und es können Gleichungen zur Definition von Umgebungsvariablen angegeben werden, die automatisch in der Prozeßumgebung aller laufenden Programme auftauchen.

### LILO deinstallieren

Bei einem Programm wie dem Linux–Boot–Loader, das sich buchstäblich am Rand des Geltungsbereichs der Betriebssysteme aufhält, taucht gelegentlich die Frage nach der Entfernung (Deinstallation) auf.
Wenn Sie den Empfehlungen zur Installation von LILO gefolgt sind und eine Sicherheitskopie des Bootsektors angelegt haben, bevor Sie ihn mit dem Boot–Loader überschrieben haben, können Sie einfach diese Sicherheitskopie wieder zurückschreiben. Wenn LILO beispielsweise auf der Partition `/dev/hda2` installiert ist und die gesicherten Daten in `/boot/bootsector` liegen, schreibt das folgende Beispiel den Bootsektor zurück:

```
# dd if=/boot/bootsector of=/dev/hda2 bs=512 count=1
1+0 records in
1+0 records out
# _
```

Im Fall der primären Partition /dev/hda2 ist die Rekonstruktion nicht unbedingt notwendig, beispielsweise führt das Löschen der Partition auch zum erwünschten Ergebnis. Wenn der Boot–Loader aus Versehen oder absichtlich im Master–Boot–Record installiert wurde, kann es zu Problemen kommen. Beim Löschen der Partitionen wird im MBR nur die Partitionstabelle, nicht aber der Primary–Boot–Loader gelöscht. Wenn Sie keine Sicherheitskopie des MBR angelegt haben, können Sie den Primary–Boot–Loader vom MS–DOS 5.0 mit dem folgenden (DOS–) Kommando restaurieren:

```
A:\ >fdisk /mbr
A:\ >
```

Das mkboot–Kommando (→ S. 202) bietet eine ähnliche Option.

### 5.1.3 Die Vorgänge bei der Systeminitialisierung

Seit der Version 0.99.12 sind alle Kernel komprimiert gespeichert[5] und entpacken sich nach dem Laden selbsttätig im Arbeitsspeicher. Wenn der Kernel ausgepackt und an die richtige Stelle (virtueller Adreßraum ab 3 GB) gerückt ist, springt der Programmzähler auf die Startadresse der Kernels, der damit die Kontrolle über den Rechner übernimmt.

Die Systeminitialisierung kann grob in zwei Phasen unterteilt werden. Die erste Phase wird von der Kerneldatei allein und weitgehend ohne Einfluß der Systemverwalterin durchgeführt und dient der Erkennung und Initialisierung der Komponenten sowie der Geräte bzw. der Gerätetreiber. Zum Abschluß der ersten Phase wird die "Benutzerebene" des Betriebssystems eröffnet und in der damit beginnenden zweiten Phase ganz bestimmte Programme automatisch ausgeführt.

Auch wenn Sie die meisten Vorgänge der ersten Phase nicht direkt beeinflussen können, ist es gut, wenn Sie einmal die verschiedenen Schritte nachverfolgen und vor allem die Bildschirmausgabe bei der Systeminitialisierung verstehen. Besonders, wenn Ihr Linux–Kernel nicht mit der speziellen Hardwarekonstellation Ihres Rechners zusammenarbeiten will, finden Sie in den Zeilen des Boot–Bildschirms meistens die entscheidenden Hinweise zur Eingrenzung des Problems, mit denen Sie es schließlich beheben können.

Die Initialisierung des Kernels findet in der Funktion start_kernel(9) (die Sie in den Kernelsourcen in der Datei `./linux/init/main.c` finden können) statt und läuft in den folgenden Schritten ab:

Zuerst 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 Prozeßtabelle und der Start der Schedulers. Dabei findet noch keine Bildschirmausgabe statt.

Dann wird die Kommandozeile des Kernels ausgewertet. Dabei können verschiedene Parameter für die Initialisierung der Gerätetreiber verändert werden. Die von der Linux–Version 1.0 unterstützten Kommandozeilenargumente sind auf Seite 218 erklärt.

Nach der Auswertung der Kommandozeile beginnt die Initialisierung der Gerätetreiber. Zuerst werden die zeichenorientierten Geräte initialisiert.

Die Systemconsole wird als erstes eingerichtet; danach können die Kernelmeldungen auf dem Bildschirm erscheinen. Wenn für den Textbildschirm kein fester Video–Modus eingestellt wurde, wird an dieser Stelle eine Liste aller von der Grafikkarte unterstützten Modi ausgegeben und auf eine Auswahl gewartet.

Als nächstes werden die "normalen" seriellen Schnittstellen initialisiert. Schnittstellenkarten mit zwei Standard–Ports werden von jedem Linux–Kernel ohne Probleme erkannt. Die 16550A–Schnittstellenbausteine werden unterstützt. Zusätzliche serielle Schnittstellen oder vom Standard abweichende Konfigurationen werden nicht initialisiert. Die Treiber sind aber ohne weiteres in der Lage, auch mit diesen Geräten umzugehen, wenn sie mit dem setserial–Programm zur Laufzeit des Kernels konfiguriert und initialisiert worden sind.

Im nächsten Schritt sucht und initialisiert der Kernel die parallele(n) Druckerschnittstelle(n), danach werden eventuell eingebundene Busmaus–Treiber sowie die Audio–Devices initialisiert. Zum Abschluß der Initialisierung zeichenorientierter Geräte wird gegebenenfalls der Treiber für den QIC–02 Streamer vorbereitet.

Bei der Treiberinitialisierung für blockorientierte Geräte wird für die Gerätetreiber der "normalen" Festplatten Kernelspeicher reserviert, es findet aber noch kein Festplattenzugriff statt. In den nächsten Schritten werden gegebenenfalls die Treiber für den XT–Controller und die Mitsumi– und Sony–CD–ROM–Laufwerke initialisiert.
Bei der Übersetzung des Kernels mit dem rdev–Programm oder mit einem LILO–Konfigurationsparameter kann der Kernel veranlaßt werden, einen Speicherbereich bestimmter Größe als RAM–Disk anzulegen. Die Initialisierung des entsprechenden Gerätetreibers findet an dieser Stelle statt.

Danach sucht der Kernel nach SCSI Hostadaptern, und wenn er welche gefunden hat, wird nach SCSI Geräten gesucht.[6] Auch hier werden einigermaßen unkomplizierte Konfigurationen automatisch erkannt.

[5] Frühere Versionen ab 0.99.6 konnten optional komprimiert sein.
[6] Hier werden auch die SCSI–Streamer initialisiert, obwohl sie zeichenorientiert arbeiten.

Wenn es Probleme mit einem Hostadapter oder einer Festplatte gibt, die prinzipiell unterstützt wird, aber in der Konfiguration nicht korrekt erkannt wird, können einige Treiber auf der Kommandozeile des Kernels konfiguriert werden, um das Problem zu lösen (→ S. 218).

In den weiteren Schritten werden der Arbeitsspeicher und eine Reihe von internen Tabellen und Parametern initialisiert, ohne daß es auf dem Bildschirm dokumentiert wird.

Dann werden die Diskettenlaufwerke gesucht und initialisiert.

Im nächsten Schritt werden die Sockets für den Kernel und die TCP/IP–Treiber eingerichtet. Eine vorhandene Ethernetkarte wird an dieser Stelle initialisiert ist aber für die Funktion der TCP/IP–Protokolle nicht unbedingt notwendig. In jedem Fall wird ein Loopback–Device eingerichtet, mit dem die verschiedenen Netzwerkprogramme auf dem lokalen Rechner arbeiten können.

Schließlich werden noch gegebenenfalls die Kernelroutinen für die Inter–Prozeß–Kommunikation (IPC) und den mathematischen Coprozessor eingeschaltet.

Mit diesen Schritten ist die erste Phase der Systeminitialisierung beendet. Auf dem Startbildschirm wird an dieser Stelle der "Linux–Banner“ ausgegeben, in dem die Versionsnummer und das Übersetzungsdatum des nun laufenden Kernels enthalten sind.

An dieser Stelle ist der Kernel im Prinzip arbeitsfähig. In der nun folgenden zweiten Phase wird das Benutzersystem initialisiert. Die Unterscheidung zwischen Kernel– und Benutzerbereich ist eines der mächtigsten Konzepte, das Linux von Unix übernommen hat. Während der Kernel die vollständige Kontrolle über den Rechner hat und alle technisch möglichen Operationen ausführen kann/darf, befinden sich Benutzerprozesse immer unter der Aufsicht des Kernels.

Um das Benutzersystem zum Leben zu erwecken, wechselt der Kernel das erste Mal in den Benutzermodus und startet die ersten beiden Prozesse (Tasks).

Die Numerierung der Prozesse beginnt mit 0. Der "nullte“ Task des Kernels ist der idle–Prozeß, der immer lauffähig ist und vom Scheduler aufgerufen wird, wenn kein anderer Prozeß Rechenzeit braucht.

### 5.1.4 init

Der Prozeß mit der Nummer eins ist die eigentliche Wurzel des Benutzersystems. Dieser Prozeß verzweigt sich mit dem fork(2)–Systemaufruf und erzeugt auf diese Weise weitere Prozesse, deren Aktivitäten das Erscheinungsbild des Linux–Systems ausmachen.

Um die Gestaltung des Benutzersystems maximal flexibel zu halten, versucht der Kernel, das eigentliche init–Kommando als externes Programm vom Root–Filesystem zu laden.[7] Um überhaupt auf das Root–Filesystem zugreifen zu können, wird als erster Schritt der setup(2)–Systemcall durchgeführt, der wieder in den Kernelmodus zurückschaltet und dort die Festplatten initialisiert. Wenn der Kernel eine RAM–Disk angelegt hat, versucht er, sie von der Bootdiskette zu laden. Bevor der Systemcall beendet und wieder zum init–Prozeß in den Benutzermodus gewechselt wird, wird das Root–Filesystem gemountet.

Die wichtigste Aufgabe des init–Prozesses ist die Eröffnung der interaktiven Oberfläche, ohne die keine Benutzeraktivität am Rechner möglich wäre. Zu diesem Zweck werden von init auf allen Ports, auf denen das Einloggen möglich sein soll, getty–Prozesse erzeugt. Um nach der Beendigung einer Session auf einem bestimmten Port immer wieder ein Login zu ermöglichen, wird von dem besonderen Eltern–Kind–Verhältnis zwischen erzeugendem und erzeugtem Prozeß Gebrauch gemacht: das init–Programm bekommt automatisch ein Signal vom Kernel, wenn der Kindprozeß zu Ende ist, und kann ihn sofort neu starten.

Der Name 'init' deutet auf eine weitere Aufgabe des Programms hin: Es initialisiert das System. Dazu werden bestimmte Shellprogramme interpretiert. Weil diese Scriptdateien auf einfache Weise von der Systemverwalterin erstellt und verändert werden können, kann diese Phase der Systeminitialisierung im Gegensatz zur Kernelinitialisierung vollständig von der Systemverwalterin konfiguriert werden.

Für Linux sind zwei sehr verschiedene Versionen von init verbreitet:

[7] Das init Programm wird in den Verzeichnissen /etc, /bin und /sbin gesucht. Wenn es in keinem dieser Verzeichnisse 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. Diese Art, Linux zu starten, ist ein Relikt aus der Zeit, als es weder init noch login für Linux gab.

Das einfache init (simpleinit) von Peter Orbaek mit Erweiterung von Werner Almesberger beschränkt sich auf die wesentlichen Aufgaben: es wird ein einziges Shellscript, die Datei `/etc/rc`, der Standardshell zur Interpretation übergeben, und es werden die in der Datei `/etc/inittab` bestimmten virtuellen Terminals und seriellen Schnittstellen mit einem getty–Prozeß belegt. Für spezielle Situationen kann das simpleinit in einem speziellen Einbenutzermodus (single user mode) gestartet werden. In diesem Modus wird das Initialisierungsscript nicht interpretiert, und es wird nur eine einzige Shell mit Root–Rechten gestartet.

Das vielseitigere, und damit natürlich auch kompliziertere, System V kompatible init Programm von Mike Jadgis und 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 oder wenn auf der Kommandozeile des Kernels das Schlüsselwort `single` angegeben wurde. Wenn außerdem die Datei `/etc/securesingle` existiert, muß das korrekte Rootpaßwort eingegeben werden, bevor init die Shell öffnet.

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. Außer den getty–Prozessen für die Terminals werden vom simpleinit nach der Ausführung des Shellscripts keine weiteren Programme gestartet. Das bedeutet, daß alle darüber hinaus notwendigen Systeminitialisierungen in diesem Shellscript durchgeführt werden müssen.

```
# /etc/rc

# Das Root-Filesystem ist Read-Only gemountet.
# Zuerst werden die Dateisysteme getestet...
echo "Dateisysteme werden geprueft..."
/sbin/fsck -A -a
f [ $? -gt 1 ] ; then
        echo "Fehler im Dateisystem gefunden. Bitte neu booten!"
        /bin/sh
fi

# ...dann wird das Root-Filesystem mit Schreiberlaubnis remountet.
/sbin/mount -n -o remount /dev/hda1 /

# Hier werden zur Sicherheit ein paar Dateien entfernt.
/bin/rm -f /etc/mtab* /etc/nologin
/bin/cat /dev/null > /etc/utmp

# Hier wird der Rest des Dateisystems ohne das NFS gemountet
# und die Swappartition aktiviert.
/sbin/mount -avt nonfs
/sbin/swapon -a

# Mit den Scriptdateien rc.inet? wird das TCP/IP Networking
# gestartet.
/bin/sh /etc/rc.inet1
/bin/sh /etc/rc.inet2

# Die Systemuhr wird aus dem CMOS gelesen.
/usr/sbin/clock -u -s
```

```
# Einige Daemonen werden gestartet.
echo -n "Starting daemons: "
/usr/sbin/crond
/usr/sbin/lpd
# Der update Prozess muss beim simpleinit unbedingt im
# rc-Script gestartet werden.
/sbin/update &

# Die Tastaturtabelle wird geladen.
/sbin/loadkeys /etc/keytables/gr-latin1.map

# Schliesslich wird noch das lokale Initialisierungscript
# aufgerufen.
/bin/sh /etc/rc.local
```

Nachdem das Shellscript `/etc/rc` abgearbeitet ist, liest das simpleinit die Datei `/etc/inittab`, in der ausschließlich die getty–Prozesse gestartet werden, mit denen das Einloggen auf den verschiedenen Terminals erst möglich wird.

```
# inittab fuer simpleinit
# Format: tty-Port:termcap-Eintrag:getty-Kommando
#
# Die ersten fuenf Zeilen sind fuer die virtuellen Terminals...
#
tty1:con100x40:/sbin/getty 9600 tty1
tty2:con100x40:/sbin/getty 9600 tty2
tty3:con100x40:/sbin/getty 9600 tty3
tty4:con100x40:/sbin/getty 9600 tty4
#tty5:con100x40:/sbin/getty 9600 tty5
#
# ...die letzte Zeile ist fuer den Modem-Port.
#
ttyS1:vt102:/sbin/getty 9600 ttyS1
```

Die Syntax für das getty–Kommando hängt von der Version ab. Jeder dieser getty Prozesse wird in einer Art Endlosschleife immer wieder erzeugt, wenn die Loginshell auf dem zugehörigen Terminal verlassen wird. Das bedeutet auch, daß der init Prozeß niemals wirklich beendet wird.
Es besteht aber die Möglichkeit, mit einem '`kill -1 1`' das init–Programm 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 simpleinit nicht automatisch beendet. Diese Prozesse müssen einzeln direkt von der Systemverwalterin beendet werden.

### Sysvinit

Wie bereits weiter oben gesagt, bietet das System–V–kompatible init durch die Runlevel eine sehr flexible Methode zur Systemkonfiguration. 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 `sysvinittab`) festgelegt. Diese Datei enthält Datensätze, deren Bedeutung bei der Reise durch das Dateisystem auf der Seite 37 beschrieben wird.
Zur Änderung des Runlevels gibt es das telinit[8]–Kommando, das als einzigen Kommandozeilenparameter den Namen oder das Zeichen (Buchstabe oder Ziffer) des gewünschten Runlevels erwartet. Dieses Kommando kann nur von der Superuserin ausgeführt werden.

[8] telinit ist ein einfacher Link auf init

## 5.2 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 der Systemverwalterin das Herunterfahren des Systems zu erleichtern, gibt es das shutdown–Programm mit den Varianten reboot, halt, fastboot und fasthalt, die in der Programmversion von Peter Orbaek jeweils Links auf shutdown sind. Eine Beschreibung ist auf Seite 211 zu finden.

Wenn aus irgendwelchen Gründen das System nicht mit shutdown angehalten werden kann, können Sie versuchen, das System "per Hand" herunterzufahren. Voraussetzung dafür ist natürlich, daß Sie Root–Privilegien haben.

Für ein geordnetes Herunterfahren des Rechners müssen alle Benutzerprozesse angehalten werden. Nachdem Sie alle aktiven natürlichen Systembenutzer mit einer wall–Message gewarnt haben, können Sie (nach einer angemessenen Zeit) die Anwenderprogramme und Loginshells mit TERM–, später KILL–Signalen beenden. Um das Einloggen weiterer Systembenutzer zu verhindern, sollten Sie vorher die Datei `/etc/nologin` anlegen. Die Systemdämonen müssen ebenfalls beendet werden.

Mit dem sysvinit haben Sie dann die Möglichkeit, das System in den Einbenutzermodus zu bringen, indem Sie das Kommando '`telinit 1`' aufrufen.

Wenn Sie das System wegen irgendwelcher Störungen herunterfahren, sollten Sie auf jeden Fall häufiger eine Synchronisation des Dateisystems mit dem sync–Kommando versuchen. Wenn Sie den Rechner ausschalten müssen, sollte sync die letzte Aktion sein, bevor Sie den Netzschalter betätigen! Um das sync–Kommando auszuführen, brauchen Sie keine Root–Privilegien.

#### Dateisystem abbauen

Mit dem Kommando '`umount -a`' 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.

Um ein Dateisystem abzubauen, darf es nicht "aktiv" sein. Das bedeutet, daß kein Dateisystem auf einem Verzeichnis dieses Systems aufgesetzt sein darf und 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.

Seit der Kernel–Version 0.99.10 wird auch das Root–Filesystem von dem umount–Kommando abgebaut. Das hat in Verbindung mit dem ext2–Dateisystem den Vorteil, daß das Valid–Flag gesetzt wird und so beim nächsten Systemstart der File–System–Check übersprungen werden kann.

## 5.3 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 Systemdaten zu einem Programm gezählt. Dazu gehört die Prozeßumgebung, die Prozeßgruppe, der Eigentümer, das kontrollierende Terminal, die von dem Programm geöffneten Dateien (auch Devices), das Arbeitsverzeichnis und eine Menge weiterer Daten.

Ein neuer Prozeß entsteht, indem ein laufender Prozeß sich mit dem fork Systemaufruf teilt. Dabei wird ein neuer Eintrag in der Prozeßtabelle angelegt, und ein paar weitere Datenstrukturen werden initialisiert. Damit ist ein neuer lauffähiger Prozeß entstanden, der als Kindprozeß (child process) bezeichnet wird.

Der Scheduler wählt aus den lauffähigen Prozessen jeweils den Prozeß mit der höchsten Priorität aus und teilt ihm Rechenzeit zu. Wenn ein neuer Eintrag das erste Mal vom Scheduler aufgerufen wird, also Rechenzeit zugeteilt bekommt, befindet sich der Kindprozeß (fast) in demselben Speicherbereich, den auch der Elternprozeß belegt. Erst wenn der Kindprozeß in eine Speicherseite schreibt, wird die Speicherseite kopiert und belegt so eigenen Raum im Arbeitsspeicher (copy on write). Durch einen exec– Systemaufruf kann der Kindprozeß ein neues Programm in den Speicher laden. Auch dabei wird vom Kernel nicht sofort der gesamte Programmtext in den Speicher geholt (der dann immer neue Seiten des Speicherbereichs des Elternprozesses überschreiben würde und deshalb in einen neuen Speicherbereich kopiert werden müßte), sondern es werden nur die Speicherseiten geladen, auf die das neue Programm gerade zugreift (demand loading).

Der erste Prozeß, und damit Vater aller weiteren Prozesse, ist das bereits beschriebene init–Programm. Die daraus entstandenen getty–Programme erzeugen die login–Prozesse, die ihrerseits eine Benutzershell starten und damit eine Session eröffnen. Wenn ein Anwender in dieser Shell ein Kommando aufruft, wird die Shell zur Mutter und das aufgerufene Kommando zum Kind. Häufig wartet ein Elternprozeß 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 anderem die Prozeßnummer (PID) und die Nummer des Elternprozesses (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.

### 5.3.1 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 nicht einmal 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 die Superuserin 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 entziehen. 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 an.

## 5.4 Systemabsturz

Was für die Anwenderprogramme gilt, ist auch für den Kernel selbst nicht verkehrt. Auch wenn der Linux–Kernel eine beachtliche Stabilität erreicht hat und auf vielen Systemen unter hoher Last wochenlang ohne Unterbrechung läuft, kann das Betriebssystem nicht auf alle möglichen Ausnahmen vorbereitet sein, und es enthält auch Fehler. Ein Fehler des Betriebssystem ist nicht so leicht abzufangen wie der eines Anwenderprogramms. In einem solchen Fall kommt es häufiger 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 Sie versuchen, den Rechner mit einem halt–Kommando anzuhalten und so den Schaden zu begrenzen. In jedem Fall sollten Sie alle normale Benutzeraktivität am Rechner beenden und häufig das sync–Kommando ausführen.

Wenn der unerfreuliche Fall eines echten Systemabsturzes eingetreten ist, erscheint meistens eine Meldung der folgenden Form auf dem Bildschirm:

```
unable to handle kernel paging request at address C0000010
Oops: 0002
EIP:   0010:00118348
EFLAGS: 00000246
eax: fffffffd   ebx: 00000000   ecx: 00000216   edx: 000003d5
esi: 00105816   edi: 0016709f   ebp: 001756f8
ds: 002b  es: 002b  fs: 002b  gs: 002b
Pid: 00, process nr: 00
89 50 04 c7 03 00 00 00 00 c7
```

Aus der Fehlermeldung in der ersten Zeile kann geschlossen werden, daß eine Kernelfunktion versucht hat, auf die Speicheradresse 0x010 zuzugreifen. Weil der Kernel in den logischen Speicherbereich über ein Gigabyte (0xC0000000) verschoben ist, erscheint die Adresse mit diesem Offset.

In der zweiten Zeile erscheint eine weitere Fehlermeldung und ein Fehlercode. Die Oops Meldung aus diesem Beispiel ist typisch für Fehler des Memory–Management. Die Fehlernummer kann weiteren Aufschluß über die Ursache des Absturzes geben; um sie zu entschlüsseln, müssen Sie die Kernelsourcen heranziehen.

Der Extended Instruction Pointer EIP zum Zeitpunkt des Fehlers läßt Rückschlüsse auf die Kernelfunktion zu, die für den Absturz verantwortlich ist. Sie können den Namen der Funktion herausfinden, indem Sie die Symboltabelle des Kernels nach der diese Adresse umfassenden Funktion durchsuchen. Das folgende Kommando erledigt diese Aufgabe:

```
# nm /usr/src/linux/tools/zSystem | sort | grep 00118...
00118294 t _try_to_free_page
00118324 T _free_page
0011847c T ___get_free_page
00118638 t _try_to_unuse
0011878c T _sys_swapoff
00118934 T _sys_swapon
00118c64 T _si_swapinfo
00118cf4 T _do_mmap
00118cf4 t ___gnu_compiled_c
00118cf4 t gcc2_compiled.
```

```
00118cf4 t mmap.o
00118ec4 T _sys_mmap
00118f50 T _unmap_fixup
# _
```

Aus dieser Liste läßt sich die verantwortliche Funktion, in diesem Beispiel `free_page`,[9] leicht herausfinden. Die EFLAGS und die in den darauffolgenden Zeilen aufgelisteten Inhalte der Prozessorregister können im Einzelfall zur genauen Bestimmung der Fehlerursache ebenso herangezogen werden wie die Maschinencode–Sequenz in der letzten Zeile.

# 5.5 Die Konsistenz des Dateisystems prüfen

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 verlorengegangen sind. Aus diesem Grund muß unbedingt nach einem solchen Vorfall das Dateisystem mit einem File–System–Check–Programm überprüft und, wenn nötig, repariert werden.

Mit den in den aktuellen Distributionen häufig verwendeten Bootutilities läßt sich diese Prozedur leicht automatisch während der Systeminitialisierung beim Booten[10] durchführen. Insbesondere das ext2–Dateisystem ist ganz hervorragend für diese Methode geeignet, weil es mit dem sogenannten Valid–Flag die Möglichkeit bietet, den File–System–Check nur dann durchzuführen, wenn es zu einem unregelmäßigen Abbau des Dateisystems, zum Beispiel durch einen Systemabsturz, gekommen ist.

Jedes der unter Linux verwendeten Dateisysteme benötigt sein spezielles Programm für den File–System–Check. Zu diesem Zweck existieren die Hilfsprogramme fsck.minix, fsck.ext2 und fsck.xiafs. Das neue fsck–Front–End (→ S. 197) ermöglicht es, mit einem einzigen Aufruf alle Dateisysteme gemeinsam zu prüfen, Sie können aber selbstverständlich auch jedes der fsck.*–Programm direkt und einzeln aufrufen.

Sie können den File–System–Check für beliebige Partitionen durchführen. Sollte ein Dateisystem allerdings Fehler enthalten, darf die Reparatur nur auf einem inaktiven (also nicht aufgesetzten) Dateisystem durchgeführt werden. Auch aus diesem Grund bietet sich der automatische Test vor dem Zusammenbau des Dateisystems beim Booten an.

## 5.5.1 Das Rootfilesystem reparieren

Seit der Kernel–Version 0.99.10 erlaubt der Linux–Kernel das Absetzen der Root–Partition. Allerdings kann damit noch nicht das Root–Filesystem auf die gleiche Weise repariert werden, wie die anderen Dateisysteme, denn es ist nicht möglich, ein File–System–Check–Programm von einem abgesetzten Filesystem zu starten.

Um das Rootfilesystem zu testen, braucht es nicht abgesetzt zu werden, eine einfache Überprüfung mit der '–v' Option kann auch für das aufgesetzte Rootfilesystem durchgeführt 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 RAM–Disk als Rootfilesystem anlegt. Von dieser RAM–Disk aus kann dann das Rootfilesystem auf der Festplatte praktisch von außen repariert werden.

Eine zweite Möglichkeit besteht darin, das Root–Filesystem Read–Only zu mounten und auf diese Weise die Veränderung der Datenblöcke durch das Dateisystem zu verhindern. Der File–System–Check wird mit der Gerätedatei und damit auf der rohen Partition direkt durchgeführt, so daß der Schreibschutz für den

[9] Um Mißverständnisse zu vermeiden: Es handelt sich um ein konstruiertes Beispiel, die `free_page` Funktion verursacht so einen Fehler nicht. Es ist dem Autor nicht gelungen, eine echte Kernelpanic für ein realistischeres Beispiel herbeizuführen.

[10] Ein Beispiel für diese Methode wird in der Systeminitialisierungsdatei auf Seite 222 gezeigt

File–System–Check nicht wirkt. Mit dem mount–Kommando von den Bootutilities kann das Read–Only gemountete Root–Filesystem "remountet" werden, so daß nachträglich das Schreiben auf der Partition erlaubt werden kann, bevor die anderen Dateisysteme darauf gesetzt werden.

## 5.6 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 der Systemverwalterin 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 die Paßwortdatei,
2. wenn nötig, der Eintrag in die Gruppendatei,
3. das Erstellen des Heimatverzeichnisses mit den Initialisierungsdateien.

### 5.6.1 Eintrag in `/etc/passwd`

Die zentrale Benutzerdatenbank ist die Datei `/etc/passwd`. In dieser Datei muß jeder Anwender aufgeführt sein, damit er sich einloggen kann. Diese Datei enthält normal lesbaren Text und kann deshalb mit jedem Editor bearbeitet werden.

Für jeden Anwender muß die Systemverwalterin hier eine Zeile mit sieben Feldern anlegen. Die Felder werden durch einen Doppelpunkt ':' voneinander getrennt. Die Felder haben folgende Bedeutung:
*Benutzername*:*Paßwort*:*Benutzernummer*:*Gruppennummer*:*GCOS*:*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.[11]

  Jedem Benutzer steht das passwd–Kommando zur Verfügung, um sein Paßwort selbst zu verändern.

  Beim Neueintrag eines Benutzers wird dieses Feld freigelassen, und unmittelbar nach der vollständigen Eintragung muß der Benutzer oder die Systemverwalterin unter dem neuen Benutzernamen das passwd Kommando aufrufen und ein Paßwort eingeben.
- Das dritte Feld enthält die Benutzernummer (UID). Die Nummer ist eine beliebige nicht negative Zahl (bis 64000). Um genügend Benutzernummern für Verwaltungsaccounts offen zu halten, sollten die eigentlichen Anwender Benutzernummern größer als 99 erhalten.

  Die Benutzer werden kernelintern nicht durch die Namen, sondern durch die Benutzernummern unterschieden. Wenn eine Nummer zweimal mit unterschiedlichen Benutzernamen vergeben wird, behandelt der Kernel intern die beiden Accounts völlig identisch. Bei den Kommandos, die einen Benutzernamen anzeigen (z.B. `ls -l` oder `id`), wird immer der Eintrag verwendet, der als erstes in `/etc/passwd` steht.

[11] Wenn das Shadow–Paßwortsystem verwendet wird, gilt nur der erste Satz. Hier ist das eigentliche Paßwort–Feld in die Datei `/etc/shadow` ausgelagert und alle Accounts die mit einem Shadow–Paßwort gesichert sind werden in der `/etc/passwd` durch ein Sperrpaßwort gesichert.

- 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.

- Der fünfte Eintrag ist das sogenannte GCOS–Feld. Es enthält in der Regel den Realnamen des Benutzers, es können aber noch zusätzliche Daten zur Person oder zur Systemverwaltung in diesem Feld enthalten sein.[12]

  Der Realname wird von vielen News&Mail–Programmen bei der Zusammenstellung der Absenderadresse benutzt.

- 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.

### 5.6.2 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 (255)[13]. Jede Gruppennummer darf nur einmal 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.

[12] Das chfn–Kommando aus der Shadow–Paßwort–Suite benutzt beispielsweise das GCOS–Feld, um weitere persönliche Daten wie Zimmer– und Telefonnummer zu speichern, wie sie vom finger–Kommando angezeigt werden.

[13] Das Minix–Dateisystem kann nur Gruppenkennzahlen bis 255 (ein Byte) verwalten. Die Inodes der anderen Linux–Dateisysteme speichern die GID in zwei Bytes, können also Gruppenkennzahlen bis 64000 verkraften.

### 5.6.3 Das Heimatverzeichnis anlegen

Jeder Anwender braucht sein eigenes Heimatverzeichnis. Dieses Verzeichnis muß von der Superuserin 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–Window–Manager angelegt werden. Eine Beschreibung dieser Ersteinrichtung finden Sie im entsprechenden Abschnitt der Reise durch das Dateisystem auf Seite 44. Ein Satz solcher Initialisierungsdateien wird meistens in dem Verzeichnis `/usr/etc/skel` oder in `/etc/skel` aufbewahrt.

## 5.7 Die "Sicherheit" des Systems

### 5.7.1 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 unberechtigtem 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 (bzw. Zugriffsbeschränkung) an. Es wird unterschieden zwischen:

- dem Eigentümer einer Datei,
- einer Benutzergruppe, der die Datei zugeordnet wird, und
- allen anderen Benutzern.

Für jede dieser Kategorien kann das Lesen, Schreiben und Ausführen erlaubt und verboten werden.

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.[14]

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 von der Systemverwalterin in der Paßwortdatei zusammen mit dem Benutzernamen abgespeichert. Dieser Gruppe wird die Datei bei ihrer Erzeugung ebenfalls zugeordnet.

Die Ausführbarkeit macht nur bei binären Programmdateien oder bei Shellscripten Sinn.[15] Beim Versuch, irgendeine andere, nicht ausführbare Datei zur Ausführung zu bringen, wird das Kommando mit einer Fehlermeldung abgebrochen.

[14] 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.

[15] 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.

### Set User ID on Execution

Eine von UNIX geerbte Eigenschaft von Linux eröffnet die Möglichkeit, ein Programm mit den Rechten und Privilegien des Eigentümers oder einer Gruppe auszuführen. Zusätzlich zu den einfachen Zugriffsrechten gibt es im Permissions–Feld zu jeder Datei drei Spezialbits, von denen zwei das Betriebssystem veranlassen bei ausführbaren Dateien zur Laufzeit die Benutzer– bzw. Gruppenrechte zu ändern.
Die Bits heißen ihren Funktionen entsprechend *Set User ID* (SUID) bzw. *Set Group ID* (SGID) und ändern die *Effektive User–ID* (EUID) oder die *Effektive Group–ID* (EGID) eines Benutzers für die Laufzeit des Programms auf die in der Inode der ausführbaren Datei für Eigentümer oder Gruppe gespeicherten Werte.
Beispielsweise setzt das `passwd`–Kommando zur Laufzeit die EUID auf 0, also die UID der privilegierten Systemverwalterin. Auf diese Weise kann jeder Benutzer sein eigenes Paßwort verändern, ohne selbst Schreibberechtigung für die `/etc/passwd` Datei zu haben. Das `passwd`–Programm gilt natürlich als sicheres Kommando und sorgt dafür, daß kein Anwender ein anderes als sein eigenes Paßwort ändern kann.
Die Ausführung eines Programms mit Superuserprivilegien ist ein potentielles Sicherheitsloch, weil es dem Benutzer jeden Systemzugriff erlaubt. Für einige Programme, wie das erwähnte `passwd`–Kommando oder den X–Server, hat das seine Berechtigung.
Für andere Aufgaben ist es besser, speziellere Methoden zu wählen. Wenn beispielsweise ein Druckerdämon zur Verwaltung des Druckers installiert ist, sollten die Anwender nur über diesen Dämon auf den Drucker zugreifen können (also nicht mit `cat` direkt darauf drucken können). Dazu muß ein Druckerdämon als "Benutzer" in der Datei `/etc/passwd` eingetragen sein und die Gerätedatei dem Druckerdämon und/oder seiner Gruppe gehören. Für alle anderen Benutzer wird die Datei schreibgeschützt. Der Druckerdämon (`lpd`) muß dann entweder SUID oder SGID laufen und so dem Benutzer zur Ausführungszeit das Recht geben, in die Gerätedatei zu schreiben.
Auf die gleiche Weise kann den Systembenutzern der Zugriff auf die Floppylaufwerke (z. B. mit `mtools`) ermöglicht werden. Die Programme der "alten" `ps`–Suite gehören der speziellen Gruppe mem (memory) und verändern die effektive User–ID, um allen Anwendern den kontrollierten Zugriff auf den Kernelspeicher zu erlauben.

