Einführung in Linux/UNIX

Wulf Alex

2008

Karlsruhe

Ausgabedatum: 18. November 2008.

Email: alex-weingarten@t-online.de

Dies ist ein Skriptum. Es ist unvollständig und enthält Fehler.

Geschützte Namen wie UNIX oder PostScript sind nicht gekennzeichnet.

Geschrieben mit dem Texteditor `vi`, formatiert mit LaTeX unter Debian GNU/Linux.

Die Skripten liegen unter folgenden URLs zum Herunterladen bereit:

`http://www.alex-weingarten.de/skripten/`
`http://www.abklex.de/skripten/`

Besuchen Sie auch die Seiten zu meinen Büchern:

`http://www.alex-weingarten.de/debian/`
`http://www.abklex.de/debian/`

Von dem Skriptum gibt es neben der Normalausgabe eine Ausgabe in kleinerer Schrift (9 Punkte), in großer Schrift (14 Punkte) sowie eine Textausgabe für Leseprogramme (Screenreader).

There is an old system called UNIX,
suspected by many to do nix,
but in fact it does more
than all systems before,
and comprises astonishing uniques.

Vorwort

Die Skripten richten sich an Leser mit wenigen Vorkenntnissen in der Elektronischen Datenverarbeitung; sie sollen – wie FRITZ REUTERS *Urgeschicht von Meckelnborg* – ok för Schaulkinner tau bruken sin. Für die wissenschaftliche Welt zitiere ich aus dem Vorwort zu einem Buch des Mathematikers RICHARD COURANT: *Das Buch wendet sich an einen weiten Kreis: an Schüler und Lehrer, an Anfänger und Gelehrte, an Philosophen und Ingenieure.* Das Lernziel ist eine Vertrautheit mit Betriebssystemen der Gattung UNIX einschließlich Linux, der Programmiersprache C/C++ und dem weltumspannenden Internet, die so weit reicht, dass der Leser mit der Praxis beginnen und sich selbständig weiterbilden kann. Ausgelernt hat man nie.

Zusammen bildeten die Skripten die Grundlage für das Buch *UNIX. C und Internet*, im Jahr 1999 in zweiter Auflage im Springer-Verlag erschienen (ISBN 3-540-65429-1). Das Buch ist vergriffen und wird auch nicht weiter gepflegt, da ich mich auf Debian GNU/Linux konzentriere. Meine Debian-Bücher (ISBN 3-540-43267-1 und 3-540-23786-0) sind ebenfalls bei Springer erschienen, aber nicht im Netz veröffentlicht. Die Skripten dagegen bleiben weiterhin im Netz verfügbar und werden bei Gelegenheit immer wieder überarbeitet.

Warum ein Linux/UNIX? Die Betriebssysteme der Gattung UNIX laufen auf einer Vielzahl von Computertypen. Unter den verbreiteten Betriebssystemen sind sie die ältesten und ausgereift. Die UNIXe haben sich lange ohne kommerzielle Einflüsse entwickelt und tun das teilweise heute noch, siehe Linux, FreeBSD, NetBSD, OpenBSD und andere. Programmierer, nicht das Marketing, haben die Ziele gesetzt. Die UNIXe haben von Anfang an gemischte Hardware und die Zusammenarbeit mehrerer Benutzer unterstützt. In Verbindung mit dem X Window System, einem netzfähigen Fenstersystem, sind die UNIXe unter den Betriebssystemen mittlerer Größe die leistungsfähigsten. Linux/UNIX-Rechner waren von Anbeginn im Internet dabei und haben seine Entwicklung bestimmt.

Warum C/C++? Die universelle Programmiersprache C mit ihrer mächtigen Erweiterung C++ ist – im Vergleich zu BASIC etwa – ziemlich einheitlich. Der Anfang ist leicht, an die Grenzen stoßen wenige Benutzer. Das Zusammenspiel zwischen C/C++-Programmen und Linux/UNIX funktioniert reibungslos.

Warum das Internet? Das Internet ist das größte Computernetz dieser Erde, ein Zusammenschluss vieler regionaler Netze. Ursprünglich auf Hochschulen und Behörden beschränkt, sind mittlerweile auch Industrie, Handel und Privatpersonen beteiligt. Unser berufliches Leben und zunehmend unser privates Dasein werden vom Internet berührt. Eine Email-Anschrift ist so wichtig geworden wie ein Telefonanschluss. Als Informationsquelle ist das Netz unentbehrlich.

Bei der Stoffauswahl habe ich mich von meiner Arbeit als Benutzer, Verwalter und Programmierer leiten lassen. Besonderer Wert wird auf die Erläuterung der zahlreichen Fachbegriffe gelegt, die dem Anfänger das Leben erschweren. Die typische Frage, vor der auch ich immer wieder stehe, lautet: *Was ist XYZ und wozu kann man es gebrauchen?* Hinsichtlich vieler Einzelheiten verweise ich auf die Referenz-Handbücher zu den Rechenanlagen und Programmiersprachen oder auf Monografien, um den Text nicht über die Maßen aufzublähen; er ist ein Kompromiss aus Breite und Tiefe. *Alles über UNIX, C und das Internet* ist kein Buch, sondern ein Bücherschrank.