### Zugriffsrechte auf Verzeichnisse

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 werden 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).

### Set User ID für Verzeichnisse

Wie die einfachen Zugriffsrechte können auch für die Verzeichnisse SUID–, SGID– und Stickybit gesetzt werden.
Die Bedeutung des SGID–Bits ist naheliegend: vorausgesetzt, ein Anwender darf überhaupt eine Datei in dem entsprechenden Verzeichnis anlegen (das Verzeichnis muß beschreibbar sein), gehört diese Datei nicht der Gruppe des Erzeugers, sondern der Gruppe, der das Verzeichnis gehört. Eine entsprechende Funktion für das SUID–Bit gibt es nicht, weil die Änderung der Eigentumsrechte an einer Datei nur mit Root–Privilegien möglich ist.
Das Stickybit ist nur für ein Verzeichnis relevant und bedeutet hier, daß nur der Eigentümer einer Datei diese auch löschen darf. (Normalerweise kann jeder Benutzer Dateien in einem Verzeichnis löschen, auf das er Schreibzugriff hat.)

## 5.8 Recompilieren des Kernels

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 gerecht zu werden, 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 verschiedene 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.

### 5.8.1 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 xfz` *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.

### 5.8.2 Konfigurieren des Kernels

#### Das Konfigurationsscript

Da, wie schon oben gennant, nicht alle Treiber immer im Kernel benötigt werden, sollte der Kernel, um Speicher zu sparen, für 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:

#### Allgemeine Konfiguration

`Kernel math emulation (CONFIG_MATH_EMULATION) [y]`

Befindet sich ein mathematischer Coprozessor im System, oder wird der Kernel für einen 80486DX übersetzt, so kann auf diese Frage mit `n` geantwortet werden, da der Kernel keinen Coprozessor zu emulieren braucht.

`Normal harddisk support (CONFIG_BLK_DEV_HD) [y]`

Hier wird der Treiber für IDE, RLL, ESDI und MFM Festplatten konfiguriert. Befindet sich einer dieser Festplattentypen in Ihrem System, sollten Sie auf diese Frage mit `y` antworten.

`XT harddisk support (CONFIG_BLK_DEV_XD) [n]`

Sollten Sie in Ihrem Rechner noch zusätzlich einen alten XT–Harddisk-Controller verwenden wollen, so bietet sich hier die Möglichkeit, einen Treiber für alte 8–Bit–Controller einzubinden.

`TCP/IP networking (CONFIG_INET) [y]`

Soll der Rechner im Netz betrieben werden, sollte diese Frage mit `y` beantwortet werden.

`Limit memory to low 16MB (CONFIG_MAX_16M) [n]`

Auf vielen Rechnern ist der externe Cache–Speicher nur für 16MB ausgelegt. Auf diesen Systemen kommt es zu starken Geschwindigkeitseinbußen, wenn Linux mit mehr als 16MB RAM betrieben wird. Um dies zu vermeiden, kann dem Kernel mit dieser Option mitgeteilt werden, daß nur die ersten 16MB RAM verwendet werden sollen.

`System V IPC (CONFIG_SYSVIPC) [y]`

Stellt einige Möglichkeiten der Interprozeßkomunikation zur Verfügung, wie Semaphoren, Shared–Memory und Messages. Insbesondere der **dosemu** erfordert diese Option.

`Use -m486 flag for 486-specific optimizations (CONFIG_M486) [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.

### Binärformate anderer Betriebsysteme

`Elf executables (CONFIG_BINFMT_ELF) [y]`
`COFF executables (CONFIG_BINFMT_COFF) [y]`

Mit diesen Optionen kann die Kernelunterstützung für die Linkformate von SystemV R3 (COFF) und SystemV R4 (ELF) eingebunden werden. Linux ist damit in der Lage, auch Binärprogramme anderer Betriebssysteme zu laden.

### SCSI–Unterstützung

`SCSI support? (CONFIG_SCSI) [n]`

Wird hier `n` angegeben, so überspringt das Konfigurationsscript alle weiteren SCSI–Konfigurationsmöglichkeiten.

`Scsi disk support (CONFIG_BLK_DEV_SD) [y]`

Bestimmt, ob SCSI–Festplatten unterstützt werden. Die Angaben zu den entsprechenden Controllern werden später erfragt.

`Scsi tape support (CONFIG_CHR_DEV_ST) [y]`

Bindet die Unterstützung für SCSI–Bandlaufwerke in den Kernel ein.

`Scsi CDROM support (CONFIG_BLK_DEV_SR) [y]`

Hier kann die Unterstützung für SCSI–CD–ROM–Laufwerke eincompiliert werden. Diese Unterstützung ist nur für SCSI–CD–ROMs vorhanden, die über einen der nachfolgend konfigurierten Hostadapter angesteuert werden. Sie arbeitet nicht mit den 'pseudo' SCSI–Schnittstellen auf Sound– oder Videokarten. Eine Ausnahme ist hier die ProAudio–Spectrum–16, für die ein SCSI–Treiber konfiguriert werden kann.

`Scsi generic support (CONFIG_CHR_DEV_SG) [y]`

Bindet einen Treiber ein, der es Benutzerprogrammen erlaubt, SCSI–Geräte zu steuern.

### Treiber für spezielle SCSI–Hostadapter

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 AHA152X support (CONFIG_SCSI_AHA152X) [y]
Adaptec AHA1542 support (CONFIG_SCSI_AHA1542) [y]
Adaptec AHA1740 support (CONFIG_SCSI_AHA1740) [y]
Future Domain 16xx SCSI support (CONFIG_SCSI_FUTURE_DOMAIN) [y]
Generic NCR5380 SCSI support (CONFIG_SCSI_GENERIC_NCR5380) [y]
PAS16 SCSI support (CONFIG_SCSI_PAS16) [y]
ST-02 and Future Domain TMC-8xx support (CONFIG_SCSI_SEAGATE) [y]
Trantor T128/T128F/T228 SCSI support (CONFIG_SCSI_T128) [y]
UltraStor SCSI support (CONFIG_SCSI_ULTRASTOR) [y]
7000FASST SCSI support (CONFIG_SCSI_7000FASST) [y]
```

#### Netzwerkkarten, –geräte und –protokolle

`Network device support? (CONFIG_ETHERCARDS) [y]`

Auch wenn kein anderes Netzwerkdevice konfiguriert werden soll, sollte diese Frage mit "y" beantwortet werden, damit das "Loopback–Device" funktioniert.

`SLIP (serial line) support (CONFIG_SLIP) [n]`

`PLIP (parallel port) support (CONFIG_PLIP) [n]`

`NE2000/NE1000 support (CONFIG_NE2000) [n]`

`WD80*3 support (CONFIG_WD80x3) [y]`

`SMC Ultra support (CONFIG_ULTRA) [n]`

`3c501 support (CONFIG_EL1) [n]`

`3c503 support (CONFIG_EL2) [n]`

`3c509/3c579 support (CONFIG_EL3) [n]`

`HP PCLAN support (CONFIG_HPLAN) [n]`

`AT1500 and NE2100 (LANCE and PCnet-ISA) support (CONFIG_LANCE) [n]`

`AT1700 support (CONFIG_AT1700) [n]`

`D-Link DE600 pocket adaptor support (CONFIG_DE600) [n]`

`AT-LAN-TEC/RealTek pocket adaptor support (CONFIG_ATP) [n]`

#### CD–ROM–Treiber

`Sony CDU31A CDROM driver support (CONFIG_CDU31A) [n]`

Bindet einen Treiber für das Sony CDU31A CD–ROM–Laufwerk ein.

`Mitsumi CDROM driver support (CONFIG_MCD) [n]`

Bindet den Treiber für verschiedene Mitsumi CD–ROM–Laufwerke ein.

#### Dateisysteme

`Standard (minix) fs support (CONFIG_MINIX_FS) [y]`

Dieses Dateisystem sollte wegen seiner weiten Verbreitung auf jeden Fall in den Kernel eincompiliert werden, da es meistens auf Disketten verwendet wird. Es ist in der Partitionsgröße auf 64 MB beschränkt.

`Extended fs support (CONFIG_EXT_FS) [n]`

Dieses Dateisystem wird nur noch aus Gründen der Kompatibilität unterstützt. Auf neu installierten Systemen sollte es nicht mehr verwendet werden.

`Second extended fs support (CONFIG_EXT2_FS) [y]`

Das "Second extended Filesystem" ist das schnellste, leistungsfähigste und derzeit am weitesten verbreitete Linux–Dateisystem. Es erlaubt Partitionsgrößen bis zu 4GB und Dateigrößen bis zu 4GB. Eine Erweiterung auf Partitionsgrößen von 2TB steht in Aussicht. Eine Konvertierung wird ohne Neuformatieren möglich sein. Das ext2fs ist von vornherein auf gute Erweiterbarkeit ausgelegt und ist "das" Linux–Dateisystem schlechthin.

`xiafs filesystem support (CONFIG_XIA_FS) [n]`

Das xiafs ist stark an das Minix–Dateisystem angelehnt und älter als das ext2fs. Es unterstützt Partitionsgrößen bis zu 4 GB und Dateigrößen bis zu 64 MB. Dateinamen können bis zu 255 Zeichen lang sein.

`msdos fs support (CONFIG_MSDOS_FS) [y]`

Wenn MS–DOS Disketten oder Partitionen gemountet werden sollen, kann das MS–DOS–Dateisystem eincompiliert werden. Wird diese Frage mit `n` beantwortet, ist es trotzdem möglich, MS–Dos Disketten mit den mtools zu bearbeiten.

`/proc filesystem support (CONFIG_PROC_FS) [y]`

Das sog. Prozeßdateisystem bildet die interne Prozeßstruktur des Kernels auf das Dateisystem ab. Die neuen Versionen von ps, free, xsysinfo, u. a. verwenden dieses Dateisystem, um die entsprechenden Informationen zu lesen.

`NFS filesystem support (CONFIG_NFS_FS) [y]`

Wird das NFS–Dateisystem mit eingebunden, so können im Netz Verzeichnisse anderer Rechner gemountet werden.

`ISO9660 cdrom filesystem support (CONFIG_ISO9660_FS) [n]`

Das ISO9660 Dateisystem ist das Standarddateisystem für CD–ROMs. Linux unterstützt sowohl das 'reine' ISO9660 Dateisystem als auch die Rock Ridge Erweiterungen.

`OS/2 HPFS filesystem support (read only) (CONFIG_HPFS_FS) [n]`

Dieser Treiber ermöglicht das Lesen von OS/2 HPFS–Dateisystemen. Ein Beschreiben der Dateisysteme ist nicht möglich.

`System V and Coherent filesystem support (CONFIG_SYSV_FS) [n]`

Auch für SystemV R2 386, Xenix und Coherent stellt Linux Treiber zur Verfügung. Das System–V–Dateisystem ist bislang auf Disketten beschränkt.

### Konfiguration der zeichenorientierten Geräte

`Keyboard meta-key sends ESC-prefix (CONFIG_KBD_META) [y]`

Hier kann konfiguriert werden, ob vor einer Taste, die zusammen mit der ALT–Taste gedrückt wird, ein ESC gesendet wird.

`Keyboard Num Lock on by default (CONFIG_KBD_NUML) [y]`

Mit diesem Feature kann die Num–Lock–Taste beim Booten ein– oder ausgeschaltet werden. Dies ist in erster Linie bei Notebooks von Bedeutung.

`Parallel printer support (CONFIG_PRINTER) [y]`

Bindet die Unterstützung für die parallelen Schnittstellen ein.

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 (CONFIG_BUSMOUSE) [n]`

`PS/2 mouse (aka ''auxiliary device'') support (CONFIG_PSMOUSE) [y]`

`C&T 82C710 mouse port support (TI Travelmate)(CONFIG_82C710_MOUSE) [y]`

`Microsoft busmouse support (CONFIG_MS_BUSMOUSE) [n]`

`ATIXL busmouse support (CONFIG_ATIXL_BUSMOUSE) [n]`

`Selection (cut and paste for virtual consoles) (CONFIG_SELECTION) [y]`

Mit dieser Option wird die Kernelunterstützung für selection eingebunden. Selection ist ein Programm, das auf normalen Konsolen die Möglichkeit des "Copy–and–Paste" mit Hilfe der Maus bietet.

`QIC-02 tape support (CONFIG_TAPE_QIC02) [n]`

Compilert den Treiber für QIC–02 Bandlaufwerke in den Kernel ein.

### Multimedia

`Soundcard support (distributed separately) (y/n, default=n)?`

Bindet Treiber für AdLib, Soundblaster, ProAudio Spectrum und Gravis Ultrasound Soundkarten ein.

### Kernelhacking

`Kernel profiling support (CONFIG_PROFILE) [n]`

Wenn Programme mit Hilfe eines Profilers optimiert werden sollen, kann mit dieser Option Unterstützung für den Profiler in den Kernel eincompiliert werden.

`Verbose scsi error reporting (CONFIG_SCSI_CONSTANTS) [y]`

Stellt ausführliche Fehlermeldungen der SCSI–Treiber zur Verfügung. Wird diese Option ausgewählt, vergrößert sich der Kernel um ca. 12KB.

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.

#### Änderungen im Makefile

Ist die Konfiguration abgeschlossen, muß noch das Makefile geändert werden, da das Shellscript nicht alle Konfigurationsmöglichkeiten berücksichtigt. Zu konfigurieren bleiben noch das Rootdevice, der Tastaturtreiber, der Videomodus und die RAM–Disk. 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 RAM–Disk wird erzeugt, indem das Kommentarzeichen vor der Variable `RAMDISK` entfernt wird und in der dahinter stehenden Präprozessordirektive `-DRAMDISK=`*xxx* die gewünschte Größe angegeben wird.

### 5.8.3 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.

## 5.9 dosemu

### 5.9.1 Allgemeines

**Warnung!**

**Dosemu** ist ALPHA-Software, d. h. er befindet sich noch in der Testphase seiner Entwicklung. Sichern Sie deshalb ihre Daten, **bevor** Sie ihn starten, und verwenden Sie ihn **keinesfalls**, um für Sie wichtige Daten **ohne Sicherungskopie** zu bearbeiten.

**Funktion:**

**Dosemu** ist ein Programm, das auf Benutzerebene arbeitet und bestimmte Fähigkeiten des Linux-Kernels und des 80386 Prozessors ausnutzt, um MS–DOS innerhalb von Linux auszuführen. Der Name "**dosemu**" ist irreführend, da es sich nicht um einen MS–DOS–Emulator handelt, sondern um ein Programm, das die Hardwareumgebung, in der MS–DOS normalerweise arbeitet, nachbildet. Es handelt sich eigentlich um einen "PC-Emulator". Um spezielle Funktionen des **dosemu** verwenden zu können, muß **dosemu** mit Superuserrechten ausgeführt werden, da Linux "normalen" Benutzern keinen direkten Zugriff auf die Hardware gestattet. Dazu gehören die Verwendung des VGA-Bios, die direkte MS–DOS Bildschirmausgabe (mit Farben und IBM-PC Zeichensatz) sowie der direkte Zugriff auf die Festplatte. Trotzdem ist es möglich, **dosemu** auch mit normalen Benutzerrechten auszuführen. Die Bildschirmausgabe erfolgt dann über Betriebssystemfunktionen und die termcap-Library (d. h. **dosemu** kann auch über ein Terminal ausgeführt werden).

**Syntax:**

**dos** [`-234ACFHKNVbcefkstx`] [`-D`*Optionen*] [`-M` *Größe*] [`-P`*Datei*]

**Optionen:**

**-2** 286 CPU auswählen[16]

**-3** 386 CPU auswählen[16]

**-4** 486 CPU auswählen[16]

**-A** von dem ersten Floppylaufwerk booten

**-C** von der ersten Festplatte booten (normalerweise "`hdimage`")

**-F** ***Anzahl*** gibt die Anzahl der Floppylaufwerke an (1–4)

**-H** ***Anzahl*** gibt die Anzahl der Festplatten an (1 oder 2)

**-K** Tastaturinterrupt (int9) freischalten[17]

**-N** kein MS–DOS booten (nur zum Debuggen)

**-V** Einschalten der BIOS–VGA–Emulation (schließt -c und -b ein)

**-b** blendet das VGA-Bios in das Emulator-RAM ein

**-c** schaltet die direkte Bildschirmausgabe auf die Grafikkarte ein[18]

**-e** ***Größe*** stellt "*Größe*" Kilobytes EMS zur Verfügung

**-f** vertauscht die Angaben für die Floppylaufwerke A: und B:

**-k** schaltet den direkten Tastaturzugriff ein

**-s** verwendet den neuen Code zur Unterstützung verschiedener Bildschirmgrößen (funktioniert noch nicht)

[16] Die Auswahl der CPU hat nur einen Einfluß darauf, was Programme wie Checkit und Norton-Sysinfo als Prozessortyp angeben. Es können trotzdem nur Programme ausgeführt werden, die im Real-Mode arbeiten.

[17] einige residente Programme, wie z. B. Sidekick, erwarten diesen Interrupt

[18] ermöglicht den IBM-Zeichensatz, sowie DOS-Textmodus-Farben; erfordert Superuserrechte

**-t** schaltet den Timer-Interrupt (`int8`) frei[19]

**-x *Größe*** stellt "*Größe*" Kilobytes XMS zur Verfügung

**-D *Optionen*** setzt die Debugmaske; "+" schaltet die Ausgabe ein, "–" schaltet sie aus (`-D-a+dv` schaltet alle Debugausgaben außer denen über Plattenzugriffe und Bildschirmausgaben aus)

| | | | | | |
|---|---|---|---|---|---|
| d | Festplatte | v | Bildschirmausgabe | R | Lesezugriffe |
| k | Tastatur | i | Portzugriffe | W | Schreibzugriffe |
| s | RS-232 | p | Druckerport | h | Hardware |
| w | Warnungen | g | Allgemeines | x | XMS |
| m | Maus | I | IPC | E | EMS |
| c | Konfiguration | | | | |

**-M *Größe*** setzt die Speichergröße auf "*Größe*" Kilobytes

**-P *Datei*** kopiert die Debugausgaben in "*Datei*"

### 5.9.2 Voraussetzungen

Um den **dosemu** zu betreiben, muß Ihr System folgende Software-Voraussetzungen erfüllen:

1. einen Kernel Linux 0.99pl10 und neuer, jedoch muß bei der Übersetzung des Kernels die IPC-Option einkompiliert werden.[20]
2. den GNU C-Compiler gcc 2.2.2d7 oder neuer
3. die Linux C-Library ab der Version 4.2[21]
4. MS–DOS Version 3.3 oder neuer

### 5.9.3 Übersetzen des dosemu

Momentan werden die meisten Konfigurationen in der Datei `/etc/dosemu/config` vorgenommen. Bei der Übersetzung des **dosemu** muß lediglich die Grafikkarte und die gewünschte nationale Tastaturbelegung im Makefile eingetragen werden. Sie können ihre Konfiguration festlegen, indem Sie das "`#`"–Zeichen vor der von Ihnen gewünschten Zeile entfernen, und sicherstellen, daß alle anderen Konfigurationszeilen mit einem "`#`"–Zeichen beginnen. Um den **dosemu** zu übersetzen, sind folgende Schritte notwendig:

1. Ändern des Makefiles, um die gewünschte nationale Tastaturtabelle und Grafikkarte einzustellen.
2. Erzeugen einer Konfigurationsdatei `config.h` mit dem Kommando "`make config`"
3. Übersetzen und Installieren des Emulators mit den Kommandos "`make dep; make install`"

### 5.9.4 Die Laufzeitkonfiguration

Die Datei `/etc/dosemu/config` enthält den größten Teil der **dosemu**-Konfiguration. Sie besteht aus verschiedenen Anweisungen, die entweder allein durch ihre Anwesenheit bestimmte Konfigurationen erreichen, oder aus Schlüsselwörtern, die einen oder mehrere Parameter erwarten. Kommentare beginnen mit einem "#" und enden am Zeilenende.
Bei den Anweisungen und Schlüsselwörtern wird Groß– und Kleinschreibung nicht unterschieden. Bestimmte Argumente, wie z. B. Dateinamen und die Druckeroptionen, werden jedoch unverändert übernommen und sind deshalb auf eine **exakte** Schreibweise angewiesen.

[19] einige residente Programme wie z. B. Bildschirmschoner verwenden diesen Interrupt

[20] Es gehen ebenso 0.99pl7 oder 0.99pl9 mit ipcdelta und mmap Patches, sowie 0.99pl8 mit ipcdelta, mmap und ALPHA Patches. Einige Kernel von 0.99pl13a—0.99pl13s sind wegen Änderungen in der Prozeßtabelle ebenfalls ungeeignet.

[21] Achtung: die Version 4.3 ist wegen "Reentrancy"-Problemen ungeeignet. Dieser Fehler ist in den Versionen 4.3.2 und 4.3.3 behoben.

### Tastatur, Ports und andere Kleinigkeiten

**bootA, bootC**
Gibt an, von welchem Laufwerk der **dosemu** MS–DOS laden soll.

**cpu** *Typ*
Bestimmt, welcher Prozessor emuliert werden soll. Als "*Typ*" kann "`2`", "`3`" oder "`4`" angegeben werden. Es ist ebenfalls möglich, den ganzen Prozessornamen anzugeben ("`80286`", "`80386`" und "`80486`").

**DebugFlags** *Flags*
Die Syntax ist dieselbe wie in der Kommandozeile, d. h. "`DebugFlags -a+dv`" schaltet alle Debugausgaben bis auf die Platten- und Bildschirmausgabemeldungen ab.

**HogThereshold** *Microsekunden*
Gibt die Anzahl Microsekunden an, die mindestens zwischen zwei int16 verstreichen müssen, damit das Programm nicht als "keyboard-hog" betrachtet wird. Da der **dosemu** keine Möglichkeit hat, festzustellen, ob ein Programm "etwas tut" oder nur auf eine Tastatureingabe wartet, benutzt er diesen Zeitabstand, um eine Tastaturschleife zu erkennen. Der Standardwert ist 20000. Wenn der **dosemu** zu viel Rechenzeit verbraucht, kann dieser Wert erhöht werden. Ein zu hoher Wert resultiert in schlechter Performance.

**Keybint** *on/off*
Legt fest, ob der Tastaturinterrupt (int9) weitergeleitet wird. Der Standardwert ist "`off`".

**FastFloppy** *on/off*
Wird dieses Flag auf "`on`" gesetzt, so verwendet der **dosemu** eine schnelle, aber unsichere Zugriffsmethode für Diskettenlaufwerke. In diesem Fall sollte eine Diskette frühestens 5 Sekunden nach dem letzten Zugriff aus dem Laufwerk entfernt werden. Der Standardwert ist "`on`".

**MathCO** *on/off*
Über diese Option wird dem dosemu mitgeteilt, ob ein Koprozessor vorhanden sein soll. Diese Option hat nichts mit einem wirklich im Rechner installierten Koprozessor zu tun. Sie kann beliebig gesetzt werden und ist vor allem für MS–DOS–Programme interessant, die unbedingt einen Koprozessor erfordern.

**Ports** *Liste*
Erlaubt dem Emulator den Zugriff auf die in "*Liste*" angegebenen I/O-Ports. "*Liste*" ist eine von "{}" eingeschlossene, durch Leerzeichen getrennte Liste der freizugebenden I/O-Adressen. Die Adressen können entweder hexadezimal oder dezimal angegeben werden. Hexadezimale Adressen werden durch Voranstellen von "`0x`" vor die Adresse gekennzeichnet. Als Operator kann in der Liste der Operator "`range`" verwendet werden, um einen ganzen Bereich zu spezifizieren.
Die Zeile "`ports { 0x42 0x61 range 0x2f8 0x2ff }`" gibt die I/O-Adressen 0x42, 0x61, sowie alle Adressen von 0x2f8 bis 0x2ff frei.

**Speaker** *Modus*
Setzt die Betriebsart des Lautsprechers. Mögliche Modi sind:

1. "`off`" Lautsprecher aus
2. "`native`" Gibt den Zugriff auf die Ports 0x42 und 0x61 frei und ermöglicht so den "gewohnten DOS-Sound". Diese Einstellung sollte nur bei direktem Arbeiten auf der Konsole verwendet werden.
3. "`emulated`" Bei dieser Option wird das ASCII-Zeichen "BEL" (7) zum Terminal gesendet. Der Emulator versucht nicht, den "orginalen" DOS-Ton zu emulieren.

**RawKeyboard**
Wird diese Option gesetzt, so versucht der **dosemu**, die Tastatur direkt abzufragen.

**Timint** *on/off*
Legt fest, ob der Timerinterrupt (int8) weitergeleitet wird.

**Timer** *Frequenz*
Gibt die Frequenz des "Tick"-Timers in Hertz an. Der Standardwert is 18,2 Hz. Diese Option ist

als Debugging-Hilfe gedacht. **Achtung:** Der Bildschirmaufbau ist mit dem Timerinterrupt gekoppelt, deshalb ist es nicht sinnvoll, einen höheren Wert als "`18`" anzugeben. Falls der Bildschirmupdate zu schnell geht, kann hier ein kleinerer Wert ausprobiert werden.

**DOSmem** *Größe*
: Gibt die Menge des verfügbaren Speichers unter DOS an. Der Standardwert ist 640KB, der maximal mögliche ist 734K.

**EMS** *Größe*
: Stellt die angegebene Menge EMS 4.0 Speicher unter DOS zur Verfügung.

**XMS** *Größe*
: Richtet entsprechend der "Extended Memory Specification 3.0" die als "*Größe*" angegebene Menge XMS ein.

### Diskettenlaufwerke

**Dosemu** kennt zwei Arten von Laufwerken, Diskettenlaufwerke und Dateien, die als Diskettenlaufwerk angesprochen werden. Die Laufwerkskonfiguration wird durch das Schlüsselwort "`floppy`" eingeleitet. **Dosemu** ordnet die unter DOS verwendeten Laufwerksbuchstaben den Laufwerken in der Reihenfolge ihrer Konfiguration zu. Die Syntax der Konfiguration ist:
"`floppy { [Schlüsselwort` *Wert*`] ... [`*Optionen*`] ... }`"
Zulässige Schlüsselwörter sind:

**Device** *Blockdevice*
: Definiert das "*Blockdevice*" als Diskettenlaufwerk, z. B. "`/dev/fd0`".

**File** *Dateiname*
: Gibt an, daß die Datei "*Dateiname*" ein Bild einer Diskette enthält und als Diskettenlaufwerk konfiguriert werden soll. Kommt diese Konfiguration zum Einsatz, so sollten die Anzahl der Sektoren, Köpfe und Spuren angegeben werden, die das "originale" Laufwerk, auf dem die Vorlage für das Image erstellt wurde, hat.

**Sectors** *Anzahl*
: Definiert die Anzahl der Sektoren auf einer Spur des Laufwerks.

**Heads** *Anzahl*
: Gibt die Anzahl der Schreib- und Leseköpfe eines Laufwerks an.

**Tracks** *Anzahl*
: Beschreibt die Anzahl der Spuren, die ein Laufwerk umfaßt.

Als Optionen stehen zur Verfügung:

**threeinch**
: Das mit dieser Option bezeichnete Laufwerk wird im vom **dosemu** emulierten CMOS-RAM als 3,5–Zoll–Laufwerk angezeigt.

**fiveinch**
: Das mit dieser Option bezeichnete Laufwerk wird im vom **dosemu** emulierten CMOS-RAM als 5,25–Zoll–Laufwerk angezeigt.

**readonly**
: Erlaubt nur Lesezugriffe auf das Laufwerk.

### Beispiel

```
floppy { Heads 2 Sectors 18 Tracks 80 threeinch file /usr/dos/fimage }
floppy { Device /dev/fd0 threeinch }
```

Durch diese Einträge wird die Datei `/usr/dos/fimage` als Laufwerk A: und das "echte" erste Laufwerk des Rechners als Laufwerk B: konfiguriert. Im "CMOS-RAM" des **dosemu** tauchen beide Laufwerke als 3,5–Zoll–Laufwerke auf.

### Festplattenlaufwerke

Bei Festplatten gibt es verschiedene Zugriffsmethoden unter dem **dosemu**.

1. **Zugriff auf die gesamte Festplatte**
   Diese Zugriffsart erlaubt dem **dosemu**, die gesamte Festplatte anzusprechen (incl. des "Master Boot Records") und jede Partition dieser Festplatte. **Dosemu** versucht automatisch, die Festplattenparameter herauszufinden. Schlägt der Versuch fehl, müssen die Parameter manuell, wie bei den Diskettenlaufwerken beschrieben, angegeben werden. Diese Konfiguration ist extrem unsicher und sollte nur zu experimentellen Zwecken verwendet werden.

2. **Zugriff auf eine Partition der Festplatte**
   Diese Zugriffsart ist unter dem Gesichtspunkt der Implementierung ähnlich dem Zugriff auf die gesamte Festplatte und arbeitet ebenfalls über den DOS-Interrupt int13. Sie ist jedoch wesentlich sicherer, da der **dosemu** so keinen Zugriff auf andere Partitionen erhält und auch den "Master Boot Record" nicht ansprechen kann. Um diese Zugriffsart verwenden zu können, muß eine Datei namens "`/etc/dosemu/partition`" existieren, die die benötigten Zusatzinformationen für den **dosemu** enthält. Die Datei kann mit dem "mkpartition"-Kommando erstellt werden. Um z. B. die Datei für die erste Partition der ersten Festplatte zu erstellen, führen Sie das Kommando "`mkpartition /dev/hda 1`" aus.

3. **Eine als Festplatte "getarnte" Datei**
   Wie bei den Diskettenlaufwerken ist es auch möglich eine Festplatte durch eine Datei im Linux-Dateisystem zu simulieren. Festplattenimages werden mit dem Kommando "mkhdimage" erstellt. Mit dem Kommando "`mkhdimage -c 300 -h 12 -s 17 > hdimage`" erstellen Sie beispielsweise eine ca. 29MB große "Festplatte" auf ihrem Linux-Dateisystem.

Die Syntax der Festplattenkonfiguration ist derjenigen der Diskettenlaufwerke ähnlich.
"`disk { [Schlüsselwort` *Wert*`] ... [`*Optionen*`] ... }`"
Zulässige Schlüsselwörter sind:

**wholedisk** *Festplatte*
: Konfiguriert den Zugriff auf eine direkt angesprochene Festplatte. Als Festplatte kann z. B. "`/dev/hda`" angegeben werden.

**partition** *Partition Partitionsnummer*
: Erlaubt den direkten Zugriff auf eine Festplattenpartition. Der Parameter "*Partition*" bezeichnet ein Blockdepot, unter Linux z. B. "`/dev/hda1`", und der Parameter "*Partitionsnummer*" die Nummer der Partition `(1-4)`.

**image** *Dateiname*
: Konfiguriert eine Datei als Festplatte. Sie muß zuvor mit "mkhdimage"–Programm erstellt werden.

Ebenso können noch die Schlüsselwörter `Tracks, Sectors` und `Heads`, wie bei den Diskettenlaufwerken beschrieben, verwendet werden.

Zulässige Optionen sind:

**readonly**
: Erlaubt nur Lesezugriffe auf das Laufwerk

### Beispiel:

"`disk { partition /dev/hda1 1 readonly }`"
"`disk { image /usr/dos/hdimage }`"

Mit diesen Zeilen wird im **dosemu** die erste Partition der ersten Festplatte als Laufwerk C: konfiguriert. Laufwerk C: darf nur gelesen werden. Laufwerk D: wird durch die Image-Datei "`/usr/dos/hdimage`" emuliert.

#### Video-Konfiguration

Die Konfiguration der Bildschirmkarte wird durch das Schlüsselwort „`video`" eingeleitet und besteht aus einer von „{}" eingeschlossenen Liste.

Erlaubte Einträge sind:

**Chipset** *Art*
: Bezeichnet den auf der Grafikkarte verwendeten Chipsatz. Im Moment werden nur „`ET4000`" und „`TRIDENT`" erkannt.

**Chunks** *Anzahl*
: Bestimmt, in wievielen Teilen der **dosemu** Textbildschirmzeilen aktualisiert. Diese Option ist nur wirksam, wenn der **dosemu** nicht auf der Konsole läuft. Wird hier z. B. „`4`" angegeben, so aktualisiert der **dosemu** nur das Viertel einer Zeile, das sich geändert hat. Sie sollten vermeiden, hier einen zu großen Wert einzutragen, da der Aufwand, den Cursor zu positionieren, dann den Nutzen übersteigt. Der Standardwert ist „`1`".

**MemSize** *Größe*
: Gibt die Größe des Bildschirmspeichers an.

**VBIOS_file** *Datei*
: Benennt eine Datei, die eine Kopie des Video–BIOS enthält. Diese Option ist in erster Linie hilfreich, um mit verschiedenen BIOS-Typen zu experimentieren. Eine BIOS-Datei kann mit dem Kommando „`/etc/dosemu/getrom > vbios`" erzeugt werden.

**VBIOS_copy**
: Der **dosemu** kopiert das Video-BIOS in das Emulator-RAM. Er verwendet dazu die Datei „`/dev/mem`", d. h. er muß mit Superuserrechten ausgeführt werden.

**VBIOS_mmap**
: Der **dosemu** verwendet den mmap() Systemaufruf, um das Video-BIOS in das Emulator-RAM einzublenden. Diese Option verbraucht weniger Speicher als VBIOS_copy, erfordert aber ebenfalls Superuserrechte.

**Graphics**
: Erlaubt dem **dosemu**, in Grafikmodi zu schalten. Diese Option funktioniert bislang nur mit VGA-Karten. Sie erfordert ebenso, daß der Chipsatz korrekt angegeben wurde, da sonst evtl. erweiterte Textmodi von Super-VGA-Karten beim Wechseln der virtuellen Konsole oder beim Verlassen des **dosemu** nicht richtig wiederhergestellt werden.

**Console**
: Ermöglicht die direkte Bildschirmausgabe und Grafikdarstellungen. Diese Option wird ignoriert, wenn der **dosemu** nicht auf der Konsole läuft.

**FullRestore**
: Bei dieser Option sichert der **dosemu** alle Register der Grafikkarte und den gesamten Speicher, wenn die virtuelle Konsole gewechselt wird. Diese Option sollte gesetzt werden, wenn der **dosemu** parallel mit dem X-Window-System betrieben wird.

**PartialRestore**
: Ist diese Option gesetzt, wird bei einem Wechsel der virtuellen Konsole nur der Speicherbereich gesichert, der von Linux im Textmodus verändert wird.

### 5.9.5 Verlassen des Emulators

Um den **dosemu** zu verlassen, führen Sie das in der Distribution enthaltene Programm exitemu.com aus, oder schicken Sie von einer anderen virtuellen Konsole ein SIGHUP oder SIGTERM an den **dosemu**. Senden Sie ein SIGKILL nur im äußersten Notfall, da der **dosemu** dann nicht mehr die Bildschirmeinstellungen zurücksetzen und die Logfiles auf die Platte schreiben kann.

### 5.9.6 Die Tastaturbelegung

Folgende Tastenkombinationen stehen im "RAW-Keyboard-Mode" zur Verfügung:

**Strg-Rollen** schreibt die ersten 0x32 Interruptvektoren in die Debugausgabe.

**Alt-Rollen** zeigt die vm86-Register

**RechtsShift-Rollen** erzeugt einen int8 (timer tick)

**LinksShift-Rollen** erzeugt einen int9 (Tastatur)

**Strg-Alt-BildAuf** rebootet den **dosemu**

**Strg-Alt-BildAb** beendet den **dosemu**

**Pause** hält den Emulator an und startet ihn auch wieder

### 5.9.7 Der Laufwerk-Emulator emufs.sys

Mit dem Gerätetreiber emufs.sys kann ein beliebiges Verzeichnis im Linux-Dateisystem als "MS–DOS–Festplatte" verwendet werden. Auf diese Weise können Sie auch auf Ihre Linux–Dateien von MS–DOS aus zugreifen. Wesentlich interessanter ist jedoch die Möglichkeit, Ihre MS–DOS–Partition in Ihr Linux-Dateisystem zu mounten, um sie dann mit Hilfe von emufs.sys im **dosemu** als Festplattenlaufwerk anzusprechen.

Um emufs.sys verwenden zu können, müssen Sie ihn auf Ihre Bootdiskette kopieren und in der Datei `config.sys` eine Zeile der Form:

device=emufs.sys *Verzeichnis* [*Flag*]

hinzufügen. Als Flag kann "`R`" angegeben werden, um Schreibzugriffe zu unterbinden. Sie können emufs.sys mehrmals eintragen. Er vergibt automatisch den ersten freien Laufwerksbuchstaben. Wenn Sie z. B. zum Booten als Laufwerk A: die Datei `/usr/dos/fimage`, als Laufwerk B: `/dev/fd0` verwenden und keine Festplatte konfiguriert haben, stehen Ihnen mit den drei Einträgen

```
device=emufs.sys /usr/dos/dos1
device=emufs.sys /usr/dos/dos2
device=emufs.sys /usr/src/ R
```

das Verzeichnis `/usr/dos/dos1` als Laufwerk C:, das Verzeichnis `/usr/dos/dos2` als Laufwerk D: und `/usr/src` als Laufwerk E: mit allen Unterverzeichnissen zur Verfügung, wobei von Laufwerk E: nur gelesen werden darf. Möchten Sie Ihre MS–DOS Partition (z. B. /dev/hda1) als Laufwerk C: ansprechen, so mounten Sie sie unter Linux mit dem Kommando "`mount -tmsdos /dev/hda1 /usr/dos/dos1`" in Ihr Linux-Dateisystem.

# Kapitel 6

# Dämonen

Bereits bei den urtümlichen Vorfahren unseres heutigen Linux hat sich eine besondere Gruppe von Programmen entwickelt, die in symbiotischer Beziehung zum Betriebssystemkern stehen. Sie arbeiten automatisch und erfüllen immer wiederkehrende Aufgaben. Weil sie vom normalen Benutzer niemals direkt aufgerufen werden, also ihre Arbeit unsichtbar im Hintergrund erledigen, werden sie als Dämonen bezeichnet.

Eine der grundlegenden Ideen bei der Konzeption von Unix ist die Beschränkung auf die absolut notwendigen Funktionen. Daher der Name "Kernel" für das reine Betriebssystem. Die vielfältigen Funktionen und Dienste, die von einem Computer als universelle Maschine ausgeführt werden können, setzen als eigenständige Programme auf diesem Kern auf. Dieses Konzept erlaubt wesentlich vielseitigere Konfigurationen als die Verwirklichung einer begrenzten Menge zusätzlicher Funktionen im Kernel selbst.

Um bestimmte Dienste, die nicht im Kernel selbst implementiert sind, aber zu den weiteren Aufgaben eines Betriebssystems zählen, jederzeit anbieten zu können, warten die Dämonen im Hintergrund und treten erst in dem Moment in Aktion, wenn ihre speziellen Dienste angefordert werden.

## 6.1 Der Druckerdämon lpd

Ein typischer Dämon ist der lpd, der Line Printer Dämon. Dieser Dämon kann die Druckaufträge (Jobs) eines Anwenders entgegennehmen und sie auf einen gegeigneten Drucker weiterleiten. Gegenüber dem ebenfalls möglichen Verfahren der direkten Übertragung an einen Drucker, beispielweise mit dem cat–Kommando, hat die Benutzung des Dämons eine Reihe von Vorteilen.

- Der Dämon kann immer Druckaufträge annehmen, auch wenn kein Drucker im System frei ist (Spooler).
- Mit Hilfe des Druckerdämons kann ein Druckauftrag in einem Netzwerk auf jedem geeigneten angeschlossenen Drucker ausgeführt werden. Insbesondere können beispielsweise auch alle Druckjobs von allen Rechnern eines Netzes auf einem einzigen Drucker zentral ausgeführt werden.
- Ein Druckerdämon kann mit sogenannten Filtern sehr unterschiedliche Druckvorlagen für jeden Drucker korrekt aufbereiten.

Als ordentlicher Dämon tritt der lpd nicht direkt in Erscheinung. Für den Anwender sind die Kommandos lpr, lpq und lprm vorgesehen.[1]

Mit dem lpr–Kommando können Sie ein Dokument an den Druckerdämon übergeben. Damit wird der im Hintergrund schlafende Dämon aktiviert, der das Dokument in das Spoolverzeichnis kopiert und eine zusätzliche Befehlsdatei schreibt. Auf diese Weise entsteht ein Druckjob, der sich in die Warteschlange (Queue) des Druckerdämons einreiht.

[1] Die Anwenderprogramme kommunizieren über ein Socket mit dem Druckerdämon. Weil dieses Socket ein sogenannter "Well Known Service" ist (515/TCP), können auch andere Programme über das Netzwerk mit dem Druckerdämon kommunizieren.

Mit dem Kommando lprm (line printer remove) kann ein abgeschicker Druckjob angehalten werden. Das lpq–Kommando zeigt den aktuellen Status der Druckerwarteschlange, das ist die Liste aller unbearbeiteten Jobs.

Die Systemverwalterin kann mit dem lpc (line printer control) Kommando den Status des Druckerdämons überprüfen und verändern.

### 6.1.1 Den lpd erziehen

Erwartungsgemäß muß ein Dämon, der seine Arbeit ja ohne direkten Kontakt mit dem Benutzer erledigt, von der Systemverwalterin konfiguriert werden.

Dazu dient die Datei `/etc/printcap`, in der das grundsätzliche Verhalten des Dämons eingestellt werden kann, sowie eine Reihe von Filtern, mit denen der Dämon aus rohen Texten wohlgesetzte Dokumente destilliert.

Wenn der Druckerdämon korrekt läuft, sollte er in der Datei `/etc/rc.local` (oder in einer vergleichbaren Datei) beim Systemstart automatisch aufgerufen werden.

Um den Druckerdämon netzwerkfähig zu machen, müssen alle Rechner, von denen der Zugriff auf den lokalen Drucker erlaubt werden soll, in der Datei `/etc/hosts.equiv` oder `/etc/hosts.lpr` aufgeführt werden.

In einer Datei mit dem Namen `minfree` im Spoolverzeichnis des Dämons kann eine Anzahl Plattenblöcke festgelegt werden, die vom Dämon freigehalten wird.

#### /etc/printcap

Die printcap[2]–Datei beschreibt das Spoolsystem und die Druckerfilter.

Jede Zeile, die nicht von einem # eingeleitet wird, definiert einen Drucker im System. Wenn ein Eintrag über mehrere Zeilen gehen soll, muß das Zeilenende durch einen Backslash versteckt werden. Ein Eintrag besteht aus mehreren Feldern, die durch Doppelpunkte voneinander getrennt werden. Das erste Feld enthält den Namen, unter dem der Drucker angesprochen wird. Die anderen Felder belegen Variable oder setzen Schalter zur Einstellung des Druckerdämons. Die Zuweisung an Zeichenvariable erfolgt mit dem Gleichheitszeichen, numerische Variable werden mit dem Nummernzeichen # belegt.

Die folgenden Variablen können mit Zeichenketten (Wörter, Namen) belegt werden.

**lp=** Die Gerätedatei des Druckers. Für Netzwerkdrucker an anderen Hosts muß diese Variable leer definiert werden, damit der voreingestellte Drucker gelöscht wird. Die Voreinstellung ist meistens `/dev/lp`.

**rm=** Der Name des Remote Host, an dem der Netzwerkdrucker angeschlossen ist.

**rg=** Die Benutzergruppe, deren Mitglieder den Drucker benutzen dürfen.

**rp=** Name des Druckers am Remote Host. Voreingestellt ist der Standarddrucker lp.

**lo=** Name der Lockdatei. Voreinstellung ist `lock`. Das Lockfile wird im Spoolverzeichnis des Druckers vom Druckerdämon (Kind) angelegt.

**st=** Name des Statusfiles. Voreinstellung ist `status`. Die Statusdatei wird vom Druckerdämon (Kind) im Spoolverzeichnis des Druckers angelegt.

**sd=** Name des Spoolverzeichnisses. Voreinstellung ist `/usr/spool/lpd`.

**af=** Name der Abrechnungsdatei. Der Name dieser Datei wird den Eingabefiltern übergeben, die dort Informationen über den Umfang des Druckjobs ablegen sollten, mit denen die Systemverwalterin die Druckkosten mit den Benutzern abrechnen kann.

**ff=** Die Zeichenkette für den Seitenvorschub.

**tr=** (trailer) Diese Zeichenkette wird nach Beendigung des letzten Druckjobs ausgegeben.

**lf=** Name des Logbuchs. In dieser Datei werden die Fehlermeldungen des Druckerdämons gespeichert.

[2] Der Name erinnert an termcap, und tatsächlich ist das Format der Einträge dieser Dateien vergleichbar. Die Funktion der Einträge ist aber sehr verschieden. Insbesondere können in der printcap–Datei **keine** Steuersequenzen für den Drucker definiert werden.

Die folgenden Variablen bestimmen die Filter zur Aufbereitung der zu druckenden Dateien.

**of=** Der Name des Ausgabefilters.

**if=** Name des (Standard–) Eingabefilters

**rf=** Name des FORTRAN–Textfilters. Dieser Filter wird von lpr mit der -r Option ausgewählt.

**tf=** Name des troff–Filters. Dieser Filter wird von lpr mit der Option -t ausgewählt.

**nf=** Name des ditroff–Filters. Dieser Filter wird von lpr mit der Option -n ausgewählt.

**df=** Name des TEX –Filters. Dieser Filter wird von lpr mit der Option -d ausgewählt.

**gf=** Name des graph–Filters. Dieser Filter wird von lpr mit der Option -g ausgewählt.

**vf=** Name des Rastergrafik–Filters. Dieser Filter wird von lpr mit der Option -v ausgewählt.

**cf=** Name des cifplot–Filters. Dieser Filter wird von lpr mit der Option -c ausgewählt.

Die folgenden Variablen können mit numerischen Werten (Zahlen) belegt werden.

**mx#** Die maximale Blockzahl (Größe) eines Druckjobs.

**mc#** Die maximale Anzahl erlaubter Kopien eines Druckjobs.

**pw#** Die (maximale) Spaltenzahl einer Druckseite.

**px#** Die Breite einer Druckseite in Pixeln.

**pl#** Die (maximale) Zeilenzahl einer Druckseite.

**py#** Die Höhe einer Druckseite in Pixeln.

Die folgenden Schalter verändern das Verhalten des Druckerdämons.

**rs** Der Zugriff auf einen Netzwerkdrucker ist nur denjenigen Anwendern gestattet, die einen Account auf dem Host mit dem Drucker haben.

**sc** Verbietet mehrfachen Ausdruck eines Dokumentes.

**sf** Unterdrückt die Ausgabe eines Seitenvorschubs nach jedem Druckjob.

**sh** Unterdrückt die Ausgabe einer Titelseite vor jedem Druckjob.

**sb** Verkürzt die Titelseite auf eine Zeile.

**hl** Veranlaßt den Druckerdämon, die Titelseite nach jedem Druckjob zu schreiben.

**rw** Öffnet den Drucker zum Lesen und Schreiben.

Die folgenden numerischen Variablen stellen die Schnittstelle für serielle Drucker ein.

**br#** Die Baudrate, wenn der Drucker an einer seriellen Schnittstelle hängt.

**fc#** Löscht die im Argument gesetzten Bits.

**fs#** Setzt die im Argument gesetzten Bits.

**xc#** Löscht die im Argument gesetzten local–mode–Bits.

**xs#** Setzt die im Argument gesetzten local–mode–Bits.

Ein typischer Eintrag sieht beispielsweise so aus:

```
# Standarddrucker an lp1 mit Spoolverzeichnis lp1 und Filter lpf
lp:lp=/dev/lp1:sd=/var/spool/lp1:of=/usr/lib/lpf:lf=/var/adm/lp_err:
```

In diesem Beispiel wird ein Drucker an der ersten parallelen Schnittstelle `/dev/lp1` unter dem Namen `lp` definiert. Als Spoolverzeichnis (sp) wird `/usr/spool/lp1` verwendet, als Ausgabefilter (of) wird das Programm lpf festgelegt, durch die Zuweisung des Wertes 0 an die Variable `mx` werden beliebig große Dokumente zugelassen, und als Logfile wird die Datei `/usr/adm/lpd-errs` bestimmt. Vorausgesetzt, die genannten Verzeichnisse und Dateien existieren, sollte mit diesem Eintrag der erste Probeausdruck möglich sein.

Eine kleine Veränderung des Beispieleintrages veranlaßt den Druckerdämon dazu, das Dokument über das TCP/IP–Netzwerk an einen anderen Rechner zu schicken und von dem dortigen Druckerdämon verarbeiten zu lassen:

```
# Standarddrucker ueber Netz an Rechner soho
lp:lp=:rm=soho:lf=/usr/adm/lpd-errs:
```

### 6.1.2 Die Druckerfilter

Mit Ausnahme des Papierformates sind in der printcap–Datei keine Informationen über die genaue Ansteuerung des Druckers gespeichert. Das bedeutet, daß der Druckerdämon nicht einmal einen Reset oder eine Druckerinitialisierung selbst ausführen kann. Trotzdem kann ein gut eingerichteter lpd eine Vielzahl unterschiedlicher Formate auf jedem angeschlossenen Drucker ausgeben, vorausgesetzt, die Systemverwalterin hat die passenden Filter installiert.

Der lpd kann einen Ausgabefilter (of) und bis zu neun Eingabefilter (cf, df, gf, if, nf, rf, tf, vf und pr, siehe oben) benutzen.

#### Der Ausgabefilter

Der Ausgabefilter ist in erster Linie dazu gedacht, die Ausgabe der vom lpd selbst generierten Titelseiten am Beginn jedes Druckjobs zu filtern. Weil eine eventuell notwendige Druckerinitialisierung bereits für diese Titelseite sinnvoll ist, sollte der Ausgabefilter auch diese Aufgabe erfüllen.

Das Zusammenspiel des Ausgabefilters mit dem lpd ist insofern etwas kompliziert, als dieser nur einmal zu Beginn des ersten Druckjobs gestartet wird. Für den Ausdruck der eigentlichen Dokumente benutzt der Druckerdämon einen der Eingabefilter und hält den Ausgabefilter bis zum Beginn des nächsten Jobs an. Dazu schreibt der Dämon die Zeichenkette '\031\1' in die Standardeingabe des Ausgabefilters. Der Filter muß sich daraufhin selbst mit SIGSTOP anhalten und auf das SIGCONT–Signal warten.

#### Die Eingabefilter

Die Zahl möglicher Formate, in denen ein Dokument vorliegen kann, und die Zahl möglicher Druckersprachen, in die dieses Format umgewandelt werden muß, um tatsächlich auf dem Papier zu erscheinen, ist so groß, daß es keine einfache Methode gibt, diese Übersetzung direkt vom Druckerdämon ausführen zu lassen.

Deshalb muß die Systemverwalterin dem lpd für jedes Eingabeformat einen Eingabefilter bereitstellen, der dieses Format für den installierten Drucker aufbereitet.

Das Zusammenwirken des lpd mit den Eingabefiltern ist sehr einfach. Mit einer der Optionen c, d, f, g, n, p, t und v gibt der Benutzer beim Aufruf von lpr an, welches Format sein Dokument hat, und bestimmt so, welcher Eingabefilter benutzt wird. Dieser Filter wird dann vom lpd für den Ausdruck des bei diesem Aufruf von lpr bestimmten Dokumentes in einer Pipeline gestartet. Der Dämon ruft den if–Filter mit folgender Kommandozeile auf:

```
if [-c] -wSpalten -lZeilen -i Einrückung -n Name -h Host Accountdatei
```

Die -c Option ist gesetzt, wenn der lpr mit der Option -l aufgerufen wurde. Alle anderen Eingabefilter werden mit der Zeile

```
filter -x Breite -y Länge -n Name -h Host Accountdatei
```

aufgerufen.

Während ein Eingabefilter aktiv ist, bleibt der Ausgabefilter angehalten. Der Eingabefilter liest aus der Pipeline (Standardeingabe), verändert den Datenstrom seinem Programm entsprechend und schreibt die Daten in die Standardausgabe, die direkt mit dem Drucker verbunden ist.

So ein Filter kann ein vollständiges C–Programm sein. Das dem lpd–Paket beiliegende Programm lpf ist beispielsweise ein Eingabefilter speziell für groff–Dokumente.

Als Eingabefilter können aber auch Shellscripts oder perl– Programme verwendet werden. Auf diese Weise kann zum Beispiel die Umwandlung einer Postscriptdatei für einen Epson–Nadeldrucker mit dem Ghostscript–Interpreter gs erfolgen:

```
#!/bin/bash
#
gs -q -dNOPAUSE -sDEVICE=EPSON -sOutputFile=- -
```

Ein schönes Beispiel für die vielfältigen Möglichkeiten, einen Filter zu bauen, ist das magic-filter–Script von Brian McCauley. Dieses Shellprogramm nutzt das file–Kommando zur Erkennung des Formats einer Eingabedatei und wählt automatisch den passenden Filter für die weitere Bearbeitung.[3]

```
#!/bin/bash
# BAM's magic-filter
# (C) Brian McCauley (B.A.McCauley@bham.ac.uk) 1993

# Der Druckerdämon hat nicht unbedingt eine brauchbare PATH Umgebungsvariable
PATH=$PATH:/usr/bin:/bin:/usr/local/bin

# Das magic-filter Script kann durch Links unter verschiedenen Namen aufgerufen
# werden. Dadurch kann das Verhalten des Filters durch die Optionen (Schalter)
# des 'lpr' Kommandos bestimmt werden.
basename=${0##*/}


# Mit dem im Paket des magic-filter enthaltenen C-Programm rewind-stdin wird
# sichergestellt, daß der Filepointer am Anfang der Eingabedatei steht. Wenn
# die Eingabe nicht aus einer Datei kommt, sondern aus einer Pipeline, schlägt
# das Zurücksetzen fehl.

if rewind-stdin ; then
  tmpfile=""
else
  # In diesem Fall wird der Name für eine garantiert neue temporäre Datei
  # bestimmt. (Dieses Script kann rekursiv aufgerufen werden.)
  while tmpfile=/tmp/$basename.$$.$RANDOM; test -e $tmpfile; do :; done
fi

# Mit dem 'file'-Kommando wird versucht, das Format der Druckdatei festzustellen.

if [ -z "$tmpfile" ] ; then
  magic=`file -`
else
  # Wenn die Datei durch eine Pipeline kommt, wird das erste Kilobyte
  # gespeichert und zur Identifizierung an das 'file' Kommando gegeben.
  dd bs=1 count=1024 of=$tmpfile 2>/dev/null
  magic=`file $tmpfile`
fi

# Der Dateiname wird von der Ausgabe von 'file' entfernt.

magic=${magic#*:\ }

filter=""
prefilter=""
nopipe=0

# Der folgende Block wird von Brian McCauley als separates Script ausgeführt.
# Das erleichtert das Einfügen neuer Filter für andere Formate. Die Anführungs-
# zeichen um die Ausdrücke mit den Wildcards in diesem Beispiel müssen ab der
# bash-Version 1.13 entfernt werden!

# Hier werden in Abhängigkeit von dem durch das 'file'-Kommando bestimmten
# Format und die Optionen beim Aufruf von 'lpr' die passenden Filter bestimmt.
```

[3] Abdruck mit freundlicher Genehmigung von Brian McCauley. Kommentare im Programm von Sebastian Hetze.

```
case $magic in

  # Wenn die Druckdatei komprimiert ist, wird ein prefilter zum Entpacken
  # zwischengeschaltet.

  "block compressed data*" | "gzip compressed*" ) prefilter="gzip -d" ;;

  # Der Name, unter dem der magic-filter aufgerufen wird, variiert in
  # Abhängigkeit von den Optionen, mit denen `lpr' aufgerufen wurde.

  "*" ) case $basename in
    # This is my example
    "lp.df" ) case $magic in
      "DVI*" ) filter=${0%/*}/sample-filters/dvi2epson; nopipe=1 ;;
      "PostScript*" ) filter=${0%/*}/sample-filters/ps2epson ;;
      "TeXinfo source" ) filter=${0%/*}/sample-filters/texi2epson ;;
    esac ;;
    # Hier ist der Eintrag für den if-Inputfilter, der aufgerufen wird, wenn
    # keine der Optionen für `lpr' benutzt wurde.
    "lp.if" ) filter=${0%/*}/text-filter ;;
  esac ;;
esac


# Wenn die Datei als komprimiert erkannt ist, wird sie in diesem Schritt durch
# das prefilter-Programm entpackt, um anschließend wieder rekursiv durch den
# magic-filter bearbeitet zu werden.

if [ -n "$prefilter" ]; then
  if [ -z "$tmpfile" ]; then
    rewind-stdin
    eval $prefilter | $0 $*
  else
    ( cat $tmpfile; rm $tmpfile; cat ) | \
    eval $prefilter | $0 $*
  fi
  exit
fi

# Wenn ein Filter für das Dokumentformat bestimmt werden konnte, wird jetzt das
# Dokument tatsächlich bearbeitet.

if [ -z "$filter" ] ; then
  echo $0: "Can't handle $magic" >&2
else
  if [ -z "$tmpfile" ]; then
    rewind-stdin
    exec $filter $*
  else

    # Wenn der Filter nicht aus einer Pipeline lesen kann, muß der Rest der
    # Druckdatei in der temporären Datei zwischengelagert werden. (Die ersten
    # 1024 Bytes wurden bereits zur Identifizierung gespeichert.

    if [ "$nopipe" = 1 ]; then
      cat >>$tmpfile
      $filter $* <$tmpfile
    else
      ( cat $tmpfile; rm $tmpfile; cat ) | $filter $*
      exit
    fi
  fi
fi

# Wenn am Ende noch eine in dieser Instanz des magic-filter-Scripts erzeugte
# temporäre Datei existiert, wird sie gelöscht.
```

```
if [ -n "$tmpfile" ] ; then
  rm $tmpfile
fi
```

## 6.2 Der Batchdämon crond

Der vielseitigste aller Dämonen ist der crond (gesprochen cron-d). Er führt beliebige Kommandos automatisch zu vorbestimmten Zeitpunkten aus, wie nach einem Fahrplan. Einmal pro Minute sieht der Dämon in seinen Terminkalender und führt zuverlässig und pünktlich alle anstehenden Kommandos aus.

Der unter Linux verbreitete crond von Paul Vixie (vixie-cron) erlaubt prinzipiell allen Benutzerinnen, mit dem crontab–Kommando eine eigene Spalte in dem Terminkalender des Dämons einzurichten. Die in dieser Spalte eingetragenen Befehle werden dann "im Auftrag", also mit der Benutzerkennung der Auftraggeberin, ausgeführt.

Wie der Druckerdämon lpd wird der crond normalerweise während der Systeminitialisierung aus einer der `rc*`–Dateien aufgerufen. Der Dämon geht von selbst in den Hintergrund.

Der Terminkalender des Dämons befindet sich im Verzeichnis `/usr/spool/cron/crontabs` und besteht aus beliebig vielen Spalten, jede in Form einer Datei mit dem Namen der Auftraggeberin.

### 6.2.1 Der Terminkalender crontab

Die einzelnen Dateien für den Terminkalender des Dämons werden von den Auftraggeberinnen verwaltet. Es handelt sich dabei um einfache ASCII–Textdateien, die beispielsweise mit dem elvis–Editor erzeugt werden können. Weil das `./crontabs`–Verzeichnis des Dämons für normalsterbliche Systembenutzerinnen nicht beschreibbar ist (es ist normalerweise nicht einmal lesbar), muß eine Vorlage für die Termindatei in einem anderen, beschreibbaren Verzeichnis angelegt werden. Die Veränderung des `./crontabs`–Verzeichnisses wird dann vom crontab–Kommando mit Superuser-Rechten vorgenommen.

#### Die crontab–Datei

In einer Termindatei werden alle Zeilen, die weder leer sind noch mit einem #–Zeichen in der ersten Spalte als Kommentar markiert sind, vom Dämon bearbeitet. Solche Zeilen können entweder eine Umgebungsvariable für die Ausführung aller in dieser Datei aufgerufenen Kommandos definieren oder eine Zeitmaske mit zugehörigen Kommando enthalten.

Eine Zeitmaske besteht aus fünf Feldern, die durch Leerzeichen voneinander getrennt werden.

**Die Zeitmaske:**

| Nr. | Bedeutung | Bereich |
|---|---|---|
| 1 | Minute | 0-59 |
| 2 | Stunde | 0-23 |
| 3 | Monatstag | 0-31 |
| 4 | Monat | 0-12* |
| 5 | Wochentag | 0-7* |

*oder Namen

Die Monate und Wochentage können auch mit ihren englischen Namen angegeben werden. Die Namen können auf drei Zeichen abgekürzt werden, Groß–/Kleinschreibung wird ignoriert.

Bei Zahlendarstellung des Wochentages entsprechen 0 und 7 dem Sonntag. Jedes Feld der Zeitmaske kann durch einen Asterisk '*' belegt werden, der auf jeden Termin paßt.

Die Felder können durch Komma getrennte Listen von Zeiteinträgen sowie Bereiche der Form *von–bis* enthalten.

In Bereichen dürfen keine Namen verwendet werden. Es können auch mehrere Bereiche aufgelistet werden. Zusätzlich können den Bereichen noch "Teiler" zur Veränderung der Schrittweite nachgestellt werden.

Die restliche Zeile bis zum Zeilenende oder einem %–Zeichen wird als Kommando ausgeführt. Wenn ein %–Zeichen gefunden wird, das nicht durch einen Backslash entwertet ist, wird der Rest der Zeile als Eingabe an das Kommando geleitet.

#### Beispiele:

`0 8-18 0 * 1-5 /usr/lib/newsbin/input/newsrun`
Das newsrun–Kommando wird montags bis freitags von 8 bis 18 Uhr zu jeder vollen Stunde aufgerufen.

`0 17 24 12 * echo %schoene Bescherung`
Heiligabend um 17 Uhr, egal was für ein Wochentag. Die Mitteilung wird per Mail an den Auftraggeber geschickt. In diesem Beispiel wird der Mitteilungstext nicht in der Kommandozeile an das echo–Kommando übergeben, sondern in seinen Standardeingabekanal geschrieben.

`0-59/5 * * * ~/bin/remind`
Das remind–Kommando wird alle fünf Minuten aufgerufen, jeden Monat, jeden Tag, jede Stunde ...

`0 6,10,14,18,22 * * * /usr/lib/uucp/uucico -s uucphost`
Das uucico–Kommando wird täglich um 6, 10, 14, 18 und 22 Uhr auf– und damit der `uucphost` angerufen.

Ein bestimmter Tag kann sowohl als Monatstag als auch als Wochentag bestimmt werden. Wenn beide Felder bestimmt, also nicht durch einen Asterisk besetzt sind, werden sie durch eine logische ODER–Verknüpfung ausgewertet. Eine Warnung der Form

`0 0-23 13 * fri wall %Achtung, Freitag der 13. *** FALSCH ***`

würde also jeden Freitag und jeden 13. eines Monats ausgegeben.

### Umgebungsvariable in der crontab–Datei

Wie bereits erwähnt, können in der `crontab`–Datei auch Umgebungsvariable für die Ausführung der Kommandos bestimmt werden. Die Variablen HOME, LOGNAME und SHELL werden von crond automatisch mit den Werten aus der `passwd`–Datei vorbelegt. Die Variable LOGNAME kann nicht verändert werden.

In der MAILTO–Variablen kann ein realer Account bestimmt werden, an den die Ausgabe der automatischen Kommandos geschickt wird. Wenn die Variable definiert, aber leer ist, geht die Ausgabe der Kommandos verloren. Wenn die Variable nicht definiert ist, wird die Ausgabe automatisch an den Eigentümer der Terminkalenderspalte gesendet.

Die Definition von Umgebungsvariablen erfolgt wie in normalen Shellscripts durch Zuweisung einer (möglicherweise leeren) Zeichenkette.

### Das crontab–Kommando

Mit dem Anwenderkommando crontab werden die benutzerdefinierten Termindateien im Verzeichnis `/usr/spool/cron/crontabs` verwaltet.

crontab erkennt folgende Optionen:

**-u** ***Benutzer*** (user) legt den Benutzernamen fest, in dessen Auftrag eine Termindatei ausgeführt werden soll. Diese Option ist nur dem Superuser (root) zugänglich. Alle normalsterblichen Systembenutzer können (höchstens) ihre eigenen Termindateien bearbeiten.

**-l** (list) zeigt den Inhalt der aktuellen Termindatei an.

**-d** (delete) löscht die Termindatei aus dem Verzeichnis `./crontabs`.

**-r** ***Datei*** (replace) ersetzt die Termindatei im `./crontabs`–Verzeichnis durch die angegebene.

Wenn im Verzeichnis `/usr/spool/cron` die Datei `allow` existiert, kann das crontab nur von den darin aufgeführten Systembenutzern ausgeführt werden. Wenn anstelle der `allow`–Datei eine Datei mit dem Namen `deny` existiert, steht das crontab–Kommando allen Systembenutzern zur Verfügung, die NICHT darin aufgeführt sind.

Wenn keine der Dateien existiert, hängt das Verhalten von crontab von den Einstellungen bei der Compilierung ab. Entweder kann jeder das crontab–Kommando ausführen, oder nur der Superuser (root).

Wenn das `./crontabs`–Verzeichnis vom crontab–Kommando verändert wird, liest der crond automatisch die neuen Daten, muß also nicht extra neu gestartet werden[4].

[4] Der crond erkennt die Veränderung an dem mtime–Eintrag in der Inode des Verzeichnisses. Dieser Eintrag existiert im Minix–Dateisystem und im alten Extended Dateisystem nicht. Wenn das `/usr/spool/crontab`– Verzeichnis in einem dieser Dateisysteme angesiedelt ist, muß es also neu gestartet werden, damit die veränderten Dateien gelesen werden.

## 6.3 Der Protokollschreiber syslogd

Das Betriebssystem (der Kernel) und seine Prozesse auf der Benutzerebene (Dämonen, Kinder von init) sind mit keinem Benutzerterminal direkt verbunden. Diese Selbständigkeit wird dann zu einem Problem, wenn ein solcher Prozeß eine Nachricht, beispielsweise eine Fehlermeldung, ausgeben will/muß. Die Kanäle für die Standardausgabe und die Standardfehlerausgabe sind bei den Systemprozessen mit der Gerätedatei `/dev/console` verbunden. Die Ausgabe für dieses Gerät wird auf dem Bildschirm der Systemkonsole direkt ausgegeben; unter X11 kann sie im xconsole–Fenster angezeigt werden.

Im Mehrbenutzerbetrieb ist die Methode, eine Systemmeldung ausschließlich als Fehlermeldung auf die Systemkonsole zu schreiben, unbefriedigend. Es kann nicht sichergestellt werden, daß die Meldung von der "richtigen" Person gelesen wird. Die Bildschirmmeldungen lassen sich nicht sichern und gehen so extrem leicht verloren.

Eine sehr umfassende Lösung dieses Problems bietet der syslogd (gesprochen süslog-d) an. Allerdings muß dieser Lösungsweg bereits bei der Programmierung des Systemprogramms eingeschlagen werden. Wegen der eindeutigen Vorteile arbeitet bereits ein großer Teil der Linux–Systemprogramme mit dem syslogd zusammen.

Anstelle der direkten Ausgabe einer Meldung mit fprintf(3) oder einer vergleichbaren C–Bibliotheksfunktion können Systemmeldungen von spezieller Bedeutung mit der syslog(3)–Funktion ausgegeben werden, die Bestandteil der Standardbibliothek des gcc für Linux ist.

Solche Meldungen werden dann automatisch vom syslogd entgegengenommen.[5]

Der syslogd erhält die Nachrichten über das Socket `/dev/log` und über die Spezialdatei `/proc/kmsg`. Er liefert die Meldung nach den Regeln aus, die die Systemverwalterin in der Konfigurationsdatei `/etc/syslog.conf` festgelegt hat.

### Die Datei /etc/syslog.conf

Der Dreh– und Angelpunkt der syslog–Installation ist die Datei `/etc/syslog.conf`. In dieser Konfigurationsdatei werden für verschiedene Klassen von Nachrichten verschiedene Auslieferungswege bestimmt.

Zur Klassifizierung gibt die Systemprogrammiererin jeder Syslognachricht zwei Merkmale: Herkunft und Priorität.

Für jede Herkunftskategorie können Nachrichten ab einer bestimmten Priorität auf einen Auslieferungsweg geschickt werden.

Als Herkunft kommen folgende Bereiche in Frage:

**kern** für die Systemmeldungen direkt aus dem Betriebssystemkern.

**auth** für Meldungen der Sicherheitsdienste (login u. ä.).

**authpriv** für vertrauliche Meldungen der internen Sicherheitsdienste.

**mail** für Meldungen des Mailsystems.

**news** für Meldungen des Nachrichtendienstes.

**uucp** für Meldungen des Fernkopierers.

**lpr** für Meldungen des Druckerdämons.

**cron** für Meldungen des crond.

**syslog** für Meldungen vom syslogd selbst.

**daemon** für Meldungen des unbekannten Dämons.

**user** für Meldungen aus normalen Anwenderprogrammen.

**local0-7** frei verwendbar für Nachrichten lokaler Bereiche.

Die Prioritäten (von der höchsten zur niedrigsten):

[5] Voraussetzung ist natürlich, daß der Dämon beim Systemstart aus einem der `rc*`–Scripts gestartet worden ist.

**emerg** für den letzten Spruch vor dem Absturz.

**alert** für alarmierende Nachrichten, die sofortiges Eingreifen erfordern.

**crit** für die Meldung kritischer Situationen, die gerade noch gut gegangen sind.

**err** für Fehlermeldungen aller Art.

**warning** für Warnungen vor Hunden, Käfern und anderen Gefahren.

**notice** zur Dokumentation besonders bemerkenswerter Situationen im Rahmen des normalen Betriebs.

**info** für die Protokollierung des normalen Betriebsgeschehens.

**debug** zur Mitteilung innerer Programmzustände bei der Fehlersuche.

**none** ist keine Priorität im eigentlichen Sinne, sondern dient dazu, alle Nachrichten einer Herkunftskategorie von einem Transportweg auszuschließen.

Die Nachrichten können auf vier verschiedene Weisen ausgeliefert werden:

1. können sie in eine Datei geschrieben werden. Dazu wird der Dateiname mit absolutem Pfad angegeben (mit einem Slash '/' beginnend).
2. können sie an den syslogd eines anderen Rechners im lokalen Netz weitergeleitet werden. Dazu wird der Zielrechnername, eingeleitet von einem '@'–Zeichen, angegeben.
3. können die Nachrichten an bestimmte Systembenutzer geschickt werden. Die Benutzernamen müssen dazu in einer durch Komma getrennten Liste aufgeführt werden. Die Nachricht wird den aufgelisteten Benutzern angezeigt, sofern diese im Augenblick des die Nachricht auslösenden Ereignisses eingeloggt sind.
4. können sie an alle eingeloggten User geschickt werden, indem ein Asterisk '*' anstelle des Usernamens angegeben wird.

Ein Eintrag in der `syslog.conf` besteht aus einer Zeile mit einem *Herkunft.Priorität*–Paar (durch einen Punkt voneinander getrennt) und einer Wegdefinition. In einem Herkunft.Priorität–Paar können auch mehrere Herkunftskategorien durch Komma getrennt aufgelistet werden. Außerdem können mehrere Herkunft.Priorität–Paare durch Semikolon getrennt in einer Zeile für einen Weg angegeben werden. Sowohl Herkunft als auch Priorität können durch einen Asterisk '*' als Wildcard ersetzt werden.
Kommentare können mit dem '#' eingeleitet werden.

**Beispiele:**

```
# Alle Kernelmessages, sowie die Nachrichten der Sicherheitsdienste
# mit der Priorität notice oder höher und alle Nachrichten mit
# der Priorität err und höher werden auf der Systemkonsole
# angezeigt.
# Die vertraulichen Nachrichten des Sicherheitsdienstes werden von
# dieser Anzeige ausgeschlossen.

kern.*;auth.notice;*.err;authpriv.none                  /dev/console

# Alle Nachrichten mit Priorität info und höher, außer den Nachrichten
# des Mailsystems und des privaten Sicherheitsdienstes, werden in der
# Datei /usr/adm/Logbuch mitgeschrieben.

*.info;mail.none;authpriv.none                          /usr/adm/Logbuch

# Die Nachrichten des privaten Sicherheitsdienstes kommen in eine
# sichere Datei.
```

```
authpriv.*                        /home/she/privat/.sicher/Nachrichten

# Alle Nachrichten mit Priorität emerg werden allen eingeloggten
# Usern sofort angezeigt, und sie werden an den syslogd eines anderen
# Rechners weitergeleitet.

*.emerg                                                              *
*.emerg                                              @soho.lunetix.de

# Die Nachrichten der Priorität alert und höher werden den
# verantwortlichen Systemadministratoren angezeigt, wenn sie gerade
# eingeloggt sind.

*.alert                                                      root,she

# Die kritischen Fehlermeldungen des News- und Mailsystems werden in
# der Datei /usr/adm/Kommunikationsstoerung gespeichert

uucp,mail,news.crit                    /usr/adm/Kommunikationsstoerung
```

# Kapitel 7

# UUCP und Smail

Als echtes Kind des Internets – was wäre LINUX ohne Netzwerk-Software? Ohne Netze wäre LINUX nie entwickelt worden; und schon gar nicht in solch atemberaubender Geschwindigkeit. Da ist es nicht verwunderlich, daß zu den ersten Applikationen, die nach dem GNU-Compiler auf LINUX portiert wurden, verschiedene Netzwerkprogramme für Mail, News und UUCP gehörten.

## 7.1 UUCP – Das Internet der Armen Leute

Wer seinen Rechner an ein Netzwerk anschließen will und nicht gerade das Glück hat, über einen lose herumliegenden Internet–Anschluß zu stolpern, muß sich gewöhnlich nach kostengünstigeren Alternativen umschauen. Die erste und beste Wahl ist in diesem Fall das Telefon. In Deutschland existieren mehrere, zum Teil bundesweite Computernetze, die ihre Mitglieder auf diese Weise miteinander verbinden.

Ein Teil der Netze, vor allem diejenigen, deren Mehrzahl aus Un⋆x–Systemen besteht, verwenden UUCP, um über das Telefonnetz miteinander zu kommunizieren. Der Name UUCP ist eine Abkürzung und steht für Unix–to–Unix–Copy. UUCP wurde 1978 von den Bell Laboratories entwickelt, um einen flexiblen Datenaustausch zwischen einzelnen Zweigniederlassungen zu ermöglichen. Dank seines einfachen Designs und seiner relativen Anspruchslosigkeit, was Installation und Wartung betrifft, hat sich UUCP rasch einen Spitzenplatz erobert, den es wohl noch einige Jahre behalten wird.

Im Laufe der Jahre hat es verschiedene Implementationen von UUCP gegeben. Dabei können grob zwei "Grundtypen" unterschieden werden, die sich im Format der Konfigurationsdateien stark unterscheiden. Der eine Typ, sogenannte "Version 2"–Implementationen, folgen einem einfacheren Konzept und stellen den älteren Zweig der Familie dar (Version 1 wurde nur innerhalb der Bell Laboratories verwendet). Der neuere Zweig stammt von einer Implementation aus 1983 von P. Honeyman, D.A. Novitz und B.E. Redman ab und heißt nach seinen Autoren auch HoneyDanBer oder HDB. Das HoneyDanBer–UUCP wurde unter der Bezeichnung *Basic Network Utilities* (BNU) ins AT&T Unix SVR3 integriert, so daß dieser Name ebenfalls in Verwendung ist.

### 7.1.1 UUCP–Anschluß — aber wie?

Ein zentrales Dogma im Usenet, dem weltweiten News–Netz, ist "*Get a feed and you're on it*," zu deutsch: Besorg' Dir einen Anschluß, und Du bist dabei. Das ist oft leichter gesagt als getan, denn Informationen über Anschlußmöglichkeiten erhält man oft nur, wenn man bereits Kontakt zu dieser Netzwelt aufgenommen hat.

Derzeit gibt es zwei größere gemeinnützige Vereine in Deutschland, die ihren Mitgliedern Netz–Dienste wie Mail und News über UUCP — und in zunehmendem Maße auch über TCP/IP — bieten. Dies sind einerseits *Individual Network e.V.* (kurz IN), andererseits der *Verein zur Förderung der privat betriebenen Datenkommunikation e.V.* (kurz Vz⋆ oder auch Sub–Netz, weil sich den vollen Namen im allgemeinen niemand merken kann). Beide Vereine kaufen Kontingente bei kommerziellen Service–Providern ein (s.u.), um auch die Versorgung mit internationaler Mail und News bieten zu können. Dem IN können nur Privatpersonen beitreten. Kontaktadressen für beide Vereine entnehmen Sie bitte dem Anhang.

Neben diesen gemeinnützigen Vereinen existiert die Möglichkeit, die Dienste von kommerziellen Providern in Anspruch zu nehmen, was für Privatpersonen allerdings kaum lohnt, da die Preise für eine Grundversorgung mit Mail und News mit rund 200 DM pro Monat ein Vielfaches der bei IN und Vz⋆ anfallenden Kosten betragen.

### 7.1.2 Was kann UUCP?

UUCP ermöglicht es, das öffentliche Telefonnetz zur Datenübertragung zu verwenden. Im Unterschied zu Terminal–Programmen, wie sie im vorigen Abschnitt vorgestellt wurden, arbeitet es aber nicht interaktiv, sondern erlaubt Ihnen, über verschiedene Dienstprogramme Aufträge zu erteilen, die zu festgelegten Zeiten selbsttätig ausgeführt werden. Um beispielsweise eine Datei von einem fremden Rechner anzufordern, können Sie folgendes Kommando ausführen:[1]

```
$ uucp -r flarp\!\~/archiv/Inhalt.Z Inhalt.Z
```

Dies erteilt UUCP den Auftrag, bei der nächsten Verbindung, die es mit dem System flarp aufbaut, aus einem allgemein zugänglichen Verzeichnis (unter Linux ist das nach dem neuen File–System–Standard

[1] Unter Umständen müssen Sie hierbei Ausrufungszeichen und Tilde durch einen vorangestellten Backslash vor Ihrer Shell schützen, da diese es als Sonderzeichen interpetieren könnte. Dies gilt sowohl für bash als auch für tcsh.

/var/spool/uucppublic) die Datei archiv/Inhalt.Z zu übertragen und im momentanen Verzeichnis in Inhalt.Z abzulegen. Das Flag -r fordert uucp auf, nicht sofort zu versuchen, die Verbindung aufzubauen.

Ebenso können Sie mit Hilfe von UUCP Kommandos auf einem fremden Rechner ausführen — sofern dieser es Ihnen erlaubt. Um beispielsweise die Datei handbuch.dvi auf dem Rechner flarp auszudrucken, müßten Sie das folgende eingeben:

```
$ uux -r flarp\!lpr -d \!handbuch.dvi
```

Das Ausrufezeichen vor handbuch.dvi weist uux darauf hin, daß es sich dabei nicht um ein einfaches Befehlsargument handelt, sondern um eine Eingabedatei, die zusätzlich zu dem Druckauftrag übertragen werden muß.

Wie Sie an den Beispielen sehen, verwendet UUCP das Ausrufezeichen, um den Namen eines Rechners von weiteren Parametern wie Datei– oder Befehlsnamen zu trennen. Es kann aber auch dazu verwendet werden, mehrere Rechnernamen voneinander zu trennen. Dieser Fall tritt beispielsweise auf, wenn Sie auf ein System zugreifen wollen, das Sie nicht direkt erreichen können. In diesem Fall müssen Sie auf die Vermittlung eines Systems zurückgreifen, das bereit ist, Ihren Auftrag an den Zielrechner weiterzuvermitteln. Läge in obigem Beispiel das Archiv nicht auf flarp, sondern auf tauris, einem Nachbarn von flarp, so müßte der Befehl stattdessen lauten:

```
uucp -r flarp\!tauris\!\~/archiv/Inhalt.Z Inhalt.Z
```

UUCP wird in diesem Beispiel den Auftrag, die Datei von tauris zu kopieren, an flarp weiterreichen. Zu geeigneter Zeit wird flarp diesen dann ausführen und die Datei bei sich zwischenspeichern. Bei der nächsten Verbindung mit Ihrem System wird diese dann endgültig an Sie übertragen.

Wie Sie sehen, gibt flarp!tauris hier den Pfad von Ihrem System zum Zielsystem an. Da das Ausrufezeichen im Computer–Englisch oft als *bang* bezeichnet wird, bezeichnet man dies auch als *bang-path*.

Neben diesen allgemeinen, allen Benutzern zugänglichen Befehlen kennt UUCP noch weitere, die hinter den Kulissen für das eigentliche Funktionieren eines UUCP–Knotens sorgen. Dies ist einerseits uucico, das Haupt– und Staats–Programm, das für die eigentliche Übermittlung der Daten zwischen zwei Systemen zuständig ist. Befehle wie uucp legen nämlich die zu übertragenden Daten und die Beschreibung ihres Auftrags im allgemeinen nur in einer Temporärdatei im sogenannten Spool–Verzeichnis ab.

Um beispielsweise den uux–Auftrag aus dem vorangehenden Beispiel an flarp zu übertragen, muß eine Verbindung zu diesem System hergestellt werden. Dies geschieht durch den Befehl

```
$ uucico -s flarp
```

uucico versucht nun, die (Telefon–) Verbindung mit flarp aufzubauen, und loggt sich dort ein. Dabei wird auf flarp ebenfalls ein uucico gestartet, mit dem uucico dann Daten austauscht. Im vorliegenden Falle sendet der lokale uucico die Datei handbuch.dvi an flarp, sowie den Auftrag, mit dieser Datei das Druckprogramm lpr aufzurufen.

Dieser Auftrag wird auf flarp anschließend von uuxqt ausgeführt, das nach Beendigung der Verbindung automatisch von uucico aufgerufen wird.

### 7.1.3 Wie sicher ist UUCP?

Diese Frage hat sich Ihnen vielleicht sofort aufgedrängt, als in den vorigen Abschnitten so freizügig die Rede davon war, daß mit UUCP Kommandos auf anderen Rechnern ausgeführt und Dateien nach Belieben hin– und herkopiert werden konnten.

Das ist natürlich etwas irreführend. All diese Sachen sind *im Prinzip* möglich; es hängt aber immer vom Verwalter des einzelnen Systems ab, was er einzelnen Gästen erlaubt, die über UUCP auf die Maschine zugreifen. Gewöhnlich erlaubt UUCP fremden Systemen nur, zwei Kommandos auszuführen: rmail und rnews, die traditionellen Befehle zum Einliefern von Mail und News über UUCP. Ebenso ist es fremden Systemen in der Regel nicht erlaubt, einfach Aufträge über Ihr System ausführen zu lassen, wie es beispielsweise im letzten

Beispiel oben geschah. Es liegt nun in Ihrem Belieben, einigen Systemen weitergehende Rechte einzuräumen, oder dies sogar weiter einzuschränken. Ferner hat UUCP eine Art "rechtsfreien Raum"; dabei handelt es sich um das Verzeichnis `/var/spool/uucppublic`. Die Voreinstellung von UUCP erlaubt fremden Systemen, in diesem Verzeichnis Daten abzulegen oder sie daraus herunterzuladen.

Im Allgemeinen wird man diese Voreinstellungen unverändert bestehen lassen. Bitte lesen Sie in den Handbüchern oder weiterführender Dokumentation nach, wenn Sie diese ändern wollen.

### 7.1.4 Taylor–UUCP

Die auf Un⋆x–Systemen am weitesten verbreitete "freie" UUCP–Implementation ist *Taylor UUCP*, so gennant nach ihrem Autor, Ian Taylor. Die derzeit aktuelle Version ist Version 1.4, die auch in den meisten LINUX–Paketen enthalten ist.

Eine Besonderheit von Taylor–UUCP ist, daß es sowohl die Konfigurationsformate des Version 2–UUCP als auch die der HoneyDanBer–Implementationen versteht. Diese Formate lassen sich bei der Kompilierung voreinstellen. Daneben wird noch ein drittes Format unterstützt, das sich vor den anderen durch eine wesentlich übersichtlichere Syntax auszeichnet. Im folgenden wird diese Taylor–spezifische Konfiguration behandelt, die auch von den in diversen LINUX–Paketen enthaltenen UUCP–Binaries verstanden wird.

Taylor–UUCP ist nicht nur in der Lage, Verbindungen über Telefonleitungen und Modems aufzubauen, sondern kann auch Transfers über ein TCP/IP–Netzwerk durchführen. Es würde hier allerdings zu weit führen, dies auch zu diskutieren, weshalb sich das hier beschriebene Beispiel auf den "traditionellen" Fall einer Wählverbindung beschränkt. Interessierte Leserinnen und Leser seien auf weiterführende Dokumentation verwiesen.

Besonders empfehlenswert ist in diesem Zusammenhang die umfangreiche Original–Dokumentation zu Taylor–UUCP, die im TEXinfo–Format vorliegt. Sie ist auf den meisten FTP–Sites[2] erhältlich, die GNU–Software anbieten. Einige LINUX–Distributionen dürften sie auch enthalten. Ebenfalls zu empfehlen ist das bei O'Reilly & Associates erschienene Buch *Managing UUCP and Usenet*, das zwar auch (noch) nicht die Taylor–Konfiguration beschreibt, aber ansonsten alles behandelt, was im Zusammenhang mit UUCP von Interesse ist.

### 7.1.5 Überblick über die Konfigurationsdateien

Bevor wir uns den einzelnen für die Konfiguration nötigen Schritten zuwenden, lohnt es sich, darüber nachzudenken, welche Informationen UUCP überhaupt benötigt, um mit einem anderen System zu kommunizieren und Daten auszutauschen.

Nehmen wir beispielsweise an, Sie wohnen in Berlin und haben sich bereits mit der Firma LunetIX in Verbindung gesetzt, die auf ihrer Maschine cicero ein Gateway für Mail und News betreibt. LunetIX hat Ihnen einen UUCP–Zugang eingerichtet. Als UUCP–Namen des Rechners haben Sie vorläufig isis vereinbart. Ihr Login–Name ist in diesem Fall Uisis, als Paßwort ist gniggl eingetragen, und die Telefonnummer ist 112 23 34. Damit hätten Sie zunächst die Information über den Zielrechner zusammen.

Aber das reicht natürlich nicht. Um überhaupt anrufen zu können, muß UUCP natürlich auch wissen, an welcher seriellen Schnittstelle sich das Modem überhaupt befindet und wie es angesprochen wird. Wir nehmen an, daß das Modem an COM2 hängt und ein Hayes–kompatibles Modem mit einer maximalen Übertragungsrate von 9600bps ist.

Mit diesen Informationen sollte UUCP nun in der Lage sein, eine Verbindung mit cicero herzustellen. Bei systematischer Betrachtung lassen sich die Informationen in mehrere Sinneinheiten zusammenfassen:

- Zunächst gibt es *systemspezifische* Informationen, wie die Telefonnummer, Login und Paßwort. Zu diesen Informationen kann unter Umständen hinzukommen, daß Sie überhaupt nur zu bestimmten Zeiten eine Verbindung mit dem anderen System aufbauen dürfen, um andere Benutzer nicht zu behindern.

[2] Der Begriff *Site* steht für einen vernetzten Rechner.

- Eine weitere Klasse von Informationen betrifft die Hardwareaspekte des angeschlossenen Übertragungsgerätes: an welcher Schnittstelle ist es zu finden, was ist seine maximale Übertragungsgeschwindigkeit, und von welchem Typ ist es?

- Die dritte Klasse von Informationen betrifft schließlich die logische Ansteuerung für Geräte eines bestimmten Typs: wie kann ein bestimmtes Gerät angesprochen werden, d.h. welche Befehle müssen ausgegeben werden, damit beispielsweise eine bestimmte Telefonnummer gewählt wird? Im allgemeinen wird es sich hierbei um modemspezifische Daten handeln; für TCP/IP–Links oder direkte serielle Leitungen beispielsweise können diese aber auch anderer Art sein.

Genau entlang dieser Einteilung ordnet Taylor–UUCP die benötigten Informationen nun auch drei verschiedenen Konfigurationsdateien zu: `sys` für die systemspezifischen Daten, `ports` für die Konfiguration der Schnittstellen und die Zuordnung der angeschlossenen Geräte, und schließlich `dial` für die Beschreibung der Modemtypen. Allgemeine Daten, wie beispielsweise der UUCP–Name des Rechners, sind in der Datei `config` abgelegt.

Alle diese Dateien, die im folgenden einzeln beschrieben werden, haben ein gemeinsames Format. Ein Eintrag besteht immer aus einem Schlüsselwort, gefolgt von Leerzeichen und/oder Tabulatorzeichen sowie einem oder mehreren Argumenten. Er kann über das Zeilenende fortgesetzt werden, wenn das letzte Zeichen der Zeile ein Backslash (\) ist. Kommentare werden durch ein Doppelkreuz (#) eingeleitet und erstrecken sich ebenfalls bis zum Zeilenende.

Die Konfigurationsdateien sind in einem gemeinsamen Verzeichnis versammelt. Abhängig von der Herkunft Ihres LINUX–Pakets sind die Binaries im allgemeinen so konfiguriert, daß sie die Konfigurationsdateien entweder in `/usr/lib/uucp`, `/usr/conf/uucp` oder `/usr/local/lib/uucp` suchen. Wenn Sie UUCP aus einer der diversen LINUX–Distributionen installieren, wird dieses Verzeichnis normalerweise angelegt und enthält oft auch Beispieldateien.

Wenn Sie Taylor–UUCP in guter alter Tradition sogar selbst kompilieren, können Sie dieses Verzeichnis auch selber festlegen. Sollten Sie LINUXmit einem lokalen Netz betreiben und das `/usr`–Verzeichnis verteilt nutzen, können Sie dem File–System–Standard entsprechend die UUCP–Konfigurationsdateien in das Verzeichnis `/etc` oder `/etc/uucp` installieren.

Im folgenden wird angenommen, daß die UUCP–Programme so konfiguriert wurden, daß sie die Konfigurationsdateien an ihrem traditionellen Platz im Verzeichnis `/usr/lib/uucp` suchen.

### 7.1.6 Log–Dateien

Da UUCP–Befehle oft im Hintergrund aufgerufen werden, senden sie Ausgaben wie Fehler– oder Diagnose–Meldungen nicht auf die Konsole, sondern in mehrere Log–Dateien, die unterhalb des Verzeichnisses `/var/spool/uucp` abgelegt werden.

Abhängig davon, wie UUCP konfiguriert wurde, werden die Log–Files unterschiedlich gehandhabt. Die meisten UUCP–Pakete unter LINUX sind dafür konfiguriert, HDB–kompatible Log–Dateien zu erzeugen. Diese Konvention kennt einerseits gewöhnliche Log–Dateien, in denen jede Transaktion vermerkt wird, und Debugging–Logs, in die zusätzliche Information abgelegt werden kann, wenn es vom Benutzer verlangt wird.

Die erste Kategorie von Log–Dateien ist unterhalb von `/var/spool/uucp/.Log` zu finden. Dieses Verzeichnis ist in drei weitere Unterverzeichnisse aufgeteilt; je eins für `uucp`, `uux` und `uucico`. Jedes dieser Unterverzeichnisse enthält ein separates Log für jedes fremde System. Ein Auftrag an uux, auf flarp das Handbuch auszudrucken, würde beispielsweise von uux in der Datei `uux/flarp` vermerkt. Beim nächsten Verbindungsaufbau mit flarp würde dann uucico einen weiteren Eintrag in der Datei `uucico/flarp` machen, daß der Auftrag auch tatsächlich übertragen wurde.

Die zweite Kategorie von Log–Dateien wird nur erzeugt, wenn Sie dies ausdrücklich von einem Programm verlangen. Debbugging–Logs werden gewöhnlich nur benötigt, um Fehlern nachzugehen. Diese Daten werden ausgegeben, indem Sie als Kommando–Parameter die Option -x mit einem Argument zwischen 1 und 11 angeben. Das Argument gibt an, wie detailliert die Information sein soll; höhere Werte bedeuten mehr Information. Diese Informationen werden in der Datei `audit.local` im Verzeichnis `/var/spool/uucp/.Admin` abgelegt. Es ist auch möglich, daß ein fremdes UUCP–System zu Beginn einer Verbindung Ihr System veranlaßt, Debug–Informationen aufzuzeichnen; diese wird dann in der Datei `audit` anstelle von `audit.local` abgelegt. Dies können Sie mit Taylor–UUCP dadurch erreichen, daß Sie zusätzlich die Option -X mit einem entsprechenden Wert zwischen 1 und 11 angeben.[3]

Ist Ihr UUCP so konfiguriert, daß er Taylor–spezifische Log–Dateien erzeugt, finden Sie diese in `/var/spool/uucp` als `Log` beziehungsweise `Debug`.

### 7.1.7 Die `config`–Datei

Die `config`–Datei kann zentral die in den Binärdateien gespeicherten Einstellungen verändern. Beispielsweise können hier Namen und Anordnung der Konfigurationsdateien und deren Formate neu definiert werden. Allerdings sind alle möglichen Parameter mit sinnvollen Voreinstellungen belegt.

Der wichtigste und meist einzige Eintrag, den Sie in dieser Datei vornehmen müssen, ist der UUCP–Name Ihres Rechers:

```
# /usr/lib/uucp/config
hostname        isis
```

Mehr brauchen Sie hier zunächst nicht einzutragen.

Falls Sie Ihr System auch für Anonymous UUCP öffnen wollen, werden in diese Datei auch die Zugangsrechte für unbekannte UUCP–Systeme eingetragen. Für Details konsultieren Sie bitte die Original–Dokumentation zu Taylor–UUCP.

### 7.1.8 Die `sys`–Datei

So einfach wie mit der `config`–Datei geht es natürlich nicht weiter. Bei den System–Daten wird es schon etwas kniffliger. Wie bereits erwähnt, werden diese in der Datei `sys` eingetragen. Das folgende Beispiel zeigt die passenden Einträge für die Verbindung zum Rechner cicero, wie sie am Anfang des Kapitels vorgestellt wurde.

```
# /usr/lib/uucp/sys
# Defaults
chat            ogin: \L ssword: \P
port            modem1
time            Any 2
protocol-parameter  g window      7
protocol-parameter  g packet-size 128

# 1. cicero.
# Kontakt: Sebastian; Tel: 1122344
system          cicero
phone           1122334
call-login      isis
call-password   gniggl

# 2. flarp
# Archiv-Site; Anonymous UUCP.
# Bem.: Miese Telefonleitung.
system          flarp
phone           7654321
call-login      uucp
call-password   nuucp
```

[3] Sie sollten vorher den Administrator des angerufenen Systems fragen. Möglicherweise ignoriert der fremde uucico diese Aufforderung auch schlicht.

```
protocol-parameter  g window      4
protocol-parameter  g packet-size 64
```

Diese sys–Datei beschreibt zwei Systeme, cicero und flarp. Jede Systembeschreibung beginnt in einer Zeile, die das Schlüsselwort system enthält. Alle Informationen, die zwischen zwei system–Zeilen auftauchen, beziehen sich ausschließlich auf die angegebene Site.

Anweisungen, die *vor* der ersten system–Zeile auftreten, werden als Default–Angaben für alle Systeme verwendet. Taucht dieselbe Angabe auch noch im System–Eintrag auf, so verdeckt diese Angabe den Default für das spezielle System. Beispielsweise haben die Protokoll–Parameter, die in der Beschreibung von flarp auftauchen, Vorrang vor den Default–Werten. (Was es mit diesen Protokoll–Parametern auf sich hat, wird später verraten).

### Telefonnummer und Login–Information

In der Beschreibung eines Systems geben die Einträge phone, call-login und call-password die beim Einwählen in dieses System zu verwendende Telefonnummer, den Loginnamen und das Paßwort an. Wichtig ist außerdem noch das Gerät, das zum Wählen verwendet werden soll. Das wird mit dem port–Befehl angegeben. Da in diesem Beispiel nur ein Modem benutzt wird, kann diese Angabe in den Defaults–Teil geschrieben werden.

Sollten Sie mehrere Modems verwenden, können Sie den Port auch für jedes System einzeln angeben. Praktischer ist dann allerdings, anstatt des Geräts nur die gewünschte Übertragungsgeschwindigkeit anzugeben; UUCP übernimmt es dann, anhand dieser Information ein passendes Gerät auszuwählen. Das hat den Vorteil, daß UUCP unter mehreren vorhandenen Geräten mit passender Übertragungsrate ein freies auswählen kann.

### Erlaubte Zeiten

Mit dem Befehl time können Sie kontrollieren, wann ein System angerufen werden darf. Der Wert Any erlaubt Verbindungen zu jeder Zeit. Sie können allerdings auch kompliziertere Zeitangaben machen, wie beispielsweise Mo-Fr0800-1700; dies bezeichnet den Zeitraum von 8 Uhr bis 17 Uhr an Werktagen. Man kann dies auch durch Wk0800-1700 abkürzen. Zeitangaben bestehen also aus einer Zeitspanne (in 24–Stunden–Schreibweise), und einer optionalen Angabe von Wochentagen (in den üblichen englischen Abkürzungen Mo, Tu, We, Th, Fr, Sa und Su).

Sie können auch mehrere Zeitangaben kombinieren, zum Beispiel Wk2305-0755,Sa,Su; dies schließt die Nachtzeit an Wochentagen sowie das gesamte Wochenende mit ein.

Der zweite Parameter ist optional und bezeichnet die Zeit, die uucico verstreichen läßt, bevor es einen erneuten Anruf bei dem Zielsystem zuläßt. Wenn uucico vorher aufgefordert wird, das System anzurufen, wird es die Arbeit verweigern und in der Log–Datei die Meldung "Retry time not reached" hinterlassen. Sie können uucico zwingen, das System trotzdem anzuwählen, indem Sie statt der Option -s die Option -S verwenden:

```
$ uucico -S cicero
$ _
```

UUCP merkt sich die Zeit des letzten Anrufs und den Status in privaten Dateien; Sie können diese Information mit dem folgenden Befehl anzeigen lassen:

```
$ uustat -m
cicero        12-06 19:25 Dial failed (3 tries, next after 12-06 19:28)
tauris        12-06 19:10 Conversation complete
$ _
```

### Das Chat–Skript

Sehr wichtig ist außerdem die chat–Information. Ein solches "chat script" (zu deutsch: Tratsch–Anweisung) erklärt UUCP, wie es sich mit dem fremden System zu unterhalten hat, damit es sich einloggen darf. Es setzt sich aus mehreren Teilen zusammen, die abwechselnd beschreiben, auf welche Ausgabe des fremden Systems UUCP warten muß, und was es darauf zu antworten hat. Das oben angegebene Skript ist sehr simpel, es beschreibt im Grunde genau das, was auch Sie immer tun, wenn Sie sich bei Ihrem System anmelden: Am Login–Prompt geben Sie Ihre Benutzerkennung ein, warten auf den Paßwort–Prompt und geben Ihr Paßwort ein. In obigem Beispiel wartet UUCP auf die Aufforderung ogin: und schickt daraufhin die Benutzerkennung. Das Chat–Skript gibt diese Benutzerkennung nicht direkt an, sondern verwendet den Platzhalter \L, der bei der Ausführung von UUCP durch den Wert von der call–login–Angabe ersetzt wird. Anschließend wartet es auf die Aufforderung ssword:, und schickt daraufhin das Paßwort (symbolisiert durch \P).

Sie mögen sich jetzt vielleicht fragen, warum im Chat–Skript die Eingabeaufforderungen so verstümmelt auftauchen. Der Grund ist, daß das Skript unabhängig davon funktionieren soll, ob das fremde System Login: oder login: ausgibt. Beim Paßwort–Prompt hat man den zweiten Buchstaben aus ästhetischen Gründen dann gleich auch noch weggelassen.

Chat–Skripts können natürlich beliebig kompliziert werden, beispielsweise, wenn das andere System gelegentlich dazu gebracht werden muß, den Login–Prompt erneut auszugeben. Dies können Sie mit einer Art Verzweigung erreichen, die immer dann ausgeführt wird, wenn UUCP eine Weile vergeblich auf eine Äußerung des fremden Systems warten mußte. Eine Allround–Version des Chat–Skripts, das in den meisten Situationen funktionieren sollte, ist folgende:

```
chat        "" \r\n\c ogin:-BREAK-ogin: \L ssword: \P
```

Die erwähnte Verzweigung sehen Sie im Mittelteil: wartet UUCP vergeblich auf ogin:, so schickt es einen BREAK und wartet erneut auf ogin:. Erst wenn es diesen nach einer Weile immer noch nicht gesehen hat, gibt es dann auf.

Die ersten beiden Teile des Skripts sind für Systeme gedacht, die erst auf ein Zeichen des Anrufers warten, bevor sie ihren Prompt ausgeben; bei allen anderen Systemen sollte dies unschädlich sein. Die beiden Anführungszeichen bewirken, daß UUCP zunächst auf gar nichts wartet, sondern sofort das zweite Feld sendet: ein Carriage–Return (ASCII 13) und ein Linefeed (ASCII 10). Das Zeichen \c bewirkt, daß kein weiteres Linefeed ausgegeben wird, wie es normalerweise geschieht.

Zuletzt sei noch erwähnt, daß manche Systeme einen Moment Bedenkzeit benötigen, bevor man ihnen die Benutzerkennung oder das Paßwort sendet (z.B. SCO Unix). In solchem Falle sollten Sie eine kleine Verzögerung einbauen, indem Sie vor die Angaben \L und \P ein \d angeben.

### Protokoll–Parameter

Um sich an verschiedene Gegebenheiten anpassen zu können, bietet UUCP eine Auswahl von Protokollen zur Übertragung von Daten an. Traditionell werden diese Protokolle durch einen einzelnen Buchstaben benannt. Das ist zwar nicht sehr einprägsam, aber so viele sind's ja auch gar nicht.

Das wohl wichtigste dieser Protokolle ist das *g*–Protokoll. Es ist das älteste und am weitesten verbreitete UUCP–Protokoll, das eigentlich jede UUCP–Implementation akzentfrei sprechen sollte. Bei g handelt es sich um ein paketorientiertes Protokoll, d.h. Daten werden nicht am Stück, sondern in einzelnen Paketen übertragen, die auf Fehlerfreiheit überprüft und im Bedarfsfalle erneut gesendet werden. Diese Eigenschaft hat zur Folge, daß g in hohem Maße für die Datenübertragung über Telefonleitungen geeignet ist.

Sie könnten nun an dieser Stelle einwenden, daß Ihr Modem aber in der Lage ist, Fehler zu erkennen und zu korrigieren, ist der Aufwand denn nötig? Ja, und zwar deshalb, weil das Protokoll des Modems zwar in der Lage ist, die meisten in der Telefonleitung auftretenden Fehler zu korrigieren, aber überhaupt keine Kontrolle darüber hat, was auf der Strecke zwischen Modem und Rechner passiert.

Für jedes fehlerfrei empfangene Paket sendet g eine Empfangsbestätigung (ACK) an den Absender; bleibt eine solche aus oder erhält der Sender stattdessen eine Fehlermeldung (NAK), geht er davon aus, daß das

Paket bei der Übertragung verlorenging oder beschädigt wurde, und sendet es erneut. Wenn nun g für jedes einzelne Paket auf ein ACK warten müßte, wäre dies äußerst langsam, da ja beide Seiten die meiste Zeit nur Däumchen drehen würden. Deshalb kennt g ein "Fenster" (engl. *sliding window*), das es ihm erlaubt, eine Anzahl von Päckchen zu schicken, bevor es auf das Eintreffen des ersten ACK warten muß. Damit kann erreicht werden, daß die Leitung zu nahezu 100% ausgelastet wird, denn oft hat das empfangende System die Quittung für das erste Päckchen bereits geschickt, bevor der Sender das letzte im Fenster liegende Päckchen auf die Leitung geben konnte.

Ganz so rosig, wie die DFÜ–Welt zunächst aussieht, ist sie aber nicht: Wenn ein Päckchen im Fenster fehlerhaft ist, müssen alle Pakete des Fensters neu übertragen werden, egal, ob sie korrekt empfangen worden sind oder nicht. Die Wahrscheinlichkeit dafür, daß ein komplettes Fenster neu übertragen werden muß, hängt auch von der Größe des Fensters ab. Wenn immer die maximale Päckchen– und Fenster–Größe verwendet wird, kann es bei schlechten Verbindungen zu einem enormen Überhang doppelt übertragener Daten kommen.

Aus diesem Grund hatten früher die meisten UUCP–Varianten feste Werte hierfür vorgegeben, nämlich eine Paketgröße von 64 Bytes und ein Fenster von 3 Paketen. Die Taylor–Konfiguration erlaubt Ihnen hingegen, diese Werte an Ihre Gegebenheiten anzupassen. Hierzu dient das Schlüsselwort protocol-parameter. Das erste Argument ist immer der Protokoll–Name (z.B. g), gefolgt von dem zu setzenden Parameter. Welche Parameter überhaupt gesetzt werden können, und welche Werte zulässig sind, hängt natürlich von dem Protokoll ab. Dies ist sehr ausführlich in der offiziellen Dokumentation zu Taylor–UUCP beschrieben. Für das Beispiel sind nur das g–Protokoll und dessen Paket– und Fenster–Größe interessant.

Die Paketgröße kann über den Parameter packet-size eingestellt werden. Erlaubt sind alle Zweierpotenzen zwischen 32 und 4096; die Voreinstellung liegt bei 64. Das Fenster wird über den Parameter window gesetzt und darf zwischen 1 und 7 liegen; Default ist 7.

Neben dem g–Protokoll versteht Taylor–UUCP unter anderem noch die Protokolle e und f, die zur Übertragung über TCP/IP–Verbindungen verwendet werden, G, das eine System V–spezifische Variante von g ist, und das neue i–Protokoll. Dieses Protokoll kann über eine Modemleitung gleichzeitig Pakete empfangen und senden: Protokolle wie g kennen zu einem bestimmten Zeitpunkt jeweils nur einen Sender und Empfänger (Master und Slave); ersterer schickt Datenpäckchen, und letzterer sendet nur die Bestätigungen (ACK und NAK) zurück, d.h. in der Richtung vom Slave zum Master ist die Leitung nur schlecht ausgelastet. Diesen Mißstand behebt i, indem es die Unterscheidung zwischen Master und Slave aufhebt, und Datenpakete und Bestätigungen kombiniert.

### 7.1.9 Die `port`–Datei

Damit UUCP Ihr Modem überhaupt ansprechen kann, müssen Sie die `port`–Datei entsprechend konfigurieren. Das ist schnell getan:

```
# /usr/lib/uucp/port
# Modem with V42bis compression
port            modem1
type            modem
device          /dev/cua1
speed           9600
dialer          hayes
```

Diese Datei enthält nur einen einzigen Eintrag, der das Modem am Port COM2 beschreibt. Ein solcher Eintrag wird durch das Schlüsselwort port eingeleitet, das dem Port einen eindeutigen Namen zuweist. Dies ist derselbe Name, wie Sie ihn in der Datei `sys` mit dem port–Befehl angegeben haben.

Die nächste Zeile benennt den Typ der Schnittstelle; anstelle von modem sind hier unter anderem Typen wie direct für eine Direktleitung, oder tcp für eine TCP/IP–basierte Verbindung zulässig.

Der Befehl device bezeichnet die Gerätedatei, durch die das Modem angesprochen werden soll. Unter Un⋆x erfolgt der Zugriff auf Peripherie–Geräte über spezielle "Dateien". LINUX verwendet für serielle Schnittstellen die Dateien `/dev/cua*` und `/dev/ttyS*`. Erstere werden zum aktiven Ansprechen verwendet, wie zur Anwahl eines anderen Rechners oder zum Konfigurieren des Modems; letzteres wird im allgemeinen verwendet, um das Einloggen über die serielle Schnittstelle zu ermöglichen.

Dabei entsprechen die Gerätedateien `cua0` bis `cua3` den Standard–Schnittstellen COM1 bis COM4.[4] Da Ihr Modem an COM2 angeschlossen ist, tragen Sie hier `/dev/cua1` ein.

Bitte beachten Sie, daß Sie hier auf jeden Fall den tatsächlichen Namen der Gerätedatei eintragen müssen. UUCP ignoriert symbolische Links, wie beispielsweise `/dev/modem`.

Der nächste Eintrag, `speed`, legt die Übertragungsgeschwindigkeit fest, die das UUCP zur Kommunikation mit dem Modem benutzen soll. Da ein Modem bei 2400 Baud bei eingeschalteter V42bis–Kompression einen maximalen Datendurchsatz von 9600 Bit/s erreichen kann, tragen Sie hier 9600 ein. Damit sendet der Rechner zwar oft schneller, als das Modem die Daten auf die Leitung schicken kann; das ist aber nicht schlimm, wenn auf der seriellen Schnittstelle Hardware–Handshake eingeschaltet wurde. So kann das Modem den Datenfluß vom Rechner regulieren. Schimmer wäre, wenn Sie diesen Wert zu niedrig wählen, weil dann der hohe Durchsatz auf der Modem–zu–Modem–Verbindung durch eine langsame Modem–zu–Rechner–Verbindung wieder zunichte gemacht wird.

Der letzte Eintrag teilt UUCP mit, um welchen Typ von Modem es sich bei dem angeschlossenen Gerät handelt. In diesem Fall ist dies ein Hayes–kompatibles Modem. Der Name, den Sie dabei vergeben, tut wenig zur Sache, da Sie die Beschreibung des Modems selber erstellen müssen.

### 7.1.10 Die `dial`–Datei

Dies ist die letzte wichtige Datei, die UUCP beschreibt, wie es das Modem ansprechen und veranlassen kann, die gewünschte Telefonnummer zu wählen. Die Konfiguration für das Hayes–Modem aus unserem Beispiel könnte beispielsweise folgendermaßen aussehen:

```
# /usr/lib/uucp/dial
# Dialer information for Hayes-compatible modem
dialer          hayes
chat            "" ATZ OK \dATH1E0Q0 OK \dATDP\D CONNECT
chat-fail       ERROR
chat-fail       BUSY
chat-fail       NO\sCARRIER
chat-fail       NO\sDIALTONE
dtr-toggle      true
```

Diese Datei enthält wiederum nur einen einzigen Eintrag, der den Modem–Typ hayes beschreibt. Im wesentlichen wird hier definiert, in welchen Bahnen die Unterhaltung zwischen UUCP und dem Modem verlaufen soll. Wieder wird dies in Form eines Chat–Skripts angegeben, wie Sie ihm bereits im Zusammenhang mit dem Einloggen im Abschnitt 7.1.8 begegnet sind.

Der wichtige Teil des Chat–Skripts wird durch den Befehl chat angegeben. In unserem Fall wartet UUCP zunächst auf keinerlei Reaktion des Modems (d.h. den leeren String), sondern schickt ihm zuerst die Sequenz ATZ. Dies ist die Aufforderung, sich zu initialisieren und interne Variablen mit vordefinierten Werten zu besetzen. Diese Sequenz sollte auf jeden Fall geschickt werden, damit sich das Modem in einem definierten Anfangszustand befindet.

Als Antwort auf dieses Kommando wartet UUCP auf den String OK. Erhält es diesen, sendet es den nächsten String, der mit einer kurzen Verzögerung (`\d`) beginnt, und anschließend den Befehl ATH1E0Q0 absetzt. Dieser hebt "den Hörer ab" (natürlich hat ein Modem keinen Hörer!), verlangt Klartext–Anwortcodes des Modems und schaltet das lokale Modem–Echo ab. Wieder erwartet UUCP als Antwort hierauf OK und fordert das Modem anschließend auf, die gewünschte Telefonnummer zu wählen. Dies geschieht durch den Befehl ATDP, gefolgt von der eigentlichen Nummer. Anstelle von \D setzt UUCP die der `sys`–Datei entnommene

[4] Namen wie COM1 sind nur Schall und Rauch. Unter LINUX haben diese überhaupt keine Bedeutung; sie werden nur häufig verwendet, weil sie sich in den Hardware–Beschreibungen für PCs eingebürgert haben. Bei Fragen wenden Sie sich bitte an Bill Gates.

Nummer ein. Zuletzt wartet es auf die Meldung CONNECT, mit der das Modem den erfolgreichen Verbindungsaufbau mit der Gegenstelle anzeigt.

Sollte UUCP innerhalb einer gewissen Zeit nicht den String finden, den das Chat–Skript angibt, so bricht es den Anwahlvorgang mit einer Fehlermeldung ab. Dieser Fehler wird in der Log–Datei als "Timed out in modem chat" vermerkt. Das kann beispielsweise passieren, wenn das Modem nicht eingeschaltet ist, aber auch, wenn beispielsweise die Gegenstelle besetzt ist. In diesem Fall liefert das Modem allerdings eine Fehlermeldung zurück, die im allgemeinen aufschlußreicher ist als ein bloßes "Timed out". Auf diese Fehlermeldungen können Sie UUCP trainieren, indem sie diese mit dem chat–fail Kommando angeben. Wann immer UUCP einen dieser chat–fail Strings erkennt, bricht es den Anwählvorgang ab und vermerkt die erhaltene Fehlermeldung in der Log–Datei.

Die in unserem Beispiel angegebenen Fehlermeldungen werden von einem Hayes–kompatiblen Modem zurückgegeben, wenn etwa der Anschluß besetzt ist (BUSY) oder nach dem Abheben kein Freizeichen zu hören ist (NO DIALTONE). Das Zeichen \s in diesem String markiert ein Leerzeichen.

Der letzte Befehl in diesem Eintrag beginnt mit dem Schlüsselwort dtr-toggle. Dieser Befehl veranlaßt UUCP, die DTR–Leitung (Data Terminal Ready) der seriellen Schnittstelle auf LOW zu ziehen, bevor es das Modem anspricht. Dies ist für manche Modems notwendig, die an dem Zustand der DTR ablesen, ob die Rechnerseite sie ansprechen will.

### 7.1.11 Testen der Konfiguration

Nachdem Sie UUCP nun installiert haben, sollten Sie das System testen, indem Sie eine Verbindung mit cicero aufzubauen versuchen. Sie führen dazu folgende Kommandos aus:

```
$ uucico -s cicero -x 2
$ tail -f /var/spool/uucp/.Admin/audit.local
uucico cicero - (1993-12-06 21:55:02.18 4172) Calling system cicero (port cua1)
uucico cicero - (1993-12-06 21:55:37.42 4172) Login successful
uucico cicero - (1993-12-06 21:55:38.02 4172) Handshake successful (protocol 'g
' packet size 128 window 7)
uucico cicero - (1993-12-06 22:01:35.45 4172) Protocol 'g' packets: sent 35, re
sent 4, received 943
uucico cicero - (1993-12-06 22:01:40.58 4172) Call complete (367 seconds)
```

Der zweite Befehl erlaubt Ihnen, den Aufbau der Verbindung zu verfolgen.

### 7.1.12 Regelmäßige Verbindungen

Wenn Sie Ihr System einmal soweit konfiguriert haben, daß Mail und News einwandfrei funktionieren, können Sie dazu übergehen, UUCP unbeaufsichtigt laufen zu lassen. Dazu können Sie beispielsweise für den Benutzer uucp einen cron–Auftrag anlegen, der zu festgelegten Zeiten uucico aufruft oder alte Log–Dateien wegwirft. Zum Beispiel könnten Sie folgendes in eine Datei crontab.uucp eintragen:

```
PATH=/usr/lib/uucp
0     22    *     *     *     uucico -s cicero
0     6     *     *     *     uucico -s cicero
```

Dies würde jeweils um 6 Uhr und 22 Uhr uucico aufrufen, um Daten mit cicero auszutauschen. Sie können den cron–Auftrag anschließend als Superuser mit folgendem Befehl installieren:

```
# crontab -u uucp -r crontab.uucp
```

Diese Lösung ist allerdings noch nicht ganz befriedigend, denn es ist möglich, daß bei cicero beispielsweise kurzzeitig besetzt ist; Ihr Anruf um 22 Uhr würde beispielsweise fehlschlagen, obwohl er vielleicht eine Minute später Erfolg gehabt hätte. Die Lösung ist hier, uucico mehrfach im Abstand von wenigen Minuten aufzurufen. Damit dies korrekt funktioniert, müssen Sie aber in der sys–Datei bei cicero noch eine Änderung vornehmen. Entweder für das in Frage stehende System oder im Defaults–Teil der sys–Datei tragen Sie die Option

```
success-wait            900
```

ein. Dies verhindert, daß uucico nach einer erfolgreichen Verbindung sofort wieder anruft. Das obige Beispiel setzt die Wartezeit auf 15 Minuten (gleich 900 Sekunden). Jetzt können Sie den cron–Job abändern, so daß uucico mehrere Male in Folge aufgerufen wird, aber höchstens einmal die Verbindung aufbaut:

```
PATH=/usr/lib/uucp
0,2,4,6     22    *     *     *     uucico -s cicero
0,2,4,6     6     *     *     *     uucico -s cicero
```

## 7.2 Elektronische Post mit smail

Einer der am weitesten verbreiteten Dienste, die durch die Vernetzung von Rechnern möglich werden, ist die elektronische Post, meist *Email* genannt. Email erlaubt es Benutzern auf unterschiedlichen Rechnern, auch auf entgegengesetzten Seiten des Erdballs, miteinander zu kommunizieren, und zwar erheblich schneller, als es mit gewöhnlicher Post jemals möglich wäre. Im Internet benötigt eine Botschaft oft nur wenige Minuten von Europa in die USA. In reinen UUCP–Netzen ist ein elektronischer Brief natürlich wesentlich länger unterwegs, da er die meiste Zeit seiner Reise auf den Spool–Platten diverser Rechner verbringt. Trotzdem kann sich die Beförderungsgeschwindigkeit immer noch mit der der Deutschen Bundespost messen.

Zur Beförderung von Email sind natürlich Standards vonnöten, damit Maschinen, die Botschaften miteinander austauschen, einander überhaupt verstehen. Ein solcher Standard ist RFC 822, der das Format von Mails im Internet regelt. Da viele UUCP–Netze eng an das Internet angebunden sind, hat sich RFC 822 auch in diesen weitgehend durchgesetzt. Wie es mit Standards so geht, hat natürlich jedes zweite Netz seinen eigenen. Wir werden uns hier jedoch nur mit RFC 822 beschäftigen.

### 7.2.1 Wie sieht eine Mail denn nun aus?

Eine Mail ist im wesentlichen eine Datei, die den Text Ihres Briefes enthält. Ein Teil dieser Datei besteht aus administrativen Daten, wie den Adressen von Absender und Empfängern, die, ganz wie im richtigen Leben, im Kopf des Briefes untergebracht sind. Wir werden diesen Teil im folgenden allgemein mit *Kopf* bezeichnen, oder mit dem englischen Ausdruck *message header*; den eigentlichen Text der Botschaft nennen wir *Rumpf* oder *message body*.

Ein elektronischer Brief könnte beispielsweise so aussehen:

```
From fluxus.in-berlin.de!susanne Thu Jul 15 09:16:21 1994 remote from fluxus
Return-Path: <fluxus.in-berlin.de!cicero.in-berlin.de!susanne>
Received: from cicero.in-berlin.de by isis.in-berlin.de with uucp
        (Smail3.1.28.1 #6) id m0oGNY2-0000H9B; Thu, 15 Jul 94 10:16 MET DST
Received: by cicero.in-berlin.de from fluxus.in-berlin.de
        msg-id m0zF9AR.000G2Za; Thu, 15 Jul 94 09:00 MET DST
Received: by fluxus.in-berlin.de
        id AA0043n; Tue, 15 Jul 94 08:53:33 CET
Message-Id: <9407130840.AA02871@fluxus.in-berlin.de>
Date: Tue, 15 Jul 94 08:53:32 MESZ
From: Susanne Bois <susanne@fluxus.in-berlin.de>
To: karla@isis.in-berlin.de (Karla Kosolowski)
Subject: Linux 3.1 draussen

Hi, Karla,

seit gestern gibt's Linux 3.1 auch auf meinem Rechner. Ich kann Dir
mal ein Band bespielen und es rueberschieben, wenn Du willst.

Susanne
```

Der Kopf des Briefes — der gesamte Bereich bis zur ersten Leerzeile — enthält die administrativen Informationen, die in einzelne Felder aufgeteilt sind. Jedes Feld beginnt mit einem Namen, gefolgt von einem Doppelpunkt, und dem eigentlichen Inhalt des Feldes. Dieses Feld kann auch auf der nächsten Zeile fortgesetzt werden, wenn die Zeile mit einem Einschub (TAB) beginnt. Einige der Informationen sind eher technischer Art, wie die Received: Felder, andere sind durchaus auch für die Empfängerin von Interesse, wie die Zeile Date:, die das Datum der Erstellung enthält, und Subject:, in der die Absenderin dem Brief eine Art Betreff–Zeile voranstellt. Die meisten dieser Felder werden von der Mail–Software automagisch ausgefüllt, beispielsweise Datum und Absenderadresse.

### 7.2.2 Adressen, Adressen, Adressen

Wenn Sie einem Menschen einen Brief oder eine Postkarte schreiben, werden Sie diese natürlich mit dessen Anschrift versehen, bestehend aus Namen, Straße und Hausnummer sowie Wohnort. Auf ähnliche Weise müssen Sie natürlich auch der Transport–Software mitteilen, wie der Empfänger Ihres elektronischen Briefs zu erreichen ist. Verschiedene Netze haben dafür natürlich auch unterschiedliche Adressierungsarten und –formate. Uns werden aber nur zwei Formate interessieren: der traditionelle UUCP *bang path*, oder !–Pfad, und das RFC 822–Format.

Beiden Adreßformaten ist gemeinsam, daß sie aus je einer Personenbezeichnung und einer Rechnerbezeichnung bestehen; d. h. ein Teil der Adresse beschreibt den Empfänger, meist durch dessen Benutzerkennung, während der andere den Zielrechner beschreibt, auf dem die Person überhaupt zu erreichen ist. Im obigen Beispiel bestand karla@isis.in-berlin.de aus den Teilen isis.in-berlin.de, dem vollständigen Namen des Rechners, und karla, dem Login–Namen von Karla. Beide werden durch das "@"–Symbol voneinander getrennt. Diese Adressen–Schreibweise entspricht den Vorschriften von RFC 822.

Dem !–Pfad sind wir bereits im vorigen Abschnitt bei der Einführung in die Welt von UUCP begegnet, wo fluxus!tauris eine Abfolge von Rechnern bezeichnete, über die ein bestimmter Auftrag übermittelt werden sollte. Diese Schreibweise wurde früher in UUCP–Netzen auch häufig zur Adressierung von Mail benutzt und findet heute noch innerhalb der Transport–Software Verwendung. Dabei wird, getreu dem Motto "Der Weg ist das Ziel", das Zielsystem nicht durch einen eindeutigen Namen angegeben, sondern durch eine Auflistung der Systeme, über die es erreichbar ist. Um beispielsweise einen Brief an Jens auf dessen Maschine tauris zu senden, müßte Karla die Adresse fluxus!tauris!jens verwenden. Eine solche Adresse teilt der Post–Software mit, daß sie die Nachricht an fluxus übermitteln soll, das diese an tauris weiterreichen wird, wo sie an den Benutzer jens ausgeliefert werden wird. Diese Form der Adressierung wird jedoch nur noch äußerst selten verwendet, weswegen wir uns an dieser Stelle nicht weiter mit ihr beschäftigen werden.

Sie sehen in diesen Beispielen auch zwei Arten, ein System zu benennen: einerseits mit seinem einfachen "Vornamen", wie isis, und mit seinem vollen Namen, isis.in-berlin.de. Diesen zusätzlichen Teil des Namens nennt man die *Domain*, d. h. den Bereich, dem der Rechner zugehörig ist. Domains wurden eingeführt, als dank der stark ansteigenden Zahl von vernetzten Rechnern sinnvolle Namen knapp zu werden begannen, und es auch immer schwieriger wurde, den Überblick über die Netzwerk–Topologie zu behalten. Die Idee von Domains ist, größere Gruppen von Systemen, die geographisch oder organisatorisch eng verbunden sind, in "Meta–Systeme" zusammenzufassen und mit einem gemeinsamen Gruppennamen zu belegen. Dies erleichtert unter anderem das Mail–Routing ganz ungemein, spielt aber auch in anderen Bereichen eine Rolle.

### 7.2.3 Taler, Taler, Du mußt Wandern

Um in einem Netzverbund eine Botschaft von einem System zu einem anderen schicken zu können, muß die Transport–Software wissen, auf welchem Weg sie diese befördern kann. Die Aufgabe, einen korrekten und möglichst optimalen Pfad zu finden, wird als *Routing* bezeichnet. In Lokalen Netzen (LANs) und dem Internet ist dies im allgemeinen recht einfach, da der Zielrechner meist direkt angesprochen werden kann.

In UUCP–Netzen ist das schon kniffliger, da die Botschaft dabei durch die Hände mehrerer Systeme gereicht werden muß. Die einfachste Lösung ist der !–Pfad, der dem Benutzer die knifflige Aufgabe überläßt, einen zuverlässigen Pfad vom lokalen System zum Zielsystem zu finden. Diese Lösung ist natürlich unbefriedigend; es wäre angenehmer, wenn das System diese Pfade selbständig generieren würde, wenn man ihm nur den Zielrechner angibt.

Dies kann beispielsweise durch die Verwendung einer sogenannten pathalias–Datei erreicht werden. In einer solchen Datei werden Rechnern oder ganzen Domains !–Pfade zugeordnet, über die sie erreichbar sind. Wenn sie Ihren gesamten Verkehr über nur ein UUCP–System abwickeln, werden sie eine solche Datei allerdings nie benötigen, da es noch eine dritte Möglichkeit gibt – nämlich die Arbeit auf andere abzuwälzen.

Bei dieser Methode stellt sich Ihre Post–Software (absichtlich) dumm, und überläßt es dem intelligenteren System, aus den Adressen der Zielrechner vernünftige Pfade zu erzeugen. Der gängige Begriff hierfür ist *smart–host routing*.

### 7.2.4 Post–Software unter LINUX

In der Un$\star$x–Welt gibt es verschiedene Programme zur Erstellung und Beförderung von Email. Eins der bekanntesten Benutzer–Programme ist elm, das Ihnen ein Cursor–gesteuertes Menü zur Verfügung stellt, von dem aus Sie Botschaften versenden und die Nachrichten in Ihrem Briefkasten lesen können. elm wird in einem der folgenden Abschnitte näher beschrieben. Daneben gibt es noch viele weitere Programme, die alle ungefähr dieselbe Funktionalität bieten, jedoch in unterschiedlichen Graden der Benutzerfreundlichkeit.

Natürlich ist die Benutzer–Software nur der eine Teil des Post–Systems; der andere Teil ist die Transport–Software. Auch hier gibt es verschiedene Pakete, wie beispielsweise sendmail, das ursprünglich für das Unix der Universität in Berkeley geschrieben wurde, und auf den meisten Un⋆x–Plattformen verwendet wird. Für LINUX existiert eine Portierung von Rich Braun. Da aber sendmail für Anfänger schwer zu installieren ist, bevorzugen viele eine andere Post–Software: smail. Dieses Paket wurde von Landon Curt Noll und Robert S. Karr geschrieben und stellt ausreichende Funktionalität für kleinere bis mittlere Systeme zur Verfügung. Im folgenden Abschnitt werden die notwendigen Schritte für die Installation von smail auf einem UUCP–System dargelegt.

### 7.2.5 Installation von smail

Abhängig von der Linux–Distribution, die Sie verwenden, finden Sie smail samt einiger zugehöriger Programme entweder in `/usr/bin` oder `/usr/local/bin`. Außerdem muß eine Kopie von smail als `/usr/lib/sendmail` vorhanden sein.

Ferner benutzt smail eine oder mehrere Konfigurationsdateien, die allesamt in `/usr/lib/smail` beziehungsweise `/usr/local/lib/smail` abgelegt werden. Falls Sie sich nicht sicher sind, in welchem Verzeichnis diese Dateien abgelegt werden, können Sie das Verzeichnis mit folgendem Befehl erfragen:

```
smail -bP smail_lib_dir
```

Wir werden im folgenden die Konfiguration für ein System besprechen, das mit nur einem weiteren System per UUCP kommuniziert. Natürlich bleibt unser Beispiel weit hinter den Möglichkeiten von smail zurück; mehr darüber können Sie in den Manual–Seiten zu smail erfahren, sowie im Networking Guide.

#### Die config–Datei

Die wichtigste Konfigurations–Datei ist config. Für Karlas System könnte die Datei beispielsweise folgendermaßen aussehen:

```
#
# config file for isis.in-berlin.de
#
hostnames=isis.in-berlin.de
visible_name=isis.in-berlin.de
uucp_name=isis.in-berlin.de
#
smart_path=fluxus
smart_transport=uux
#
error_copy_postmaster
```

Zeilen, die mit einem Doppelkreuz beginnen, sind Kommentare und werden ignoriert. Alle Einträge in der Datei besetzen verschiedene Konfigurations–Variablen. Die ersten drei Variablen setzen den Namen des Systems für verschiedene Operationen fest; in diesem Fall ist dies immer isis.in-berlin.de. Die genauen Bedeutungen dieser Variablen können Sie der Dokumentation entnehmen. Im Falle eines einzelnen Computers genügt es, wenn Sie alle drei Variablen mit dem offiziellen Namen Ihres Systems besetzen.

Die Variablen smart_path und smart_transport beziehen sich auf das oben vorgestellte smart–host routing. Erstere enthält den UUCP–Namen des Rechners, über den Sie alle ausgehenden Nachrichten ausliefern. In Karlas Fall geht alle Post über fluxus. Die zweite Variable teilt smail den Transport mit, über den dies erfolgen soll; ein Wert von uux bezeichnet die Auslieferung über UUCP an das Programm rmail auf dem nächsten Rechner, also fluxus. Dies ist die Standard–Methode, um in einem UUCP–Netz Post zu transportieren.

Die letzte Variable, `error_copy_postmaster`, ist eine Boolesche Variable. Wird sie gesetzt,[5] wie in unserer Beispieldatei, wird der Benutzer postmaster von jedem Fehler bei der Auslieferung einer Botschaft benachrichtigt und erhält eine Kopie derselben.

Neben config kennt smail noch einige weitere Konfigurations–Dateien, die aber zum Funktionieren einer einfachen Installation nicht nötig sind: routers, directors und transports, die Details des Routing, der lokalen Auslieferung sowie des Transports zu anderen Systemen regeln. Wenn diese Dateien nicht existieren, verwendet smail sinnvolle Default–Werte.

### 7.2.6 Elektronische Post mit elm

In der Un⋆x–Welt gibt es verschiedene Programme zur Erstellung und Beförderung von Email. Eins der bekanntesten Benutzer–Programme ist elm, das Ihnen ein Cursor–gesteuertes Menü zur Verfügung stellt, von dem aus Sie Botschaften versenden und die Nachrichten in Ihrem Briefkasten lesen können. Die Oberfläche von elm könnte sich Ihnen beispielweise so darstellen:

```
          Mailbox is '/usr/mail/okir' with 22 messages [ELM 2.4 PL17]


     12  Jul 28 Vince Skahan       (128)  Re: Some questions for the NAG
     13  Jul 28 Vince Skahan       (1256) Linux news/mail/uucp compilation
     14  Aug 4  Juergen Unger      (273)  Re: UUCP config (anon uucp)
     15  Aug 5  Vince Skahan       (58)   Re: anon UUCP.
     16  Oct 9  Barry Flanagan     (85)   Re: your mail
     17  Oct 21 E. Marinyak        (79)   Network configuring
     18  Oct 25 E. Marinyak        (79)   Re: Network configuring
     19  Nov 9  Shyh-Horng Jou     (58)
     20  Nov 24 Wolfgang Michaelis (54)   Re: Neues Release (war nix)
     21  Nov 28 obarriga@abello.se (42)   Okie dokie
->   22  Dec 2  Sebastian Hetze    (38)   Telefonnummer


   You can use any of the following commands by pressing the first character;
 d)elete or u)ndelete mail,  m)ail a message,  r)eply or f)orward mail,  q)uit
    To read a message, press <return>.  j = move down, k = move up, ? = help

Command:_
```

Sie sehen hier eine mäßig gefüllte Mailbox; in der Mitte werden die Nachrichten der aktuellen Mailbox mit laufender Nummer, Absender, Zeilenzahl und Betreff–Zeile angezeigt. Der Pfeil am linken Rand markiert die aktuelle Nachricht, auf die sich alle der in dem kleinen Hilfsmenü angezeigten Funktionen beziehen. Wenn Sie im obigen Beispiel an der Eingabeaufforderung RETURN eingeben, wird elm die Nachricht eines gewissen Sebastian Hetze anzeigen. Wenn Sie elm das erste Mal starten, wird Ihre Mailbox allerdings im Allgemeinen leer sein.

Um mit dem Programm vertraut zu werden, können Sie zunächst ein wenig herumexperimentieren. Haben Sie bereits smail installiert, so können Sie jetzt nämlich bereits Post an andere Benutzer auf Ihrem Rechner verschicken, ohne daß weiterer Aufwand dazu nötig wäre.

Sie starten dazu elm von der Shell aus. Wenn Sie elm zum allerersten Mal aufrufen, werden Sie danach gefragt, ob elm zwei Verzeichnisse in Ihrem Benutzer–Verzeichnis anlegen darf; antworten Sie hierauf mit "y(es)". Dann erscheint das Hauptmenü und zeigt aller Wahrscheinlichkeit nach eine leere Mailbox.

Wollen Sie nun eine Botschaft schreiben und abschicken, geben Sie an der Eingabeaufforderung w ein, und werden nacheinander nach der Adresse des Empfängers, der Betreff–Zeile, und eventuellen weiteren

[5] Eine Boolesche Variable können Sie auf false setzen, indem Sie ein Minus–Zeichen voranstellen.

Empfängern gefragt. Im folgenden Beipiel schicken sie eine Test–Botschaft an sich selbst; Ihre Eingaben sind kursiv gesetzt:

```
Command: Mail
Send the message to: karla
Subject of message: Test
Copies to: ¡Return¿
Invoking editor...
```

Anschließend ruft elm einen Editor auf, in dem Sie Ihren Brief erstellen. Der voreingestellte Editor ist meist vi. Da dies nicht jedermenschs Sache ist, stellen wir weiter unten eine Möglichkeit vor, dies zu ändern.

Nachdem Sie Ihren Brief geschrieben, abgespeichert und den Editor verlassen haben, stellt elm Ihnen folgende Frage:

```
Please choose one of the following options by parenthesized letter: s
            e)dit message, edit h)eaders, s)end it, or f)orget it.
```

Um den Brief tatsächlich abzuschicken, drücken Sie entweder s oder RETURN. Die Option e bringt Sie wieder in den Editor zurück, und f wirft den Brief weg (falls Sie sich anders entschieden haben sollten). Mit der Option h gelangen Sie in ein Menü, in dem Sie den Kopf der Botschaft editieren können; dort können Sie beispielweise die Liste der Empfänger oder die Betreff–Zeile verändern.

Nachdem Sie Ihren Brief abgeschickt haben, sollte Ihre Festplatte etwas rattern, und zunächst wieder die unverändert leere Mailbox dargestellt werden. Das liegt daran, daß smail Ihren Brief im Hintergrund ausliefert. Nachdem das Rattern der Platte dann etwas nachgelassen hat, sollten Sie irgendeine Taste drücken (beispielsweise eine Cursor–Taste); elm nimmt dies zum Anlaß, nachzuprüfen, ob neue Post eingetroffen ist. Ist alles gutgegangen, erscheint in Ihrer Mailbox jetzt Ihre Testnachricht, und Sie können sie anzeigen lassen, indem Sie die Return–Taste drücken.

### elm–Konfiguration

Um mit elm auch Post über das Netz verschicken zu können, müssen Sie ihn ebenfalls noch konfigurieren. Die elm–eigene Konfigurations–Datei befindet sich wiederum in einem Verzeichnis namens `/usr/lib/elm` oder `/usr/local/lib/elm`, und heißt `elm.rc`. Für unser Beispiel–System isis könnte sie beispielsweise so aussehen:

```
# elm.rc
#
hostname      = isis
hostdomain    = .in-berlin.de
hostfullname  = isis.in-berlin.de
#
# Transport von Umlauten etc.
charset       = iso-8859-1
displaycharset= iso-8859-1
textencoding  = 8bit
#
# Editor für Mails
editor        = /usr/bin/emacs
```

Die ersten drei Befehle teilen elm den Namen des lokalen Systems mit, aufgespalten in den eigentlichen Rechner–Namen und den Domain–Namen.

Die folgenden drei Variablen erzeugen Informationen im Mail–Kopf, die helfen sollen, Sonderzeichen heil durch das Netz zu manövrieren. Im US–ASCII–Standard–Zeichensatz sind nämlich Zeichen wie die uns so vertrauten Umlaute keineswegs vorhanden; es gibt sie nur in erweiterten Zeichensätzen, wie beispielsweise dem von DOS bekannten IBM–Zeichensatz. LINUX benutzt zur Darstellung von internationalen Zeichen einen anderen Standard, bekannt unter dem Namen ISO–8859–1. Um der transportierenden und empfangenden Mail–Software mitzuteilen, daß eventuell in der Nachricht auftauchende Umlaute in diesem Standard kodiert sind, kann elm diese Information in den Mail–Kopf einfügen.

Die letzte Variable ist die versprochene Methode, den vi–Editor durch einen anderen Editor zu ersetzen, hier durch EMACS.

Parameter, die Sie in der Datei `elm.rc` einstellen, gelten für alle Benutzer. Nun kann es aber vorkommen, daß manche Benutzer elm gerne anders konfigurieren würden. Das ist grundsätzlich möglich, da elm neben der globalen Konfigurations–Datei auch noch eine private Konfigurations–Datei kennt, nämlich `elmrc` im Verzeichnis `.elm` in Ihrem Home–Verzeichnis. Sie können diese Werte in dieser Datei auch verändern, indem Sie in elm das Konfigurations–Menü aufrufen (mit dem Befehl o) und die veränderten Werte anschließend abspeichern.

### 7.2.7 Ein Test

Um zu testen, ob Ihre Konfiguration korrekt ist, versuchen Sie, eine Nachricht an einen Benutzer auf einem anderen System zu schicken, beispielsweise an die Administatoren des UUCP–Systems, an das Sie sich angeschlossen haben.

Wenn Sie die Nachricht mit elm erstellen, geben Sie an der Eingabeaufforderung “`Send the message to:`” die volle Adresse an, z.B. root@cicero.in-berlin.de.

Nachdem Sie die Nachricht geschrieben und abgeschickt haben, können Sie mit folgendem Befehl überprüfen, ob sie auch tatsächlich an UUCP weitergegeben worden ist:

```
$ uustat -a
ciceroC0001 cicero "" 12-06 22:39 Executing rmail root (sending 436 bytes)
```

Die Ausgabe sollte ähnlich wie die zweite Zeile dieses Beispiels aussehen, die anzeigt, daß für das System cicero der Auftrag aufgegeben wurde, den Befehl “rmail root” auszuführen, und eine zusätzliche Datei von 436 Bytes — die eigentliche Mail — zu übertragen.

Sollte der Befehl uustat allerdings keine Ausgabe zeigen, ist etwas schiefgelaufen. In diesem Falle finden Sie im Verzeichnis `/usr/spool/smail/log` die Log–Dateien `logfile` und `paniclog`; diese sollten aufschlußreiche Fehlermeldungen enthalten. Falls der Fehler innerhalb der UUCP–Konfiguration zu suchen ist, können Sie auch die entsprechenden UUCP–Log–Files zu Rate ziehen. Sollten Sie mit Hilfe dieser Fehlermeldungen nicht in der Lage sein, den Fehler zu beheben, hilft wohl nur noch, zu weiterführender Dokumentation zu greifen.

# Kapitel 8

# Die Installation von Linux

Linux wird auf unterschiedlichen Wegen in verschieden gestalteten Paketen, sogenannten Distributionen, angeboten. Solche Distributionen können beliebig kopiert werden. Aus diesem Grund ist es nicht besonders schwierig an eine aktuelle Linux–Version heranzukommen. Wenn Sie sich Linux von einem Versandhändler besorgen, bekommen Sie meistens eine Installationsanleitung mitgeliefert. Die neueren Linux–Distributionen kommen mit ausgefeilten, interaktiven Installationsprogrammen, mit denen die Einrichtung eines Linux–Systems weitgehend automatisiert werden kann.

Eine Aufgabe, die jeder Linuxinstallation vorangeht, ist die Partitionierung der Festplatte und die Einrichtung eines Linux–Dateisystems auf den dafür vorgesehenen Partitionen. Weil dieser Schritt einen tiefen Eingriff in ein bestehendes System bedeutet, ist er im folgenden Abschnitt sehr ausführlich beschrieben.

## 8.1 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 Hersteller abhängig. Eine Platte mit 124Mbyte kann beispielsweise mit 1001 Zylindern, 15 Köpfen und 17 Sektoren pro Spur im CMOS eingetragen sein.

Aus verschiedenen Gründen ist es sinnvoll, die gesamte Kapazität einer Festplatte in verschiedene **Partitionen** zu unterteilen:

1. können manche Betriebssysteme nicht 124MByte 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.

### 8.1.1 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 Das Betriebssystem auch von anderen gebootet werden, allerdings muß LILO selbst immer auf einer primären Partition der ersten Festplatte installiert werden (möglicherweise auch auf der erweiterten Partitition). Weitere Hinweise dazu finden sich in der Dokumentation, die mit dem LILO-Paket verteilt wird.

### 8.1.2 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.
- Es gibt verschiedene Dateisystemmodelle zur Auswahl:
  - Das Minix Dateisystem ist immer noch der Standard für Linux. Es kann nur Dateisysteme bis zu 64 MB verwalten, beschränkt die Dateinamen auf 14/30 Zeichen, unterstützt nur eine Zeitmarke für jede Datei und läßt nur bis zu 256 verschiedene Benutzergruppen zu. Es wird von allen Kernelversionen unterstützt. Seine Bedeutung wird es vor allem als Diskettendateisystem behalten. Für Festplattenpartitionen hat es nur noch auf kleinen Systemen (<<64MB) und wegen seiner durch jahrelange Praxis bewährt einfache Struktur seine Berechtigung.
  - Das neue erweiterte Dateisystem (ext2fs) ist ein zukunftsorientiertes Modell mit vielen Features und Platz für zukünftige Entwicklung. Es kann bereits jetzt für Partitionen bis zu 2 Gigabyte verwendet werden, läßt Dateinamen bis zu 255 Zeichen zu und unterstützt die drei “üblichen” Zeitmarken für jede Datei.

  - Das von Frank Q. Xia aus dem Minix Dateisystem entwickelte xiafs kann ebenfalls Partitionen bis zu 2 GB verwalten, erlaubt Dateinamen bis 255 Zeichen Länge und unterstützt drei Zeitmarken je Datei. Das xiafs verzichtet auf weitere Features, die das ext2fs auszeichnen. Dadurch gibt es beim xiafs nur sehr begreinzten Entwicklungsspielraum, andererseits ist es damit auch weniger Fehlerträchtig.

  Eine genaue Beschreibung der Dateisysteme ist im Anhang ab Seite 287 zu finden.

- 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 TEX 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.

  - Das Root-Filesystem sollte sehr klein sein, damit es ohne Probleme häufig gesichert werden kann (beispielsweise auch auf Disketten). Auf dieser kleinen Partition sollten nur die essentiell notwendigen Systemprogramme untergebracht werden.
  - Das `/usr` Verzeichnis sollte auf einer eigenen, großen Partition angesiedelt werden. Dieses Verzeichnis trägt den Hauptast der Linux–Distribution. Es sollte so konzpiert werden, daß es "Read-Only" gemountet werden kann.
  - Wenn genügend Festplattenplatz vorhanden ist, kann für die zur Laufzeit variierenden Daten eine eigene Partition auf das `var` aufgesetzt werden. Das erhöht die die Betriebssicherheit.
  - Wenn das System von vielen verschiedenen Benutzern genutzt wird, erhöht sich die Betriebssicherheit durch eine eigene Partition für die Heimatverzeichnisse auf `/home` noch weiter. Außerdem erleichtert diese Trennung eine Vergrößerung des Anwenderbereiches.
  - Wenn Sie zum Beispiel Nachrichten aus dem USENET beziehen wollen, ist dafür eine eigene Partition praktisch, die auf dem Verzeichnis `/var/spool` aufgesetzt wird.
  - Wenn das System in größerem Umfang zum Erstellen oder Portieren von C–Programmen benutzt werden soll, kann dafür eine Partition auf das Verzeichnis `/usr/src` aufgesetzt werden.

- Um Linux von der Festplatte booten zu können, muß das Rootfilesystem auf einer primären Partition liegen.

Sie finden weitere Hinweise für die konzetionelle Gestaltung des Dateisystems im File–System–Standard, der auf Seite 25 Beschrieben ist.

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.

Sie sollten bei der Aufteilung der Festplatte auch bedenken, das auf allen Partitionen ein gewisser Bereich frei bleiben muß, damit die jeweilige Partition benutzbar bleibt. Durch die Zerstückelung der Festplatte in viele kleine Partitionen wird auf diese Weise wertvoller Platz verschenkt. Sie müssen also sehr sorgfältig alle Vor- und Nachteile abwägen, bevor Sie zur Partitionierung schreiten.

### 8.1.3 Benutzung von fdisk

Das fdisk-Programm von Linux dient zur Erstellung von Linux-Partitionen auf Festplatten. Um DOS Partitionen einzurichten, sollte das gleichnamige Programm für DOS verwendet 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: 15 heads, 17 sectors, 1001 cylinders
Units = cylinders of 255 * 512 bytes

   Device Boot  Begin   Start     End  Blocks   Id  System
/dev/hda1           1       1     402   51246+   6  DOS 16-bit >=32M
/dev/hda2   *     403     403     550   18870   81  Linux/MINIX
/dev/hda3         551     551     920   47175   83  Linux extfs
/dev/hda4         921     921    1001   10327+  82  Linux swap
```

In diesem Beispiel werden 4 Partitonen angezeigt. Es sind vier Rootpartitionen angelegt. Von den Partitionen hda2 kann gebootet werden.

Das **unit**-Kommando ändert die Einheiten, in denen die Partitionstabelle angezeigt und bearbeitet wird. Das Kommando schaltet zwischen Sektoren und Zylindern als Einheit um. Weil MS–DOS die Partitionen auf kompletten Zylindern erwartet, ist es sinnvoll mit der (voreingestellten) Zylindereinheit zu arbeiten.

Die Anzeige in Sektoren soll trotzdem am gleichen Beispiel gezeigt werden:

```
Disk /dev/hda: 15 heads, 17 sectors, 1001 cylinders
Units = sectors of 1 * 512 bytes

   Device Boot  Begin   Start     End  Blocks   Id  System
/dev/hda1           1      17  102509   51246+   6  DOS 16-bit >=32M
/dev/hda2   *  102510  102510  140249   18870   81  Linux/MINIX
/dev/hda3      140250  140250  234599   47175   83  Linux extfs
/dev/hda4      234600  234600  255254   10327+  82  Linux swap
```

Der Beginn einer Partition muß nicht mit dem Start des Datenbereichs übereinstimmen, wie das Beispiel der MS-DOS Partition zeigt, bei der 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 nicht belegten 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 der 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 nicht existierende 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 Partionen sofort neu numeriert, so daß alle logischen Partitionen immer fortlaufende Nummern haben.

Das **new** (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 Partiton 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 cylinder (403-1001): _
```

Die Zahlen bezeichnen den ersten und den letzten freien Zylinder. Es müssen nicht alle Zylinder in dem angegebenen Bereich frei sein, weil innerhalb des Bereichs noch eine komplette andere Partition liegen kann. Die Angabe eines bereits belegten Zylinders wird einfach abgelehnt, und es wird erneut nach dem ersten Zylinder der neuen Partition gefragt.

Wenn ein gültiger Zylinder für den Beginn der Partition bestimmt ist, wird nach dem letzten Zylinder gefragt:

```
Last cylinder (403-1001): _
```

Hierbei sind alle Zylinder in dem angegebenen Bereich erlaubt.

Wenn nicht alle freien Zylinder belegt worden sind, kann das Kommando erneut aufgerufen werden.

Bevor die veränderte Partitionstabelle gesichert wird, sollte 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!

### 8.1.4 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 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 (Master Boot Record) 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. LILO

### 8.1.5 Die Bedeutung des Partitionstyps

Um den verschiedenen Betriebssystemen die Möglichkeit zu geben, Partitionen anderer Formate zu erkennen, werden die Partitionen mit Kennzahlen (ID) versehen. Linux selbst kümmert sich nicht wirklich um die Partitionskennung, schließlich kann Linux mit einer Vielzahl fremder Dateisysteme gut umgehen. Der Kernel prüft beim Mounten eines Dateisystems immer die für das jeweilige Dateisystem typische magische Zahl, und erkennt so, ob es sich um ein gültiges Dateisystem eines bestimmten Typs handelt.

Andere Betriebssysteme sind da viel weniger Flexibel. In der Regel ordnet jeder Hersteller seinem Betriebssystem eine bestimmte Kennzahl für den Partitionstyp zu, mit der das Betriebssystem “seine” Partitionen markieren und erkennen kann. `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 anderen 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            40  Venix          94  Amoeba BBT
 1  DOS 12-bit FAT   51  Novell?        b7  BSDI fs
 2  XENIX root       52  Microport      b8  BSDI swap
 3  XENIX user       63  GNU HURD       c7  Syrinx
 4  DOS 16-bit <32M  64  Novell         db  CP/M
 5  Extended         75  PC/IX          e1  DOS access
 6  DOS 16-bit >=32  80  Old MINIX      e3  DOS R/O
 7  OS/2 HPFS        81  Linux/MINIX    f2  DOS secondary
 8  AIX              82  Linux swap     ff  BBT
 9  AIX bootable     83  Linux extfs
 a  OPUS             93  Amoeba
```

Voreinstellung für neue Linux–Partitionen ist ‘83’ für das Linux-Dateisystem. Diese Kennzahl ermöglicht es den anderen aufgeführten Betriebssystemen, die Linux–Partitionen als “fremd” zu erkennen.

Die ID 83 führt beim DR-DOS zu Problemen. Wenn eine Festplatte von Linux und DR-DOS gemeinsam genutzt werden soll, ist es sinnvoll anstelle der 81/83 die 42 oder 43 als Linux-ID zu verwenden.

### 8.1.6 Dateisystem einrichten

Wenn die Festplatte partitioniert ist, muß der Rechner neu gebootet werden, damit die neue Partitionstabelle ins BIOS geladen wird.

Damit aus einer Festplattenpartition eine "echte" Linux-Partition wird, muß ein Linux–Dateisystem darauf eingerichtet werden. Manche Linux–Installationsprogramme können diesen Schritt selbst ausführen. Sie sollten in diesem Fall aber besonders aufmerksam alle Bildschirmmeldungen lesen und gegebenenfalls Fragen genau beantworten, damit Sie nicht unabsichtlich eine benutzte Partition überschreiben. Die Daten wären in diesem Fall verloren.

Wenn Sie das Dateisystem "von Hand" einrichten wollen, müssen Sie sich zuerst für eines der angebotenen Modelle entscheiden. Im Anhang finden Sie eine genaue Beschreibung und Gegenüberstellung der Linux–Dateisyteme.

Wenn Sie sich für ein Dateisystem entschieden haben, müssen Sie mit dem entsprechenden mkfs–Programm die Datenstrukturen zur Verwaltung der Datenblöcke auf die Partition schreiben. Die Benutzung dieser Programme ist ab Seite 203 beschrieben.

Spätestens wenn das Dateisystem gemacht ist setzt die interaktive/automatische Installationsprozedur der Distribution ein. Am Ende der Installation wird in der Regel das neue System soweit Konfiguriert, das Sie in der Lage sind mit einer Bootdiskette oder sogar direkt von der Festplatte das neue System zu starten.

Wenn Sie soweit gekommen sind, sollten Sie eine Reise durch Ihr neues Dateisystem machen. Es ist sehr zu empfehlen, gleich zu Anfang dieser Reise einen "normalen" Benutzer–Account einzurichten. Wenn Sie mit diesem Account auf die Reise gehen, kann Sie das Betriebssystem vor unbeabsichtigten Veränderungen des Systems schützen.

## 8.2 Zusätzliche Programme installieren

Linux ist ein sehr vielseitiges System. Durch die weitgehende Übereinstimmung mit dem POSIX Standard und die darüber hinaus reichende ähnlichkeit mit Unix ist praktisch das gesamte Angebot an freier Software für Linux verfügbar, und es werden mit Sicherheit auch kommerzielle Anbieter ihre Standardprodukte für Linux anbieten, sobald die Nachfrage deutlich wird.

Die Wege an zusätzliche Programmpakete für Linux heranzukommen, sind vielfältig. Mit der Verbreitung von Linux wird das bereits vorhandene Angebot an freier Software bei den PD Händlern und in den Mailboxen wachsen. Das Tauschen von Programmen und Quelltexten ist im Bereich der freien Software ausdrücklich erlaubt.

Besonders attraktive Möglichkeiten an freie Software für Linux heranzukommen bietet sich den Besitzern eines von Linux unterstützten CD-ROM Laufwerkes. Im Versandhandel oder in guten Buchhandlungen (z.B. J.F. Lehmanns) finden Sie ein breites Angebot aktueller CDs mit Unix-Quelltexten.

Die freien Softwareprodukte sind meistens in sogenannte Tapearchive verpackt und anschließend komprimiert. Tapearchive werden vom tar Programm erzeugt und verwaltet.[1] Wie der Name schon andeutet, werden mit tar Daten auf Bandlaufwerken gespeichert. Dabei werden Verzeichnisse oder ganze Verzeichnisbäume in einem einzigen Datenstrom auf das Medium geschrieben. Das moderne GNU tar (→ Seite 186) kann den Datenstrom auch in eine normale Datei schreiben und wieder daraus lesen. Ein als Datei gespeichertes Tapearchiv erkennt man in der Regel durch die Endung '`.tar`'.

Das tar Programm faßt nur einzelne Dateien und Verzeichnisse zu einer einzigen Datei zusammen. Es komprimiert die Daten nicht. Um die in Dateien gespeicherten Tapearchive platzsparend aufzubewahren, werden sie normalerweise komprimiert. Ein komprimiertes Tapearchiv können Sie in der Regel durch eine Endung wie `.Z`, `.gz` oder `.z` erkennen.

Das GNU tar kann mit der −z Option dazu veranlaßt werden, die Kompression beziehungsweise die Dekompression automatisch auszuführen.

### Nomenklatur

Wie schon gesagt, enden viele komprimierten Tapearchive mit '`.tar.Z`'. Natürlich setzt diese Endung voraus, daß das Betriebssystem die Punkte wie normale Zeichen im Dateinamen behandelt. Wenn ein solches Tapearchiv unter MS-DOS gespeichert wird, finden Sie häufig eine Endung wie `.TAZ`.

[1] Eine weitere verbreitete Variante zum "verpacken" von Quelltexten sind die Shell–Archive (shar). Hierbei handelt es sich um Shellscript–Dokumente, wie sie auf Seite 67 beschrieben sind.

Unter Linux werden zwei verschiedene Kompressionsprogramme verwendet, gzip und compress. Der Vorteil des traditionellen compress ist seine große Verbreitung in der Unix–Welt. Wegen der deutlich besseren Verdichtung wird unter Linux fast nur noch gzip eingesetzt. 5besseren Kompressionsrate des neuen Programms. Die mit gzip komprimierten Dateien sind an der Endung `.gz` oder `.z` zu erkennen, also ein kleines anstelle des großen Z. Weil Betriebssysteme wie MS-DOS diese Unterscheidung nicht zulassen, wird hier die Endung '`.TGZ`' verwendet. Vor allem durch die Verwendung in der SLS Distribution (bis Version 1.0) hat die Endung '`.TPZ`' auch einige Verbreitung gefunden.

Wer nun meint, die Endungslehre bei den Tapearchiven sei kompliziert, wird an den eigentlichen Dateinamen gelegentlich verzweifeln. Die beste Methode den Inhalt eines solchen Archives zu ergründen, ist ein Listing des Inhaltes mit dem Kommando

```
tar tvfZ upd0.2_0.3.tar.Z
```

(oder wie immer das Objekt des Interesses heißt). Wenn sich dabei eine Datei mit dem vielsagenden Namen README zeigt, können Sie mit dem Kommando

```
tar xvfZ upd0.2_0.3.tar.Z README
```

die informative Datei aus dem Archiv herausholen.

Wenn keine solche Datei existiert oder die notwendigen Kommandos nicht ausgeführt werden können, weil die Mailbox das nicht erlaubt, hilft nur die Phantasie. Auch die kann zu beachtlichen Ergebnissen führen. Wenn man beispielsweise die oben genannte Datei in einem Verzeichnis namens 'dosemu' findet, noch dazu gemeinsam mit einer Datei namens 'dosemu0.2.tar.Z', dann handelt es sich mit an Sicherheit grenzender Wahrscheinlichkeit um das Update von Version 0.2 auf die Version 0.3 des DOS Emulators für Linux.

Zugegeben, auch diese Methode hilft nicht immer. Wer aber absolut nicht ergründen kann, was das Archiv 'stopalop0.6.tar.Z' im Verzeichnis 'BETA/packaging' enthält, oder was mit dem ka9qbin.14.tar.Z angefangen werden kann, der muß es entweder ausprobieren oder vergessen.

### Quelle oder Binär?

Es gibt zwei grundsätzlich verschiedene Distributionsformen für freie Software. Zum einen gibt es sogenannte Biärdistributionen, in denen die ausführbaren Programme (binaries) und die Hilfstexte zusammengepackt sind.

Die zweite Form verteilt anstelle der ausführbaren Dateien die Quelltexte (sources). Diese Programmtexte sind in der Regel in einer Hochsprache, meistens in C geschrieben und müssen erst von einem Compiler übersetzt werden. Die dabei entstehenden lauffähigen Programme können dann durch ein entsprechendes Kommando automatisch installiert werden, und sind dann mindestens ebenso lauffähig wie die ersten.

Die erste Distributionsform ist zweifellos weniger aufwendig. Wenn die Systemvoraussetzungen und die Anforderungen übereinstimmen, das heißt, wenn die Binärdateien für ein ähnlich konfiguriertes System erstellt worden sind.

Die zweite Form verlangt ein wenig mehr Vorbereitung. Das Ergebnis ist aber in der Regel besser an die Gegebenheiten und die Anforderungen des eigenen Systems angepaßt. Der Gestaltungsspielraum für den Systemverwalter ist hier größer.

Manchmal ist ein auf einem fremden Rechner fertig übersetztes Paket auf dem eigenen Rechner nicht lauffähig. Dann ist der Weg durch den eigenen Compiler der einzig gangbare. Weil die Binärdistributionen häufig nicht von den Programmautoren selbst erstellt werden, sind manchmal nicht die aktuellsten Programmversionen erhältlich.

### Ein Binärpaket installieren

Die ausführbaren Dateien einer Binärdistribution sollten in einem Verzeichnis installiert werden, das im Pfad der Shell liegt. Das ist jedenfalls üblich wenn die Programme allen Anwendern angeboten werden sollen. Die Manualpages müssen in einem Verzeichnis des MANPATH Pfades stehen, damit das man-Kommando darauf zugreifen kann.

Das Binärpaket enthält die Dateien zusammen mit den Verzeichnissen, oder wenigstens mit einem Teil der Verzeichnisse, meistens.

Um das geeignete Verzeichnis zum Auspacken eines fertig übersetzten Tapearchivs zu finden, ist es am besten, zuerst mit dem Kommando

`tar tvfZ irgendeinarchiv.tar.Z`

den Inhalt der Datei aufzulisten.

Wenn die Verzeichnisse mit eingapackt sind und als Teil des normalen Verzeichnisbaums erkennbar sind, sollten sie so ausgepackt werden, daß sie sich in den bestehenden Verzeichnisbaum einfügen. Wenn das Listing zum Beispiel die Verzeichnisse `./usr/bin` anzeigt, ist es ratsam das Archiv im Wurzelverzeichnis auszupacken. Um in das für normale Anwender schreibgeschützte Verzeichnis /usr/bin schreiben zu können, muß man mit Superuserprivilegien arbeiten.

Andere Orientierungspunkte bieten die man/man? und man/cat? Verzeichnisse, die jeweils unter /usr/man oder /usr/local/man eingefügt werden können.

Wenn die Verzeichnisse im Tapearchiv keinem Teil des bestehenden Verzeichnisbaums zugeordnet werden können, ist wieder die Installation in /usr und in /usr/local denkbar. Wenn die Dateien im Tapearchiv ohne Verzeichnis gespeichert sind, ist es ratsam, zuerst ein Verzeichnis dafür anzulegen und nach dem auspacken zu entscheiden, ob eventuell einzelne Dateien verschoben werden müssen.

Mit dem Kommando

`tar xvfpZ irgendeinarchiv.tar.Z`

werden die im Tapearchiv zusammengefaßten Dateien mit ihren originalen Zugriffsrechten in den ursprünglichen Verzeichnissen ausgepackt.

Bereits beim Listing des Paketes sollte auf die Lokalisierung der README (oder ähnlich suggestiv benannter Dateien) und auf die Namen der zum Paket gehörenden Manualpages geachtet werden. Die Manualpages sind typischerweise an der Endung `.1`, `.8` oder `.man` zu erkennen. Die README Datei sollte ebenso gelesen werden wie die Manualpages.[2] Diese Lektüre liefert in der Regel Hinweise auf eventuell notwendige Anpassungen und andere Hürden die vor dem ersten erfolgreichen Aufruf des neuen Programms zu nehmen sind.

### Ein Sourcepaket installieren

Weil der Compiler die Quelltexte nicht an einem bestimmten Ort erwartet, kann das Sourcearchiv in einem beliebigen Verzeichnis ausgepackt werden. Vorausgesetzt es ist genügend Platz für die Objektdateien und die fertigen Programme vorhanden.

Es ist ratsam, auch die Sourcearchive vor dem Auspacken einmal aufzulisten, um festzustellen ob es sich um eines der wenigen Archive handelt, die kein Verzeichnis für das Paket enthalten. In diesem seltenen Fall muß erst ein Verzeichnis erstellt werden, in dem das Paket dann ausgepackt werden kann. Sonst wird das Archiv einfach mit dem Kommando

`tar xvfZ das.sourcearchiv.tar.Z`

beispielsweise im Verzeichnis /usr/src ausgepackt.

Ein rekursives Listing des neuen Verzeichnisses wird mit Sicherheit viele unbekannte Dateien mit vielsagenden Namen wie tcalou.c, remunk.c, maxuxq.c, hlocnm.c[3] zum Vorschein bringen. Das macht nichts.

Zuerst empfiehlt es sich, die README Datei(en) zu lesen. Wenn andere ähnlich provokativ benannte Dateien zu finden sind, sollten auch sie gelesen werden. Der Aufwand lohnt sich meistens, weil in diesen Dateien neben einigen Ankündigungen zum Programm auch alle wichtigen Hinweise für die Übersetzung und die Installation gegeben werden.

Viele moderne Sourcedistributionen enthalten ein ausführbares Shellscript mit dem Namen configure. Dieses Script kann entweder wie ein normale Kommando aufgerufen werden, oder es wird mit einer eigenen Shell aufgerufen (`sh ./configure`).

[2] Zum Lesen der Datei(en) stehen Kommandos wie less, more oder view zur Verfügung. Die Manualpages werden normalerweise mit dem man-Kommando angezeigt. Gelegentlich werden anstelle der Manualpages sogenannte TEXinfo Dateien angeboten. Sie sind in der Regel viel ausführlicher als die Manualpages und deshalb vorzuziehen. Die Infodateien können mit TEX formatiert und mit einem geeigneten Druckertreiber ausgedruckt werden. Als Onlinehilfe können sie mit dem info-Kommando benutzt werden.

[3] Alle Namen original aus /usr/src/uucp-1.04/uuconf.

Dieses Shellscript untersucht das System, indem es bestimmte Dateien lokalisiert, die Funktionsbibliotheken durchsucht und Testprogramme übersetzt und ausführt. Durch diese Prozedur werden die Eigenschaften und Fähigkeiten von Linux meistens so gut bestimmt, daß die Übersetzung des Programms ohne weitere Schwierigkeiten durchgeführt werden kann.

Die Compilerläufe müssen nicht von Hand ausgeführt werden, sondern werden duch das make-Kommando automatisiert. Zur Steuerung von make dient das Makefile[4]. Diese Datei muß vor dem Aufruf von make in jedem Fall durchgesehen werden. Hier wird festgelegt in welchen Verzeichnissen die Programme installiert werden sollen. Häufig ist in den Makefiles der Pakete das Verzeichnis /usr/local/bin als Zielverzeichnis angegeben. Wenn die Dateien stattdessen in /usr/bin installiert werden sollen, muß hier der Prefix in /usr geändert werden.

Gelegentlich, vor allem bei den Sourcen für X11 Clients, ist anstelle des Makefiles eine Datei namens Imakefile zu finden. Das Kommando xmkmf erstellt daraus das Makefile.[5]

Je nach Umfang und Komplexität des Paketes, können noch andere Dateien zu begutachten und eventuell von Hand zu verändern sein. In jedem Fall lohnt sich der Aufwand für den Systemverwalter, der sich auf diese Weise bereits sehr gut mit dem Programm vertraut macht.

Wenn alle Vorbereitungen soweit abgeschlossen sind, wird mit dem Kommando make die Übersetzung gestartet. Durch den Aufruf

```
make -k 2>&1 | tee make.out
```

wird dieser Vorgang auch durch Fehler nicht abgebrochen, und alle Meldungen werden in der Datei make.out gespeichert. Diese Datei kann anschließend untersucht und eventuell aufgetretene Fehler behoben werden.

### Ein paar Portierungshinweise

Weil davon auszugehen ist, daß die Sourcen auf den Systemen wo sie entwickelt wurden Fehlerfrei compiliert werden können, handelt es sich bei Fehlern um Portierungsprobleme.

Die meisten Programme sind bei ihrer ersten offiziellen Auslieferung bereits auf mehreren Systemen übersetzt und getestet worden. Die systemspezifischen Anpassungen sind im Quelltext enthalten und können durch sezten bestimmter Präprozessorkonstanten aktiviert werden. Solche Einstellungen werden an einer auffälligen Stelle vorgenommen, meistens im Makefile oder in einer speziellen Headerdatei. Durch geschickte Wahl dieser Einstellung(en) kann das Programm häufig doch übersetzt werden.

Wie schon gesagt entspricht Linux dem POSIX[6] Standard. Ziel dieses Standards ist es, die Portierung und damit die Wiederverwendung von Programmen zu erleichtern. Wenn der Quelltext POSIX konform ist, sollten die entsprechenden Einstellungen gewählt werden. Wenn diese Option nicht besteht, ist es ein guter Ausgangspunkt für weitere Versuche sich am System V zu orientieren.

Einige häufig auftauchende Fehler sind leicht zu beheben. Die folgenden Tips helfen hier weiter; andere ernstere Fragen der Softwareportierung können und sollen hier nicht angesprochen werden.

**Typenfehler** (type mismatch error) treten auf, wenn das Programm eine Funktion deklariert oder gar definiert, die bereits in der Standardbibliothek enthalten ist. Dieser Fehler kann behoben werden, indem sowohl die Deklaration (Prototyp in einer Headerdatei) als auch die Definition durch die `#ifndef linux` ...`#endif` Präprozessordirektiven eingeschlossen werden. Vorher sollte sichergestellt werden, daß die Funktion im Programm sinngemäß mit der in der Bibliothek übereinstimmt. Ist das nicht der Fall hilft nur noch das Umbenennen der Funktion im Programm.

**Undefinierte Symbole** (undefined symbol error) entstehen wenn eine symbolische Konstante nicht vom Präprozessor ersetzt wird. Mit dem grep-Kommando lassen sich die Systemheader schnell nach der Konstante durchsuchen. Ist sie in einem Header definiert, muß festgestellt werden ob dieser Header

[4] Das GNU make Utility erkennt auch die Namen makefile und GNUmakefile automatisch als Steuerdateien. Alle anderen Namen müssen ausdrücklich mit der '–f' Option angegeben werden.

[5] Diese Imakefiles erleichtern die Portierung von Software erheblich. Sie enthalten programmspezifische Informationen, die vom imake Kommando gemeinsam mit systemspezifischen Daten zu einem angepaßten Makefile destiliert werden. Das Shellscript xmkmf ruft imake mit den korrekten systemspezifischen Konfigurationsdateien für X11 Clients auf. Die Konfigurationsdteien für X11 Clients sind im Verzeichnis /usr/X386/lib/X11/config zu finden.

[6] POSIX ist eine Abkürzung für Portable Operating System Interface for uniX.

vom Präprozessor überhaupt eingefügt wird (existiert eine passende `#include` Direktive?), oder ob die Definition wegen einer `#if(n)def SCHALTER` Anweisung nicht benutzt wird. Im zweiten Fall liefert der SCHALTER einen Hinweis auf mögliche Veränderungen der Einstellungen.

Wenn es sich bei den undefinierten Symbolen um SIGBUS oder SIGPWR handelt, können die Programmzeilen die die Fehlermeldungen verursachen einfach in `#ifndef linux` Direktiven ein und damit von der Übersetzung ausgeschlossen werden. Der PC erzeugt diese Signale nicht.

**Segmentation Faults** Wenn ein Programm fertig Übersetzt und gestartet werden kann, dann aber bei der ersten Gelegenheit Core dumpt, kann es daran liegen, daß es in den Datenbereich schreiben will. Linux verbietet es normalerweise den Programmen, ihren eigenen Bereich initialisierter Daten zu beschreiben und bricht sie mit einem SIGSEGV ab. Die Compileroption '`-fwritable-strings` gibt das Schreiben in Stringkonstanten frei und löst so dieses Problem vorläufig.

# Anhang A

# Dateisysteme

Für die Leistungsfähigkeit eines Betriebssystems ist die Verwaltung der Massenspeicher von großer Bedeutung. Massenspeicher, das sind vor allem Festplatten und Disketten, neuerdings aber auch zunehmend CD-ROMs und Bandlaufwerke.

Auf unterster Ebene sind diese Medien in Speicherblöcke einheitlicher Größe unterteilt. Um auf Dateien unterschiedlicher Größe einzeln Zugreifen zu können, ist eine weitere Strukturierung des Mediums notwendig. Die Organisation der Dateien in Verzeichnissen ist auf der obersten Ebene so verbreitet, daß sie selbstverständlich erscheint. Trotzdem gibt es bei der internen Realisierung dieser Struktur enorme Unterschiede.

Bei Linux treten diese Unterschiede in einer Vielfalt auf, die man fast schon als Wildwuchs bezeichnen möchte. Es gibt mindestens vier eigene reguläre Linux-Dateisysteme; dazu kommen mehrere von anderen Architekturen übernommene Dateisysteme und ein Pseudodateisystem ohne dauerhaftes Medium (das Pozeßdateisystem).

Vor der Entscheidung für oder gegen die Installation eines der angebotenen Dateisysteme auf der Festplatte oder einer Diskette muß sich die Systemverwalterin zuerst einen Überblick verschaffen. Es ist Ziel dieses Kapitels, die dazu notwendigen Begriffe zu bilden und die Zusammenhänge zu veranschaulichen. Dazu werden zuerst anhand des von MINIX übernommenen Dateisystems die grundsätzlichen Funktionen beschrieben. Danach werden die wichtigsten Komponenten der regulären Linux-Dateisysteme dargestellt. Zum Abschluß werden noch kurz die zusätzlichen Dateisystem-Varianten für Linux beschrieben. Diese Beschreibungen sollen Ihnen einen Eindruck vermitteln, welche Möglichkeiten das Dateisystem-Konzept von Linux bietet Eine ausführliche Erklärung aller Details finden Sie nicht hier, sondern in den verschiedenen Dokumenten zu den jeweiligen Source-Paketen.

## A.1 Das Minix Dateisystem

Sein erstes Dateisystem hat Linux von Minix "geborgt". Dieses Minix-Dateisystem wird "traditionell" auch heute noch als Standard für Linux eingesetzt, es wird wenigstens theoretisch von allen Linux-Installationen unterstützt. Seine einfache Struktur macht es zum bevorzugten Objekt dieser Erklärung.

Der Massenspeicher, sei es eine Diskette oder eine Festplattenpartition, stellt sich dem Betriebssystem zuerst als eine lineare Kette gleichgroßer Blöcke dar. Typischerweise sind diese Blöcke genau ein Kilobyte, das sind 1024 Bytes, groß.

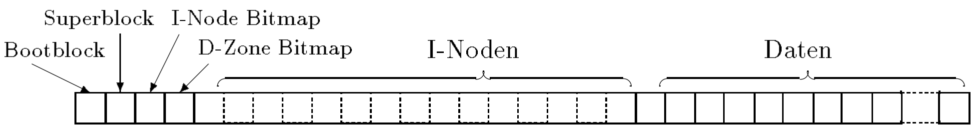


Abbildung A.1: Sechs Bereiche des Dateisystems

Das Minixdateisystem unterteilt diese Kette in sechs Bereiche. Der Aufbau ist in Abbildung A.1 dargestellt.

Die Proportionen der Bereiche sind für ein Minixdateisystem auf einer 3,5 Zoll HD Diskette dargestellt. Der Bereich für die Daten ist dabei stark verkürzt. Bei einer Gesamtzahl von 1440 Datenblöcken werden jeweils ein Block als Bootblock, Superblock, als I-Node Bitmap und als Datenzonen Bitmap verwendet. Weitere 15 Blöcke enthalten die I-Nodes, der gesamte Rest steht für Daten zur Verfügung.

<table>
<tr><td>Eigner</td><td>Gruppe</td><td>Links</td></tr>
<tr><td colspan="3">Größe</td></tr>
<tr><td colspan="3">Zeit</td></tr>
<tr><td>Typ/Rechte</td><td colspan="2">1. Zone</td></tr>
<tr><td>2. Zone</td><td colspan="2">3. Zone</td></tr>
<tr><td>4. Zone</td><td colspan="2">5. Zone</td></tr>
<tr><td>6. Zone</td><td colspan="2">7. Zone</td></tr>
<tr><td>1 indirekt</td><td colspan="2">2 indirekt</td></tr>
</table>

Abbildung A.2: Minix I-node

<table>
<tr><td>Anzahl der I-Nodes</td><td>Anzahl der Zonen</td></tr>
<tr><td>I-Node Bitmap Gr.</td><td>Zonen Bitmap Gr.</td></tr>
<tr><td>erste Datenzone</td><td>$log_2(Bl./Zone)$</td></tr>
<tr><td colspan="2">Max Dateigr.</td></tr>
<tr><td>Magie Zahl</td><td></td></tr>
</table>

Abbildung A.3: Minix Superblock

### A.1.1 I-Nodes

Der logistische Kern des Dateisystems sind die I-Nodes (Informations-Knoten). Für jede Datei existiert eine dieser Datenstrukturen, deren zentrale Komponenten die Zeiger auf die eigentlichen Datenzonen der Datei sind. Die Datenzonen bestehen aus einer konstanten Anzahl Datenblöcke. Ebenso wie die Blöcke sind auch die Zonen wie eine Kette aneinandergereiht. Damit sind die Datenzonen über ihre Nummer adressierbar. Ein Zeiger auf eine Datenzone enthält also die Nummer der Datenzone als Adresse. Wenn ein Zeiger unbenutzt ist (zum Beispiel weil die Datei 0 Bytes, also keine Daten enthält), ist die Adresse Null.

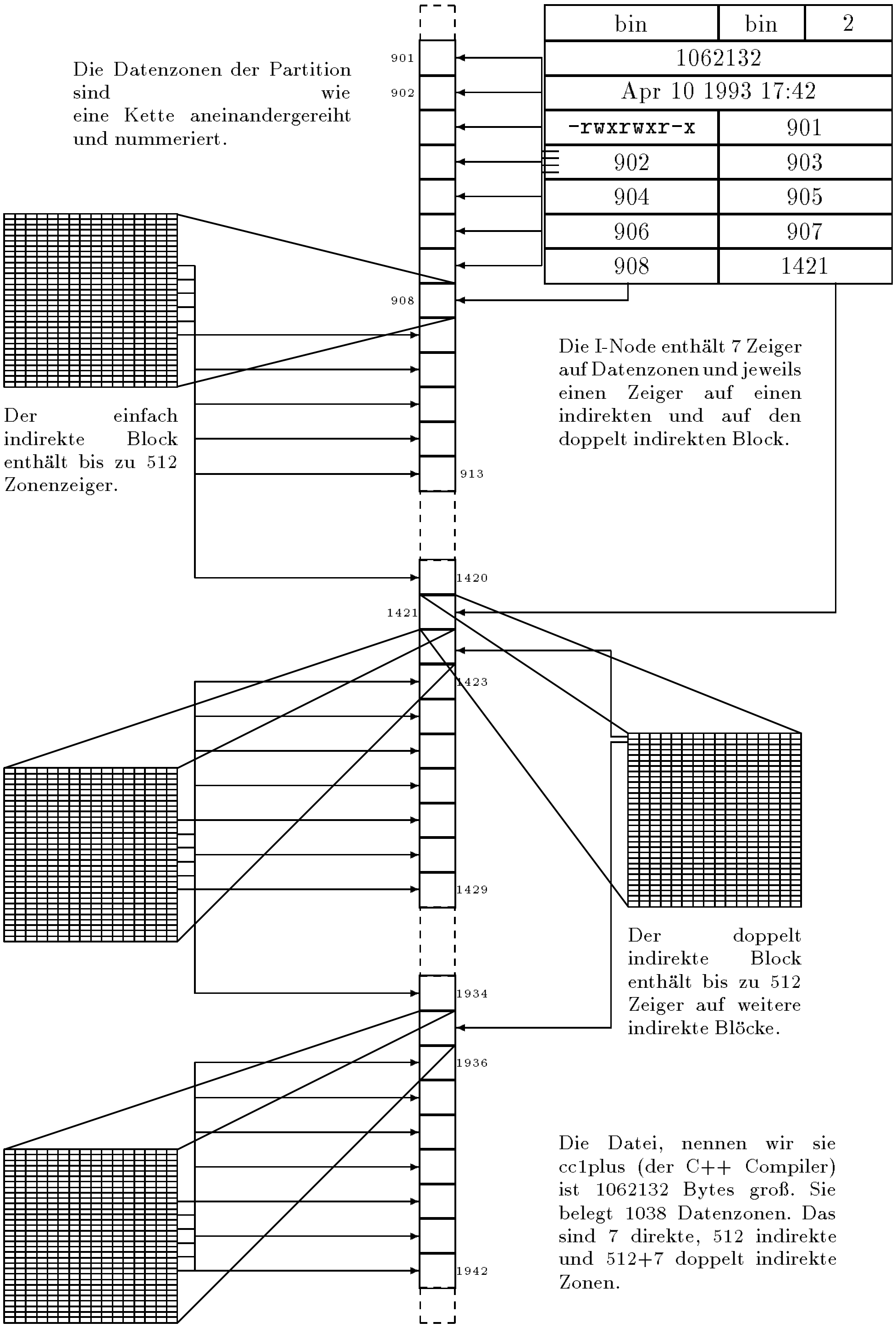


Abbildung A.4: Direkte und Indirekte Zuordnung der Datenzonen zu einer Datei

Die Datenzonen können theoretisch mehrere Blöcke groß sein, bei der normalen Konfiguration entsprechen sie genau einem Block. Damit wird für jedes angefangene kilobyte Daten ein Zeiger benötigt.

Um die Verwaltung der Datenstruktur einfach zu halten und damit die Performance des Dateisystems zu verbessern, sind alle I-Nodes gleich groß (beim Minix Dateisystem sind es 32 Bytes). Mit der festgelegten Größe ist zwangsläufig auch die Anzahl der Zeiger in der I-Node beschränkt. Beim Minix Dateisystem lassen sich höchstens sieben Datenzonen direkt aus der I-Node adressieren. Weil die Beschränkung auf eine Dateigröße von 7 Kilobyte inakzeptabel ist, werden weitere Datenzonen indirekt adressiert. Dazu gibt es einen Zeiger auf einen Datenblock, der nichts anderes als Zeiger auf die eigentlichen Datenzonen enthält.

Bei der Realisierung von Zonenzeigern durch vorzeichenlose Zweibytezahlen (unsigned short) können mit einem indirekten Zeigerblock 512 Datenzonen adressiert werden[1], was die maximale Dateigröße auf immerhin 519 Kilobyte erhöht. Weil diese Einschränkung aber immer noch unbefriedigend ist, gibt es die doppelt indirekte Adressierung. Der doppelt indirekte Zeiger zeigt auf einen Datenblock, der nur Zeiger auf einfach indirekte Zeigerblöcke in der oben besprochenen Art enthält. Auf diese Weise lassen sich 262663 Datenzonen adressieren. Das sind über 256 Megabyte, die aber wegen der Einschränkung des Minix Dateisystems auf 64 Megabyte Partitionen nicht ausgenutzt werden können.

Die Abbildung A.4 stellt diese Verbindung zwischen der I-Node und den Datenzonen dar.

Außer den Zeigern auf die Datenzonen enthalten die I-Nodes noch andere Informationen. Hier sind der Eigentümer und die Gruppe der Datei mit den jeweiligen Zugriffsrechten (Permissions) gespeichert, es gibt Informationen über den Dateityp, die Größe, die Anzahl der Links und das Datum der letzten Veränderung. Der Kernel liest bei jedem Zugriff eines Benutzers auf eine Datei die entsprechende I-Node und stellt fest, ob dieser Benutzer die erforderlichen Rechte für den angeforderten Zugriff hat. Nur wenn der Zugriff erlaubt ist, wird die entprechende Lese- oder Schreiboperation durchgeführt.

### A.1.2 Verzeichnisse

Die auf der Oberfläche das Dateisystem darstellenden Verzeichnisse sind Dateien eines speziellen Typs. Jedes Verzeichnis hat eine I-Node, in der Eigentümer und Zugriffsrechte für das Verzeichnis genau wie für eine normale Datei gespeichert sind. Das Verzeichnis selbst, das heißt die Einträge, werden in (mindestens) einem Datenblock gespeichert, der ebenso wie bei einer normalen Datei durch einen Zonenzeiger adressiert wird. Wenn ein Verzeichnis größer als 7 Kilobyte wird, werden weitere Datenblöcke indirekt adressiert; auch darin unterscheidet sich ein Verzeichnis nicht von einer normalen Datei.

[1] Genau diese Adressierung der Datenblöcke mit zwei Bytes beschränkt die Partitionsgröße im Minix-Dateisystem auf 64 Megabyte, weil mit zwei Bytes maximal 65536 Datenzonen adressiert werden können.

```
ls -ldi /bin
3 drwxr-xr-x 2 bin bin 528 Apr 3 18:00 /bin
```

Der Eintrag /bin im Wurzelverzeichnis zeigt auf die I-Node Nummer 3. In dieser I-Node sind alle Informationen zum /bin Verzeichnis enthalten. Das /bin Verzeichnis selbst ist im (ganz normalen) Datenblock Nummer 1223 gespeichert. Jeder Eintrag in diesem Datenblock ist genau 16 Bytes lang. Die ersten beiden Bytes sind wieder ein Zeiger auf eine I-Node, die wiederum alle Informationen zu dieser Datei enthält. Wenn der Name nicht die zulässigen 14 Zeichen lang ist, wird der Eintrag mit Nullbytes aufgefüllt.

| | | |
|---|---|---|
| bin | bin | 2 |
| 528 | | |
| Apr 3 1993 18:00 | | |
| drwxr-xr-x | 1223 | |
| 0 | 0 | |
| 0 | 0 | |
| 0 | 0 | |
| 0 | 0 | |

Inode Nummer 3

Datenblock Nummer 1223

| | | | |
|---|---|---|---|
| 1 | .. | 3 | . |
| 120 | awk | 49 | bash |
| 8 | cat | 9 | chgrp |
| 10 | chmod | 11 | chown |
| 55 | compress | 12 | cp |
| 52 | date | 13 | df |
| 53 | echo | 34 | getty |
| 105 | kill | 20 | ll |
| 18 | ln | 37 | login |
| 19 | ls | 20 | mkdir |
| 24 | more | 25 | mv |
| 39 | passwd | 26 | rm |
| 27 | rmdir | 49 | sh |
| 29 | stty | 41 | su |
| 87 | tar | 55 | uncompress |
| 45 | who | 44 | write |
| 55 | zcat | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Abbildung A.5: Verzeichnis im Minix FS

Jeder Eintrag in ein normales Minix-Verzeichnis wird in einem 16 Byte großen Feld gespeichert. Die ersten zwei Bytes enthalten die Nummer der zur Datei gehörenden I-Node, sie sind also ein Zeiger auf eine I-Node. Die darauffolgenden vierzehn Bytes enthalten den Dateinamen. Wenn der Dateiname kürzer als vierzehn Zeichen ist, werden die restlichen Bytes durch Nullen aufgefüllt. Eine Linux-spezifische Erweiterung des Minix-Dateisystems ermöglicht die Verwendung von Dateinamen bis zu einer Länge von 30 Zeichen, wenn das mkfs.minix-Kommando bei der Erzeugung des Dateisystems mit der entsprechenden Option (−n30) aufgerufen wurde.[2]

Jedes Verzeichnis enthält mindestens zwei Einträge, den Punkt und den Doppelpunkt. Der Zeiger des Doppelpunkt-Verzeichnisses zeigt auf die I-Node des übergeordneten Verzeichnisses, der Eintrag mit dem einfachen Punkt zeigt auf die I-Node des Verzeichnisses selbst. Diese beiden Einträge sind nötig, um ein Verzeichnis oder eine Datei relativ vom aktuellen Verzeichnis aus benennen zu können, zum Beispiel um ein übergeordntes Verzeichnis anzuzeigen.[3]

### A.1.3 Superblock und Bitmaps

Der erste Block einer Diskette oder einer Partition kann ein Programm zum Laden des Betriebssystems, den sogenannten Boot-Loader, enthalten. Deshalb heißt dieser Block Bootblock. Er wird beim Einschalten des Rechners gelesen und ausgewertet, das eigentliche Dateisystem fängt erst mit dem zweiten Block, dem Superblock, an.

Der in Abbildung A.3 dargestellte Superblock des Minix Dateisystems enthält Daten über den Typ und den Aufbau des Dateisystems. Mit einer magischen Zahl unterscheidet das Betriebssystem die unterschiedlichen Dateisystemtypen.

Durch die Anzahl der I-Nodes und die Anzahl der Datenzonen ist die Größe des Dateisystems bestimmt.

Weitere Strukturen die zur Verwaltung des Minix Dateisystems verwendet werden und deren Größe im Superblock festgehalten wird, sind die sogenannten Bitmaps. Es gibt je eine Bitmap für die I-Nodes und für die Datenzonen. Jeder I-Node beziehungsweise jeder Datenzone ist ein Bit in der entsprechenden Bitmap zugeordnet. Wenn die Zone oder die I-Node benutzt ist, wird das entsprechende Bit gesetzt.

Wenn der Kernel eine freie I-Node oder eine freie Datenzone braucht, weil eine neue Datei angelegt werden soll oder eine bestehende erweitert wird, kann mit diesen Bitmaps sehr schnell die nächste freie Stelle gefunden werden.

#### Datenzonen und Plattenblöcke

Die Datenzonen des von Linux verwendeten Minix-Dateisystems entsprechen in ihre Größe genau den Plattenblöcken. Deshalb ist die begriffliche Unterscheidung vielleicht etwas verwirrend, jedenfalls erscheint sie unnötig. Die interne Realisierung dieser Trennung erlaubt es aber, mehr als einen Plattenblock je Datenzone zu verwenden,[4] indem der Superblock entsprechend verändert wird.

Auf diese Weise könnte die Beschränkung des Minix Dateisystems auf 64 Megabyte je Partition überwunden werden. Weil aber die Veränderung der Zonengröße auf zwei oder mehr Blöcke eine Verschwendung von kostbarem Plattenplatz wäre,[5] ist mit den neuen Linux-Dateisystemen ein anderer Weg eingeschlagen worden.

---

[2] Diese Variante des Minix-Dateisystems unterscheidet sich vom Original eigentlich nur dadurch, daß die Feldgröße für einen Verzeichniseintrag genau doppelt so groß ist. Zur Unterscheidung der beiden Varianten wird eine andere magische Zahl in den Superblock eingetragen.

[3] Wenn ein Verzeichnis durch symbolische Links aus mehreren Verzeichnissen heraus angesprochen werden kann, zeigt der Doppelpunkt-Eintrag natürlich weiterhin nur auf das eine, eindeutige Verzeichnis in dem der "echte" Eintrag steht. Allein die Shell kann über ihr "Gedächtnis" den korrekten Rückweg aus den den unterschiedlichen übergeordneten Verzeichnissen auswählen. Dadurch werden sowohl relative Pfadangaben als auch das cd-Shellkommando so ausgeführt, wie Sie es als Benutzer erwarten: der symbolische Link erscheint wie ein echtes Verzeichnis. Wenn Sie aber beispielsweise mit dem ls-Kommando den Inhalt des Doppelpunkt-Verzeichnisses anzeigen, wird immer das "echte" übergeordnete Verzeichnis aufgelistet.

[4] Es können nicht beliebig viele Blöcke zu einer Zone zusammengefaßt werden. Die Anzahl muß eine Potenz von 2 sein.

[5] Die Zuteilung der Datenzonen an eine Datei kann nur in ganzen Einheiten erfolgen. Eine Fragmentierung der Datenzonen ist im Minix-Dateisystem nicht vorgesehen. Bei einer Zonengröße von zwei Blöcken ist der Verlust durch die "Aufrundung" der Dateigröße auf das nächste vielfache der Zonengröße statistisch doppelt so groß wie beim zusammenfallen von Zonen und Blöcken.

Die Anzahl der Blöcke je Datenzone ist im Superblock nicht direkt gespeichert, weil das Betriebssystem diese Zahl niemals verwendet. Stattdessen ist der Logarithmus dieser Zahl im Superblock festgehalten. Diese Zahl gibt die Anzahl der Schiebeoperationen (shifts) an, die notwendig sind, aus einer Zonenadresse die entsprechende Blockadresse zu ermitteln.

### A.1.4 Links

Wie bereits oben erklärt wurde, bestehen die Einträge eines Verzeichnisses aus dem Dateinamen und einem Zeiger auf die zu der Datei gehörenden I-Node. Der einzige Zugang zu einer Datei wird durch die I-Node hergestellt. Dieser festgelegte Weg erlaubt es auf einfache Weise, mehrere Verzeichniseinträge für ein und die selbe Datei zu erzeugen.

Die Verbindung einer I-Node (und der damit über die Zonenzeiger verbundenen Datei) mit einem Verzeichniseintrag wird als Link bezeichnet. Mit dem ln-Kommando (→ Seite 155) kann ein solcher Link auf eine bestehende Datei erzeugt werden. Das Kommando erzeugt einfach einen neuen Eintrag im aktuellen oder einem anderen angegebenen Verzeichnis, und benutzt den I-Node Zeiger der bestehenden Datei.

Weil die I-Nodes Bestandteil des Dateisystems einer in sich geschlossenen Partition ist, können solche Links nur innerhalb einer einzigen Partition bzw. einer einzigen Diskette erzeugt werden. Um Links über die Partitionsgrenzen hinweg zu ermöglichen, bietet Linux sogenannte symbolische Links an. Die symbolischen Links werden als spezieller Dateityp im Verzeichnis eingetragen, und belegen im Fall des Minix Dateisystems vier Datenzonen. In diesen Zonen ist der Zugriffsweg auf die Datei gespeichert. Es ist auch möglich, daß ein symbolischer Link ins leere zeigt, zum Beispiel wenn die Datei auf die er zeigen sollte gelöscht wurde.

#### Links auf Verzeichnisse

Wie die Beispiele des Punkt- und Doppelpunkt-Eintrags in jedem Verzeichnis zeigen, sind harte Links auf Verzeichnisse möglich. Es ist aber nur dem Superuser (root) erlaubt, solche Links anzulegen. Im Unterschied zu harten Links auf normale Dateien sind harte Links auf Verzeichnisse nicht gleichwertig, weil der rückwärts gewandte Link des Doppelpunkt-Eintrags nur auf eines der Oberverzeichnisse zeigen kann. Deshalb sollten harte Links auf Verzeichnisse nicht angelegt werden. Die bessere Lösung zum Erzeugen zusätzlicher Zugänge für Verzeichnisse sind symbolische Links.

## A.2 Die neuen Dateisysteme

Das Minix Dateisystem weist einige wichtige Eigenschaften eines Unix Dateisystems auf. Die Vereinfachungen, die es einerseits zu einem hervorragenden Lehrbeispiel machen, erweisen sich auf der anderen Seite aber als ein Mangel. Vor allem die Beschränkung auf 64 Megabyte je Partition ist für das umfangreiche Linux-System äußerst unpraktisch. Die Beschränkung der Dateinamen auf 14 Zeichen ist ebenfalls unbefriedigend. Die Unterstützung nur einer Zeitmarke (mtime) entspricht darüberhinaus nicht dem POSIX Standard, der drei Zeitmarken für eine Datei vorsieht.

#### Die Zeitmarken in der I-Node

Eine Zeitmarke, in der automatisch das Datum und die Uhrzeit der letzten Veränderung einer Datei gespeichert wird, gibt es bei den meisten Betriebssystemen (mtime). Diese Zeitmarke ermöglicht beispielsweise die Unterscheidung verschieden aktueller Versionen der gleichen Datei. Die von Unix bekannte und in den neuen Linux Dateisystemen unterstützte Speicherung von zwei zusätzlichen Zeitmarken, das Datum der letzten Statusänderung[6] und das Datum des letzten Zugriffs, gibt den Dateien zusätzliche interessante Merkmale. Mit geeigneten Kommandos wie find (→ Seite 139) und ls (→ Seite 156) kann so beispielsweise festgestellt werden ob ein Brief bereits gelesen wurde oder ob ein Text bereits überarbeitet ist.

Die Vorteile mehrerer Zeitmarken können übrigens auch bei den einfachen Dateisystemen, die nur eine Zeitmarke speichern, beobachtet werden, weil der Kernel intern im sogenannte virtuellen Dateisystem (virtual

[6] Das ist der letzte Schreibzugriff auf die I-Node.

file system, vfs) eine abstakte Verallgemeinerung aller I-Nodes benutzt. Solange sich die I-Node im Speicher befindet werden alle Zeitmarken festgehalten und benutzt. Erst wenn die I-Node auf die Platte geschrieben werden muß weil der Arbeitsspeicher für eine neue Aufgabe benötigt wird gehen die zusätzlichen Zeitmarken verloren.

#### Entwicklung

Der Weg zu einem neuen Dateisystem für Linux ist mit dem "extended filesystem" (extfs) von Remy Card begonnen worden. Dieses Dateisystem erlaubt bereits Partitionsgrößen von 2 Gigabyte und Dateinamen mit einer Länge von bis zu 255 Zeichen. Freie I-Nodes und Datenzonen werden im extfs durch verkettete Listen anstelle der Bitmaps verwaltet.[7]

Die Entwicklung eines "passenden" Dateisystems für Linux war mit dem extfs nicht zu Ende. Der experimentelle Charakter des extfs ist von Anfang an betont worden und seine Ablösung lange angekündigt.

Das neue, zweite erweiterte Dateisystem (ext2fs) erhebt den Anspruch das Linux-Dateisystem der Zukunft zu sein. In seiner 128 Bytes großen I-Node enthält es neben den erwarteten Ergänzungen, wie zum Beispiel die Unterstützung aller von POSIX definierten Zeitmarken, eine Reihe ganz neuer Attribute, die erst von zukünftigen Betriebssystemversionen und Tools unterstützt werden sollen.

Neben diesem sehr weitsichtig angelegten Modell gibt es noch weitere Entwicklungen, von denen das von Q. Frank Xia entwickelte xiafs die meiste Bedeutung erlangt hat.

### A.2.1 Das zweite erweiterte Dateisystem (ext2fs)

Die Haupteigenschaften des ext2fs sind:

- Maximale Partitionsgröße ist 2 Gigabyte. Die Erweiterung auf bis zu 4 Terabyte ist in der Struktur bereits vorgesehen.
- Die zulässige Länge für Dateinamen beträgt 1-255 Zeichen.
- Die maximale Größe einer Datei beträgt 16 Gigabyte.
- Es werden alle drei von POSIX festgelegten Zeitmarken für I-Nodes unterstützt (atime, ctime, mtime).
- Eine zusätzliche Zeitmarke für die Löschzeit ermöglicht zukünftigen Spezialprogrammen das Retten versehentlich gelöschter Dateien.
- Ein Eintrag in der I-Node kann zukünftig vom NFS-Server zur Unterscheidung verschiedener Versionen einer Datei verwendet werden.
- Zwei zusätzliche Einträge in der I-Node sind für zukünftige Erweiterung der Zugriffskontrolle vorgesehen. (Access Control List)
- Die Unterstützung fragmentierter Datenzonen mit mehr als einem Block ist vorgesehen.
- Ein Teil des Dateisystems kann dem Superuser (root) zur Benutzung vorbehalten werden.
- Der Kernel unterhält ein eigenes Cache für die Verzeichnisse des ext2fs.
- Es können 12 Datenzonen direkt adressiert werden.
- Die Unterteilung des Dateisystems in Gruppen erlaubt die Rettung von Daten auch bei Schäden in sensiblen Bereichen.
- Die symbolischen Links belegen keine Datenzonen mehr sondern speichern die Information allein in der I-Node (fast symbolic link).

[7] In diesen Listen sind die leeren I-Nodes beziehungsweise die leeren Blöcke enthalten, die jeweils nur eine laufende Nummer und einen Zeiger auf die nächste freie I-Node oder den nächsten freien Block enthalten.

<table>
<tr><td>permission</td><td colspan="2">links</td><td>owner</td><td>group</td></tr>
<tr><td colspan="3">size</td><td colspan="2">creation time</td></tr>
<tr><td colspan="3">modification time</td><td colspan="2">access time</td></tr>
<tr><td colspan="3">deletion time</td><td colspan="2">blockcount</td></tr>
<tr><td colspan="3">flags</td><td colspan="2">file version (NFS)</td></tr>
<tr><td colspan="3">file ACL</td><td colspan="2">dir ACL</td></tr>
<tr><td>fragment adr.</td><td>fr.size</td><td>frag.nr</td><td colspan="2">reserved</td></tr>
<tr><td colspan="3">1. block data</td><td colspan="2">2. block data</td></tr>
<tr><td colspan="3">3. block data</td><td colspan="2">4. block data</td></tr>
<tr><td colspan="3">5. block data</td><td colspan="2">6. block data</td></tr>
<tr><td colspan="3">7. block data</td><td colspan="2">8. block data</td></tr>
<tr><td colspan="3">9. block data</td><td colspan="2">10. block data</td></tr>
<tr><td colspan="3">11. block data</td><td colspan="2">12. block data</td></tr>
<tr><td colspan="3">simple indrect</td><td colspan="2">double indirect</td></tr>
<tr><td colspan="3">triple indirect</td><td colspan="2">reserved</td></tr>
<tr><td colspan="3">reserved</td><td colspan="2">reserved</td></tr>
</table>

Abbildung A.6: Extended 2 FS I-node

- Ein spezielles Flag im Superblock erleichtert das auffinden fehlerhafter Dateisysteme.
- Das ext2fs ist derzeit das einzige Linux-Dateisystem, bei dem die automatische Synchronisierung von Cache und Festplatte bei jeder Veränderung der Daten implementiert ist. Dabei können sogar einzelne Dateien synchron verwaltet werden.
- Ein spezielles Dateiattribut veranlaßt den Kernel, eine Datei beim Löschen mit zufälligen Daten zu überschreiben und so die Rekonstruktion der Datei sicher zu verhindern.

Die Abbildung A.6 stellt die I-Node des ext2fs dar.

Viele der Komponenten haben ihre Entsprechung in der Minix I-Node. Der auffällige Größenunterschied resultiert nur zum Teil aus den vielen zusätzlichen Einträgen. Vor allem die Vergrößerung der Zonenzeiger von zwei auf vier Bytes trägt einen weiteren Teil bei. Diese Verdoppelung ermöglicht auf der einen Seite die Adressierung von bis zu 4 Terabyte großen Partitionen.[8] Auf der anderen Seite können in einem indirekten Block nur noch 256 Zeiger untergebracht werden, was die Einführung eines dreifach indirekten Blockes sinnvoll gemacht hat. Der dreifach indirekte Block enthält Zeiger auf bis zu 256 weitere doppelt indirekte Blöcke, die jeweils wieder Zeiger auf einfach indirekte Blöcke enthalten.

Der Eintrag für die Gruppenkennzahl ist bereits seit dem ersten erweiterten Dateisystem auf zwei Bytes (unsigned short) gewachsen.

### Das Valid-Flag

Ein spezielles Byte im Superblock, das Valid-Flag, wird vom Kernel beim Aufsetzen einer ext2fs-Partition gelöscht. Wenn das Dateisystem beim regulären Systemhalt wieder abgesetzt wird, setzt der Kernel das Valid-Flag wieder. Bei einem Systemabsturz bleibt das Flag selbstverständlich ungesetzt. Indem es vor dem aufsezten eines ext2fs dieses Flag prüft, kann das Betriebssystem feststellen, ob es sich um ein gültiges (valid) Dateisystem handelt, und nötigenfalls eine Warnung ausgeben.

Das `e2fsck`-Programm testet ebenfalls dieses Flag, bevor es das Dateisystem überprüft, und überspringt die Prüfung gültiger Dateisysteme wenn es nicht durch die entsprechende Option dazu gezwungen wird. Auf diese

[8] Die Implementierung des ext2fs erlaubt zur Zeit “nur” Partitionen mit bis zu 2 Gigabyte.

Weise kann das e2fsck-Kommando automatisch beim Booten des Systems mit den Bootutilities aufgerufen werden bevor das Dateisystem zusammengesetzt wird, ohne daß deshalb nach jedem regulären Systemhalt gleich das ganze Dateisystem durchgekämmt werden muß.

Seit der Kernel-Version 0.99.10 wird auch das Root-Filesystem beim regulären Systemhalt mit den anderen Dateisystemen zusammen abgesetzt und das Valid-Flag gesetzt.

Zusätzlich zu der passiven Indikatorfunktion kann das Valid-Flag auch vom Kernel benutzt werden, um ausdrücklich ein fehlerhaftes Dateisystem zu markieren. Beim Aufsetzen eines ext2fs führt der Kernel immer eine minimalen Konsistenzprüfung durch. Wenn bei diesem Test ein Fehler festgestellt wird, setzt der Kernel das Valid-Flag auf einen speziellen Wert um damit dem e2fsck und den Bootutilities die Notwendigkeit eines Dateisystem-Checks zu signalisieren.

Außer dem Valid-Flag wird im Superblock des ext2fs noch die maximale Anzahl normaler Mounts festgehalten[9]. Beim Aufsetzen eines ext2fs wird automatisch ein Zähler im Superblock erhöht, wenn dieser Zähler größer wird als die festgelegte Maximalzahl wird beim automatischen Aufruf durch die Bootutilities ein File-System-Check durchgeführt, sonst wird eine Warnung ausgegeben.

### Datei-Attribute

In den I-Nodes des ext2fs steht ein vier Bytes großes Feld für "Flags" zur Verfügung. Von diesem 32 Bit großen Bereich ist zur Zeit nur die Bedeutung der ersten vier Bits für bestimmte Dateiattribute definiert, der Rest des Feldes steht für zukünftige Erweiterungen bereit.

Die vier definierten Attribute haben folgende Bedeutung:

- **s** (secure) Durch dieses Attribut können Dateien, deren Inhalt besonders vor dem unberechtigten Zugriff anderer Systembenutzer geschützt werden soll, zum sicheren Löschen markiert werden. Die Datenblöcke werden dann beim Löschen durch zufällige Daten überschrieben.[10]
- **u** (undelete) Noch ohne Funktion. Durch dieses Attribut soll eine Datei zukünftig auch nach dem Löschen noch intakt gehalten werden, damit das Retten der Daten durch die noch zu entwickelnde undelete-Funktion möglich ist.
- **c** (compressed) Noch ohne Funktion. Dieses Attribut wird einmal den Kernel veranlassen, die entsprechende Datei komprimiert zu speichern.
- **S** (Sync) Dieses Attribut veranlaßt den Kernel, das Dateisystem in dem sich die entsprechende Datei befindet mit dem Blockdepot synchron zu halten, solange die Datei zum Schreiben geöffnet ist.

Die Dateiattribute können Sie mit dem Kommando lsattr für alle Dateien eines ext2-Verzeichnisses anzeigen lassen. Zum Ändern der Attribute gibt es das chattr-Kommando. Veränderungen an den Dateiattributen können nur die Benutzer vornehmen, die auch die Datei selbst verändern dürfen (also schreibberechtigt sind).

### Gruppierung der Datenzonen

Das ext2fs benutzt nicht die beim Minix Dateisystem vorgestellte einfache Unterteilung der Partition in sechs Bereiche, sondern es unterteilt eine Partition zusätzlich in sogenannte Gruppen. In der aktuellen Version des ext2fs ist jede der Gruppen genau 8192 Blöcke groß; eine variable Gruppengröße ist für zukünftige Versionen vorgesehen. Die Gruppen enthalten jeweils eine Kopie des Superblockes sowie einen dem Superblock in seiner Funktion ähnlichen Gruppenblock, die Bitmaps und die I-Nodes für die Datenblöcke. Damit werden die einzelnen Gruppen fast so verwaltet wie ganze Partitionen im Minix Dateisystem.

Neben dem Geschwindigkeitsvorteil den die räumliche Nähe der I-Nodes zu den Datenblöcken vor allem bei großen Partitionen bringt, erhöht diese Unterteilung die Datensicherheit. Weil auf der Partition mehrere

[9] Die maximale Anzahl normaler Mount-Zyklen können sie mit dem tune2fs-Kommando verändern.

[10] Normalerweise werden beim Löschen einer Datei nur ein Verzeichniseintrag und, wenn keine weiteren harten Links auf die I-Node bestehen, ebendiese aus dem Dateisystem gelöscht. Die Datenblöcke bleiben erhalten, der ordentliche Zugriff ist aber wegen der fehlenden I-Node nicht mehr möglich. Wenn ein beliebiger Benutzerprozeß freie Festplattenblöcke vom Betriebssystem anfordert kann er den Dateiende-Zeiger auf das Blockende verschieben ohne selbst Daten in den Festplattenblock zu schreiben. In der so entstandenen Datei können Daten aus bereits gelöschten Dateien gelesen werden.

Kopien des Superblocks an genau definierten Stellen zu finden sind, kann der eventuell beschädigte erste Superblock vom e2fsck-Programm oder von dem speziell zu diesem Zweck entworfenen mksuper-Kommando repariert werden. Außerdem kann dem mount-Kommando als zusätzliche Option beim mounten eines Ext2-Dateisystems eine Blocknummer angegeben werden, aus der es dann den Superblock liest.

| permission | links |
|---|---|
| owner | group |
| size | |
| creation time | |
| modification time | |
| access time | |
| 1. block data | |
| 2. block data | |
| 3. block data | |
| 4. block data | |
| 5. block data | |
| 6. block data | |
| 7. block data | |
| 8. block data | |
| simple indirect | |
| double indirect | |

Abbildung A.7: Xia FS I-node

| permission | links |
|---|---|
| owner | group |
| size | |
| creation time | |
| modification time | |
| access time | |
| 1. block data | |
| 2. block data | |
| 3. block data | |
| 4. block data | |
| 5. block data | |
| 6. block data | |
| 7. block data | |
| simple indirect | |
| double indirect | |
| triple indirect | |

Abbildung A.8: New MinixFS Inode

### A.2.2 Das xiafs

Das von Frank Q. Xia entwickelte xiafs ist eine Weiterentwicklung des Minix Dateisystems. Die hervorstechenden Eigenschaften sind:

- Reservierter "Systembereich" für den Kernel und Loader im Bootsektor erlauben das booten von Festplatte.
- Partitionen bis zu zwei Gigabyte werden unterstützt, Erweiterung möglich.
- Dateinamen bis zu einer Länge von 255 Zeichen sind zugelassen.
- Alle drei von POSIX im `stat` Struct vorgesehenen Zeitmarkierungen werden unterstützt.
- Variable Zonengröße, acht direkte Zonen, je ein indirekter und doppelt indirekter Block (Dateigröße bis 64 Megabyte bei 1 Block/Zone).
- Zonen und I-Node Verwaltung mit Bitmaps.

Besonders interessant ist die Bootfähigkeit von primären xiafs Partitionen der ersten Festplatte. Der Kernel wird in einem reservierten Bereich am Anfang der Partition gespeichert und automatisch geladen wenn die Partition aktiviert ist. Der Kernel und der Loader werden mit dem mkboot Programm installiert. Zusätzlich kann das mkboot Programm einen Master Boot Loader installieren, der die Auswahl zwischen mehreren Bootpartitionen erlaubt.[11]

[11] Der Master Boot Loader ist nicht an das xiafs gebunden. Das Programm ist auf Seite 202 beschrieben.

## A.3 Bewertung

Wenn auf einer Linux-Distribution mehr als ein Dateisystem zur Installation angeboten wird, steht die Systemverwalterin vor der Qual der Wahl. Ist ein Dateisystem erst einmal Installiert und in Benutzung, kann es nicht ohne weiteres umgebaut werden. Hier soll und kann keine Ihnen nicht die eigene Entscheidung für eines der Dateisysteme abgenommen werden. Jedes Modell hat seien spezifischen Vor- und Nachteile. Eine Zusammenfassung der wichtigsten Aspekte kann Ihnen die Entscheidung aber sicher erleichtern.

Eine absolute Enscheidung ist nicht unbedingt notwendig. Die Dateisysteme sind miteinander gut verträglich, es können also mehrere Partitionen mit unterschiedlichen Dateisystemem zu einem Dateisystembaum zusammengesetzt werden. Allerdings vergrößert die Unterstützung jedes einzelnen Dateisystems den Kernel, damit wird der verbleibende Arbeitsspeicher für die Anwendungen kleiner.

Außer für Disketten, die auch weiterhin häufig mit dem Minix Dateisystem ausgerüstet werden, weil es wenig Verwaltungs-Overhead hat und uneingeschränkt durch alle Linux-Versionen unterstützt wird, fällt die Wahl wohl auf eines der neuen Dateisysteme.

Ein schwerwiegendes Kriterium ist die Funktionssicherheit eines Dateisystems. Das ext2fs hat mit der SLS-Distribution sehr weite Verbreitung gefunden und konnte sich so in der Praxis als äußerst zuverlässig beweisen. Deshalb, und weil Remy Card weiterhin am Ausbau der geplanten Features seines Dateisystems arbeitet, hat es sich zum De-Facto-Standard für Linux entwickelt.

Die vielen neuen Features und Erweiterungsmöglichkeiten geben ihm ein zukunftsorientiertes und luxuriöses Image. Allerdings bringen die Veränderungen immer auch die Gefahr neuer Bugs mit sich. Außerdem verbraucht das ext2fs mit seinen großen I-Nodes den meisten Platz für Verwaltungsstrukturen.

Das xiafs ist im Vergleich zum ext2fs einige Wochen älter und es enthält weniger Erweiterungen. Es hat sich in der Praxis aber nicht richtig durchgesetzen können und ist deswegen nicht so gut ausgetestet. Trotzdem kann ihm eine gute Stabilität bescheinigt werden. Der in das Dateisystemkonzept integrierte Bootloader ist ein wichtiger Wertungspunkt für das xiafs.

## A.4 Spezialdateisysteme

### A.4.1 Das Prozeßdateisystem

Das Prozeßdateisystem von Linux stellt zur Laufzeit des Betriebssystems Daten aus dem Kernel in der Form eines normalen Dateisystems dar. Es belegt dabei aber keinen Platz auf der Festplatte, die Verzeichnisse und Dateien existieren allein im Arbeitsspeicher des Kernels.

Mit Hilfe des Prozeßdateisystems stellt Linux eine universelle Schnittstelle zu allen relevanten Kernel-Daten zur Verfügung. Auf diesem Weg können alle Programme auf diese Daten zugreifen, ohne direkt im Kernelspeicher zu lesen. Das hat einerseits Vorteile in Hinblick auf die Datensicherheit, andererseits erspart es auch die sonst notwendige Verwaltung einer Symboltabelle in der die Speicheradressen der gesuchten Kerneldaten verzeichnet sind.

Das Prozeßdateisystem kann/muß wie jedes andere Dateisystem mit dem `mount`-Kommando auf ein existierendes Verzeichnis im Dateisystembaum aufgesetzt werden. Auch wenn im Prinzip jedes beliebige Verzeichnis als Aufsetzpunkt geeignet ist, wird in der Regel das Verzeichnis `/proc` verwendet.

```
# mount -t proc proc /proc
# ls -F /proc
1/       21/      36/      46/      57/      kcore    net/
132/     2966/    40/      47/      59/      kmsg     self/
133/     2985/    43/      48/      62/      loadavg  uptime
157/     3/       44/      55/      754/     meminfo  version
19/      34/      45/      56/      755/     modules
# _
```

Normalerweise wird das Prozeßdateisystem zusammen mit den anderen Dateisystemen in der Datei `/etc/fstab` eingetragen und beim Systemstart automatisch gemountet. Sie können aber selbstverständlich auch das Prozeßdateisystem "manuell" aufsetzen, wie im Beispiel oben gezeigt. In jedem Fall finden Sie dann im /proc-Verzeichnis eine Menge Unterverzeichnisse und ein paar Dateien.

In der Datei kcore ist der gesamte Arbeisspeicher des Rechners abgebildet. Diese Datei kann nur mit Root-Privilegien gelesen werden.

Die Datei kmsg enthält die Kernel-Meldungen, wenn sie nicht durch den syslogd-Protokollschreiber in eine andere Datei umgelenkt werden.

Aus der Datei loadavg können Sie drei Zahlen lesen, die Ihnen mitteilen wieviele Prozesse durchschnittlich innerhalb der letzten 1, 5 und 15 Minuten gelaufen sind. Diese Zahlen werden normalerweise vom uptime Kommando ausgegeben.

Die Datei meminfo wird normalerweise vom free-Kommando ausgewertet und zeigt die Auslastung des Arbeitsspeichers und des Swap-Space an.

Die Datei modules enthält eine Liste der zur Laufzeit in den Kernelspeicher geladenen und noch darin befindlichen Kernel-Module. Es werden die Modul-Namen und -Größe angezeigt. Außerdem werden die Module markiert, die geladen aber noch nicht initialisiert sind oder die gelöscht aber noch nicht aus dem Speicher entfernt worden sind. Der Inhalt dieser Datei wird normalerweise mit dem lsmod-Kommando ausgegeben.

Aus der Datei uptime können Sie die zwei Zahlen lesen. Die erste sagt Ihnen, wie viele Sekunden das System läuft, die zweite Zahl zeigt die Anzahl der Sekunden, die der Rechner im Idle-Prozeß gelaufen ist.

Die Datei version enthält den Linux-Banner mit der Versionsnummer und dem Übersetzungsdatum des laufenden Kernels.

### Die Prozeßverzeichnisse

Die meisten Unterverzeichnisse sind mit bloßen Nummern benannt. Diese Nummern entsprechen den Prozeß-IDs der aktuell in der Prozeßtabelle eingetragenen Prozesse, in den Verzeichnissen finden Sie alle relevanten Daten zu dem jeweiligen Prozeß.

Das Verzeichnis self ist so etwas wie ein Link auf das Prozeßverzeichnis des aktuellen Prozesses. Hier kann also jedes Programm auf seine eigenen Daten zugreifen ohne seine eigene Prozeßnummer zu kennen.

```
$ ls -F /proc/self/
cmdline  environ  fd/      mem      root@    statm
cwd@     exe@     maps     mmap/    stat
$ _
```

Die Datei cmdline enthält die komplette Kommandozeile für den Prozeß, wenn der Prozeß nicht vollständig in den Swapbereich ausgelagert ist.

Die Datei cwd ist ein Link auf das aktuelle Arbeitsverzeichnis des Prozesses.

Die Datei environ enthält die Daten aus der Prozeßumgebung.

Die Datei exe ist ein Link auf die ausführbare Programmdatei des Prozesses.

Das Verzeichnis fd enthält Links auf alle offenen Dateien des Prozesses.

Die Datei maps

Die Datei mem

Das Verzeichnis mmap

Die Datei root ist ein Link auf das Root-Verzeichnis des Prozesses.

Die Dateien stat und statm enthalten Statusinformationen zum Prozess aus der Prozeßtabelle. Diese Daten werden normalerweise mit dem ps-Kommando angezeigt.

### Das Verzeichnis /net

Im Verzeichnis net befinden sich sieben Dateien aus denen Sie Informationen über den Status des TCP/IP Networking im Kernel ziehen können. Normalerweise werden diese Dateien von speziellen Programmen ausgelesen, sie können sich aber alle Dateien auch einfach mit cat auf den Bildschirm schreiben lassen.

```
# ls -F /proc/net/
arp    dev    raw    route  tcp    udp    unix
# _
```

In der Datei `route` finden Sie die Routingtabelle, `arp`, `tcp` und `udp` zeigen Ihnen den Status der jeweiligen Protokoll-Layer, `unix` enthält Information über die Unix-Domain-Sockets, `raw` über "rohe" Sockets und `dev` über die Gerätetreiber auf der untersten Ebene der TCP/IP-Schichten..

# Anhang B

# Systemcalls

In der folgenden Tabelle finden Sie eine Aufstellung der 132 Systemaufrufe von Linux Version 1.0.

Bei den hier dokumentierten Funktionen handelt es sich um die "echten" Kernelfunktionen, nicht um die Bibliotheksfunktionen wie sie in C-Programmen für die Benutzerebene benutzt werden. Die Funktionsnamen aller Systemcalls beginnen mit `sys_`. Um die Tabelle übersichtlicher zu halten werden die Präfixe nicht mit aufgeführt.

In der zweiten Spalte der Tabelle finden Sie den Namen des Verzeichnisses (relativ zu `/usr/src/linux`) in dem der Sourcecode für den jeweiligen Systemaufruf zu finden ist.

| | |
|---|---|
| access(const char * filename,int mode) | fs |
| acct(void) | kernel |
| adjtimex(struct timex *txc_p) | kernel |
| alarm(long seconds) | kernel |
| break(void) | kernel |
| brk(unsigned long brk) | kernel |
| chdir(const char * filename) | fs |
| chmod(const char * filename, mode_t mode) | fs |
| chown(const char * filename, uid_t user, gid_t group) | fs |
| chroot(const char * filename) | fs |
| clone | none |
| close(unsigned int fd) | fs |
| creat(const char * pathname, int mode) | fs |
| create_module(char *module_name, unsigned long size) | kernel |
| delete_module(char *module_name) | kernel |
| dup(unsigned int fildes) | fs |
| dup2(unsigned int oldfd, unsigned int newfd) | fs |
| execve(struct pt_regs regs) | fs |
| exit(int error_code) | kernel |
| fchmod(unsigned int fd, mode_t mode) | fs |
| fchown(unsigned int fd, uid_t user, gid_t group) | fs |
| fcntl(unsigned int fd, unsigned int cmd, unsigned long arg) | fs |
| fork(struct pt_regs regs) | kernel |
| fstat(unsigned int fd, struct old_stat * statbuf) | fs |
| fstatfs(unsigned int fd, struct statfs * buf) | fs |
| fsync(unsigned int fd) | fs |

| | |
|---|---|
| ftime(void) | kernel |
| ftruncate(unsigned int fd, unsigned int length) | fs |
| get_kernel_syms(struct kernel_sym *table) | kernel |
| getegid(void) | kernel |
| geteuid(void) | kernel |
| getgid(void) | kernel |
| getgroups(int gidsetsize, gid_t *grouplist) | kernel |
| getitimer(int which, struct itimerval *value) | kernel |
| getpgrp(void) | kernel |
| getpid(void) | kernel |
| getppid(void) | kernel |
| getpriority(int which, int who) | kernel |
| getrlimit(unsigned int resource, struct rlimit *rlim) | kernel |
| getrusage(int who, struct rusage *ru) | kernel |
| gettimeofday(struct timeval *tv, struct timezone *tz) | kernel |
| getuid(void) | kernel |
| gtty(void) | kernel |
| idle(void) | mm |
| init_module(char *module_name, char *code, unsigned codesize, struct mod_routines *routines) | kernel |
| ioctl(unsigned int fd, unsigned int cmd, unsigned long arg) | fs |
| ioperm(unsigned long from, unsigned long num, int turn_on) | kernel |
| iopl(long ebx,long ecx,long edx,long esi,long edi,long ebp,long eax,long ds,long es, long fs, long gs, long orig_eax, long eip,long cs,long eflags,long esp,long ss) | kernel |
| ipc(uint call, int first, int second, int third, void *ptr) | ipc |
| SEMOP | ipc |
| SEMGET | ipc |
| SEMCTL | ipc |
| MSGSND | ipc |
| MSGRCV | ipc |
| MSGGET | ipc |
| MSGCTL | ipc |
| SHMAT | ipc |
| SHMDT | ipc |
| SHMGET | ipc |
| SHMCTL | ipc |
| kill(int pid,int sig) | kernel |
| link(const char * oldname, const char * newname) | fs |
| lock(void) | kernel |
| lseek(unsigned int fd, off_t offset, unsigned int origin) | fs |
| lstat(char * filename, struct old_stat * statbuf) | fs |
| mkdir(const char * pathname, int mode) | fs |
| mknod(const char * filename, int mode, dev_t dev) | fs |
| mmap(unsigned long *buffer) | mm |
| modify_ldt(int func, void *ptr, unsigned long bytecount) | kernel |
| mount(char * dev_name, char * dir_name, char * type, unsigned long new_flags, void * data) | fs |
| mprotect(unsigned long addr, size_t len, unsigned long prot) | mm |
| mpx(void) | kernel |
| munmap(unsigned long addr, size_t len) | mm |
| newfstat(unsigned int fd, struct new_stat * statbuf) | fs |
| newlstat(char * filename, struct new_stat * statbuf) | fs |
| newstat(char * filename, struct new_stat * statbuf) | fs |
| newuname(struct new_utsname * name) | kernel |
| nice(long increment) | kernel |
| olduname(struct oldold_utsname * name) | kernel |
| open(const char * filename,int flags,int mode) | fs |
| pause(void) | kernel |
| phys(void) | kernel |

| | |
|---|---|
| pipe(unsigned long * fildes) | fs |
| prof(void) | kernel |
| profil(void) | kernel |
| ptrace(long request, long pid, long addr, long data) | kernel |
| quotactl(void) | none |
| read(unsigned int fd,char * buf,unsigned int count) | fs |
| readdir(unsigned int fd, struct dirent * dirent, unsigned int count) | fs |
| readlink(const char * path, char * buf, int bufsiz) | fs |
| reboot(int magic, int magic_too, int flag) | kernel |
| rename(const char * oldname, const char * newname) | fs |
| rmdir(const char * pathname) | fs |
| select( unsigned long *buffer ) | fs |
| setdomainname(char *name, int len) | kernel |
| setgid(gid_t gid) | kernel |
| setgroups(int gidsetsize, gid_t *grouplist) | kernel |
| sethostname(char *name, int len) | kernel |
| setitimer(int which, struct itimerval *value, struct itimerval *ovalue) | kernel |
| setpgid(pid_t pid, pid_t pgid) | kernel |
| setpriority(int which, int who, int niceval) | kernel |
| setregid(gid_t rgid, gid_t egid) | kernel |
| setreuid(uid_t ruid, uid_t euid) | kernel |
| setrlimit(unsigned int resource, struct rlimit *rlim) | kernel |
| setsid(void) | kernel |
| settimeofday(struct timeval *tv, struct timezone *tz) | kernel |
| setuid(uid_t uid) | kernel |
| setup(void * BIOS) | drivers/block |
| sgetmask(void) | kernel |
| sigaction(int signum, const struct sigaction * action, struct sigaction * oldaction) | kernel |
| signal(int signum, unsigned long handler) | kernel |
| sigpending(sigset_t *set) | kernel |
| sigprocmask(int how, sigset_t *set, sigset_t *oset) | kernel |
| sigreturn(unsigned long __unused) | kernel |
| sigsuspend(int restart, unsigned long oldmask, unsigned long set) | kernel |
| socketcall(int call, unsigned long *args) | net |
| SYS_ACCEPT | net |
| SYS_BIND | net |
| SYS_CONNECT | net |
| SYS_GETPEERNAME | net |
| SYS_GETSOCKNAME | net |
| SYS_GETSOCKOPT | net |
| SYS_LISTEN | net |
| SYS_RECV | net |
| SYS_RECVFROM | net |
| SYS_SEND | net |
| SYS_SENDTO | net |
| SYS_SETSOCKOPT | net |
| SYS_SHUTDOWN | net |
| SYS_SOCKET | net |
| SYS_SOCKETPAIR | net |
| ssetmask(int newmask) | kernel |
| stat(char * filename, struct old_stat * statbuf) | fs |
| statfs(const char * path, struct statfs * buf) | fs |
| stime(long * tptr) | kernel |
| stty(void) | kernel |
| swapoff(const char * specialfile) | mm |
| swapon(const char * specialfile) | mm |
| symlink(const char * oldname, const char * newname) | fs |
| sync(void) | fs |
| sysinfo(struct sysinfo *info) | kernel |

| | |
|---|---|
| syslog(int type, char * buf, int len) | kernel |
| time(long * tloc) | kernel |
| times(struct tms * tbuf) | kernel |
| truncate(const char * path, unsigned int length) | fs |
| ulimit(void) | kernel |
| umask(int mask) | kernel |
| umount(char * name) | fs |
| uname(struct old_utsname * name) | kernel |
| unlink(const char * pathname) | fs |
| uselib(const char * library) | fs |
| ustat(int dev, struct ustat * ubuf) | fs |
| utime(char * filename, struct utimbuf * times) | fs |
| vhangup(void) | fs |
| vm86(struct vm86_struct * v86) | kernel |
| wait4(pid_t pid,unsigned long * stat_addr, int options, struct rusage * ru) | kernel |
| waitpid(pid_t pid,unsigned long * stat_addr, int options) | kernel |
| write(unsigned int fd,char * buf,unsigned int count) | fs |

Um Mißverständnisse auszuschließen muß hier nochmal darauf hingewiesen werden, daß es sich bei den hier aufgeführten Funktionen um die Kernelroutinen handelt. Diese Funktionen können nicht direkt aus einem Benutzerprogramm aufgerufen werden. Wenn Sie einen der Systemcalls einmal direkt ausführen wollen, müssen Sie eines der in /usr/include/syscall.h aufgeführen Makros verwenden.

Die Systemaufrufe werden normalerweise nicht direkt, sondern über eine C-Bibliotheksfunktion ausgeführt. Die Namen der Bibliotheksfunktionen stimmen mit den in der Tabelle für die Systemcalls aufgeführten in der Regel überein, die Aufrufparameter und Rückgabewerte unterscheiden sich dagegen häufig. Die genaue Syntax für den Aufruf der Bibliotheksfunktionen entnehmen Sie bitte den Manual-Pages der Sektion 2. Weil Linux dem POSIX Standard entspricht, können Sie auch jedes gute Buch über C-Programmierung in der UNIX-Umgebung zu Rate ziehen.

Die beiden Systemcalls ipc(2) und socketcall(2) fassen jeweils eine ganze Gruppe von verwandten Funktionen zusammen. In diesen Fällen werden von der C-Library anstelle der "echten" Systemaufrufe Front-Ends angeboten, die jeweils eine der speziellen Funktionen ausführen. In der Tabelle sind bei diesen beiden Systemaufrufen die Präprozessorkonstanten für den `call` Parameter aufgeführt. Die Namen der Front-End Funktionen der C-Bibliothek stimmen mit den Basisnamen der Präprozessorkonstanten überein.

In einigen weiteren Fällen werden einzelne Systemaufrufe durch mehrere Front-Ends repräsentiert. Beispielsweise flock(2) ist unter Linux eine spezielle Form des fcntl(2) Aufrufes.

# Anhang C

# Individual Network e.V.

Der Individual Network e.V. (kurz: IN) ist ein Verein mit folgenden Zielen:

- Schaffung und Sicherung von günstigen und stabilen Zugängen zu den internationalen Kommunikationsnetzen (Connectivity) für Privatpersonen.
- Eine Infrastruktur zu schaffen, so daß auch die Übertragungs- kosten minimiert werden können und diese Übertragungswege (z.B. zu Unis) rechtlich abzusichern.
- Neue Netz-Dienste, wie z.B. IP-Dienste oder X.400 zu Gebühren ermöglichen, die sich auch Privatpersonen leisten können.
- Aufbau eines "Network Management" zwischen den Teilnehmerns, um Verbindungen zuverlässig, schnell und kostengünstig zu organisieren.
- Die Vertretung von Privatpersonen nach außen und eine vernüftige Entwicklung der Netzlandschaft im Interesse der Privatpersonen.

Mit diesen Zielen im Hintergrund hat das IN 1991 seine Arbeit aufgenommen und ist heute eine Art Dachverband für Privatpersonendomains im deutschen Teil des Internet.

Die Strukturen in den einzelnen IN-Regionalgruppen (auch Domains genannt) sind vielfältig und das gesamte IN läßt sich nicht mit wenigen Worten beschreiben. Als Basisdienst kann aber jede der Domains UUCP anbieten, so daß eine Mindestversorgung mit EMail und NetNews in jedem Falle möglich ist. Weiterhin haben bereits viele Domains die Möglichkeit, echte Internet-Connectivity zu bieten. Das bedeutet, daß man sich dauerhaft (per Standleitung, ISDN oder ähnlichem) oder temporär (per Modem-Verbindung mit einem speziellen Protokoll wie SLIP oder PPP) an einen Rechner einer solchen Domain hängt und so auf seinem Linux-Rechner (oder auch Amiga, DOS, Mac und anderen Unix-Rechnern) alle Internet-Dienste nutzen kann.

Welche der Möglichkeiten man nun nutzen kann und nutzen will, hängt von mehreren Faktoren ab. Neben den finanziellen Unterschieden ist es derzeit noch so, daß nicht jede Domain Internet-Connectivity bieten kann. Ebenso sind die Möglichkeiten des Internetanschlusses sehr unterschiedlich. Ob eine Domain nur SLIP oder auch Standleitungen anbieten kann, ISDN oder Modem bevorzugt hängt unter anderem von den finanziellen Möglichkeiten der Domain ab und ist von Domain zu Domain anders.

Es gibt (für die Leute mit Internetanschluss) eine Liste aller IN-Domains mit Informationen und den entsprechenden Ansprechpartnern auf dem FTP-Server des IN:

| | |
|---|---|
| Server | : ftp.individual.net |
| Dir | : /pub/doc/IN/info |
| Datei | : INfo |

Ebenso sind dort Protokolle vergangener Mitgliederversammlungen und andere interessante Dokumente zum IN zu finden. Reinschauen lohnt sich allemal.

Wer keinen Internetzugriff hat, wohl aber eine EMail im Internet versenden kann, kann seine Fragen bei in-info@individual.net loswerden. Man erfährt dort unter anderem, wo die nächste Domain liegt, an die man sich per UUCP oder Internet anschliessen kann.

Eine weitere Informationsquelle ist die Newsgruppe de.org.in, wo die wichtigen Informationsschriften auch in regelmäßigen Abständen veröffentlicht werden.

Eine Möglichkeit, Anfragen zum IN loszuwerden, besteht auch per Modem über eine Mailbox. Unter der Nummer +49 2238 15071 (8N1 V.32bis) und dem Login "info" besteht die Möglichkeit, die IN-Infotexte online abzurufen und eine Nachricht an die IN-Info-Adresse abzusenden.

Für den Notfall (falls wirklich keine EMail oder ähnliches möglich sein sollte) kann man auch eine Anfrage mit der Post schicken. Die Anfragen werden dann nach Möglichkeit direkt an den zuständigen Ansprechpartner weiterge- leitet. Die Adresse:

Individual Network e.V.
c/o Thomas Neugebauer
Neusser Gasse 93
50259 Pulheim

Ausführliche Informationen über Kosten und Möglichkeiten einzelner Domains sind nur bei dem entsprechenden Domainverwalter erhältlich. Als groben Rahmen kann man jedoch annehmen, daß UUCP-Dienste monatlich zwischen 8,- und 30,- DM kosten, SLIP, PPP und ISDN-Internetanschlüsse monatlich zwischen 40,- DM und 80,- DM. Hinzu kommen natürlich noch die Gebühren an die DBP Telekom.

Die Domains in Kürze:

| Stadt/Region | Domain | Info-Adresse |
|---|---|---|
| Aachen | oche.de | marvin@tornado.oche.de |
| Berlin | in-berlin.de | in-berlin-admin@in-berlin.de |
| Bodensee | lake.de | chris@apollo.lake.de |
| Bonn | rhein.de | vorstand@rhein.de |
| Braunschweig | escape.de | vorstand@escape.de |
| Bremen/Oldenbg. | north.de | bbs@olis.north.de |
| Chemnitz | in-chemnitz.de | Info@IN-Chemnitz.de |
| Darmstadt/Taunus | swb.de | CB@brewhq.swb.de |
| Düsseldorf/Neuss | gun.de | andreas@easix.gun.de |
| Franken | franken.de | ac@franken.de |
| Gießen/Wetzlar | lahn.de | joscha@atab.lahn.de |
| Hamburg | hanse.de | info@hanse.de |
| Hannover | han.de | vorstand@hannet.han.de |
| Kiel | toppoint.de | verein@tpki.toppoint.de |
| Köln | dinoco.de | postmaster@dino.dinoco.de |
| Krefeld/Kempen | ish.de | postmaster@ish.de |
| Magdeburg | boerde.de | info@boerde.de |
| München | muc.de | info@muc.de |
| Münster/Osnabr. | westfalen.de | roger@larry.ms.open.de |
| Ostwestfalen/Lippe | owl.de | martin@bi-link.owl.de |
| Rhein/Main | rhein-main.de | info@rhein-main.de |
| Rodgau | robin.de | admin@robin.de |
| Ruhrgebiet | ruhr.de | info@ruhr.de |
| Saarland | saar.de | IPinfo@saar.de |
| Sachsen | sax.de | kieber@sax.sax.de |
| Schleswig-Holstein | syd.de | postmaster@syd.de |
| Thüringen | central.de | info@central.de |
| Südniedersachsen | central.de | info@central.de |
| Nordhessen | central.de | info@central.de |
| Ulm | in-ulm.de | herby@in-ulm.de |
| Wuppertal | wupper.de | info@wupper.de |

# Anhang D

# Literaturliste

### Linux

Zu Linux speziell gibt es zur Zeit (das ist im Dezember 1993) keine weiteren "echten" Bücher. Es gibt aber eine Reihe guter Texte auf elektronischen Datenträgern, die Sie sich auf dem Bildschirm durchlesen oder auf Papier ausdrucken können.
Zuerst sind die Texte des Linux–Documentation–Project zu nennen, die auf manchen Distributionen enthalten sind oder über das Internet bezogen werden können.


Johnson, M.K. *Linux Kernel Hacker's Guide*

Kirch, O. *Linux Network Administrator's Guide*

Welsh, M. *Linux Installation and Getting Started*

Wirzenius, L. *Linux System Administrator's Guide*

Greenfield, L. *Linux User's Guide*


Eine wichtige Quelle für aktuelle Information über Linux ist das USENET. Hier werden regelmäßig die HOWTOs gepostet, in denen häufig gestellte Fragen beantwortet und wertvolle Tips weitergegeben werden.


Johnson, M.K. *Linux META-FAQ*

Johnson, M.K. *Linux INFO-SHEET*

Welsh, M. *Distribution HOWTO*

Becker, D.J., Gortmaker, P. *Ethernet HOWTO*

Carp, E. *Linux Hardware Compatibility HOWTO*

Welsh, M. *The Linux Installation HOWTO*

Skahan, V. *The Linux Electronic Mail HOWTO*

Dawson, T., Welsh, M. *The Linux NET-2 HOWTO*

Skahan, V. *The Linux News HOWTO*

Taylor, G., McCauley, B. *The Linux Printing HOWTO*

Eckhardt, D. *The Linux SCSI HOWTO*

Skahan, V. *The Linux UUCP HOWTO*


Ein weiteres Linux–Buch in deutscher Sprache ist beim Springer–Verlag angekündigt:


Strobel, S., Uhl, T. *LINUX. Vom PC zur Workstation* 1994 ca. 200S. (Springer)


### Unix allgemein


Libes, D., Ressler, S. *Live with UNIX: A Guide for Everyone* 1989 368S. (Prentice Hall)

Peek, J., O'Reilly, T., Loukides, M. *UNIX Power Tools* 1993 1162S. (O'Reilly)


### Unix intern


Bach, M.J. *The Design of the UNIX Operating System* 1986 270S. (Prentice Hall)

Leffler, S.J., Karels, M.J., Quarterman, J.S. *The Design and Implementation of the 4.3 BSD UNIX Operating System* 1989 471S. (Addison-Wesley)

Tanenbaum, A.S. *Operating Systems. Design and Implementation* 1987 768S. (PH)

### Unix Systemverwaltung

Frisch, A. *Essential System Administration* 1991 466S. (O'Reilly)
Nemeth, E., Snyder, Seebass *Unix System Administration* Handbook 1989 593S. (PH)
Richter, H. *UNIX V.4 Systemverwaltung* 1993 350S. (Addison-Wesley)

### X Window System

Gilly, D., O'Reilly, T. *The X Window System in a Nutshell for Version 11, Release 4 & 5* 1992 411S. (O'Reilly)
Jones, O. *Introduction to the X Window System* 1989. (Prentice-Hall)
Mansfield, N. *The X Window System: A User's Guide* 1992 400S. (Addison-Wesley)
Mansfield, N. *The Joy of X* 1993 350S. (Addison-Wesley)
Quercia, V., O'Reilly, T. *X Window System User's Guide* 1992 752S. (O'Reilly).

### Vernetzung

Costales, B., Allman, E., Rickert, N. *Sendmail* 1993 600S. (O'Reilly)
Hahn Sout *The Internet Complete Reference* 1994 814S. (Mc Graw Hill)
Hunt, C. *TCP/IP Network Administration* 1992 502S. (O'Reilly)
Krol, E. *The Whole Internet - User's Guide* & Catalog 1992 400S. (O'Reilly)
Liu, C., Albitz, P. *DNS and BIND* 1993 418S. (O'Reilly)
O'Reilly, T., Todino, G. *Managing UUCP and Usenet* 1990 368S. (O'Reilly)
Rheingold H. *Virtuelle Gemeinschaft* 1994 390S. Addison Wesley (Addison-Wesley)
Rost, M., Schack M. *DFÜ - Recherchen in weltweiten Netzen* 1993 389S. (Heise)
Todino, G., Dougherty, D. *Using UUCP and Usenet* 1990 210S. (O'Reilly]

### Programmierung

Kernighan B., Ritchie, D. *Programmieren in C* 1988 ca280S. (Hanser)
Stevens, W.R. *Advanced Programming in the UNIX Environment* 1992 762S. (A-W)
Stevens, W.R. *UNIX Network Programming* 1990 (Prentice Hall)

# Anhang E

# GNU GENERAL PUBLIC LICENSE

Version 2, June 1991



### 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