An einigen Stellen gehe ich außer auf das Wie auch auf das Warum ein. Von Zeit zu Zeit sollte man den Blick weg von den Wellen auf das Meer richten, sonst erwirbt man nur kurzlebiges Wissen.

Man kann den Gebrauch eines Betriebssystems, einer Programmiersprache oder der Netzdienste nicht allein aus Büchern erlernen – das ist wie beim Klavierspielen oder Kuchenbacken. Die Beispiele und Übungen wurden auf einer Hewlett-Packard 9000/712 unter HP-UX 10.20 und einem PC der Marke *Weingartener Katzenberg Auslese* unter Debian GNU/Linux entwickelt. Als Shell wurden Bourne-Abkömmlinge bevorzugt, als Compiler wurde neben dem von Hewlett-Packard der GNU gcc verwendet. Die vollständigen Quellen der Beispiele stehen im Netz.

Dem Text liegen eigene Erfahrungen aus fünf Jahrzehnten zugrunde. Seine Wurzeln gehen zurück auf eine *Erste Hilfe für Benutzer der Hewlett-Packard 9000 Modell 550 unter HP-UX*, im Jahr 1986 aus zwanzig Aktenordnern destilliert, die die Maschine begleiteten. Gegenwärtig verschiebt sich der Schwerpunkt in Richtung Debian GNU/Linux. Ich habe auch fremde Hilfe beansprucht und danke Kollegen in den Universitäten Karlsruhe und Lyon sowie Mitarbeitern der Firmen IBM und Hewlett-Packard für schriftliche Unterlagen und mündlichen Rat sowie zahlreichen Studenten für Anregungen und Diskussionen. Darüber hinaus habe ich fleißig das Internet angezapft und viele dort umlaufende Guides, Primers, HOWTOs, Tutorials und Sammlungen von Frequently Asked Questions (FAQs) verwendet.

Weingarten (Baden), 15. Februar 2006 Wulf Alex

Übersicht

Inhalt

Abbildungen

Tabellen

Programme und andere Quellen

Zum Gebrauch

- Hervorhebungen im Text werden *kursiv* dargestellt.
- Titel von Veröffentlichungen oder Abschnitten, kurze Zitate oder wörtliche Rede werden im Text *kursiv* markiert.
- In Aussagen über Wörter werden diese *kursiv* abgesetzt.
- Stichwörter für einen Vortrag oder eine Vorlesung erscheinen **fett**.
- Namen von Personen stehen in KAPITÄLCHEN.
- Eingaben von der Tastatur und Ausgaben auf den Bildschirm werden in `Schreibmaschinenschrift` wiedergegeben.
- Hinsichtlich der deutschen Rechtschreibung befindet sich das Manuskript in einem Übergangsstadium.
- Hinter Linux/UNIX-Kommandos folgt manchmal in Klammern die Nummer der betroffenen Sektion des Referenz-Handbuchs, z. B. `vi(1)`. Diese Nummer samt Klammern ist beim Aufruf des Kommandos nicht einzugeben.
- Suchen Sie die englische oder französische Übersetzung eines deutschen Fachwortes, so finden Sie diese bei der erstmaligen Erläuterung des deutschen Wortes.
- Suchen Sie die deutsche Übersetzung eines englischen oder französischen Fachwortes, so finden Sie einen Verweis im Sach- und Namensverzeichnis.
- UNIX wird hier immer als die Gattung der aus dem bei AT&T um 1970 entwickelten Unix ähnlichen Betriebssysteme verstanden, nicht als geschützter Name eines bestimmten Produktes.
- Ich gebe möglichst genaue Hinweise auf weiterführende Dokumente im Netz. Der Leser sei sich aber bewußt, dass sich sowohl Inhalte wie Adressen (URLs) ändern. Bei Verweisen auf Webseiten (URLs) ist die Angabe des Protokolls `http://` weggelassen.
- Unter *Benutzer, Programmierer, Verwalter* usw. werden sowohl männliche wie weibliche Erscheinungsformen verstanden.
- Ich rede den Leser mit *Sie* an, obwohl unter Studenten und im Netz das *Du* üblich ist. Gegenwärtig erscheint mir diese Wahl passender.

Rien n'est simple.
Sempé

1 Über den Umgang mit Computern

1.1 Was macht ein Computer?

Eine elektronische Datenverarbeitungsanlage, ein **Computer** oder **Rechner**, ist ein Werkzeug, mit dessen Hilfe man **Informationen**

- speichert (Änderung der zeitlichen Verfügbarkeit),
- übermittelt (Änderung der örtlichen Verfügbarkeit),
- erzeugt oder verändert (Änderung des Inhalts).

Für Informationen sagt man auch **Nachrichten** oder **Daten**[1]. Sie lassen sich durch gesprochene oder geschriebene Wörter, Zahlen, Bilder oder im Rechner durch elektrische oder magnetische Zustände darstellen. **Speichern** heißt, die Information so zu erfassen und aufzubewahren, dass sie am selben Ort zu einem späteren Zeitpunkt unverändert zur Verfügung steht. **Übermitteln** heißt, eine Information unverändert einem anderen – in der Regel, aber nicht notwendigerweise an einem anderen Ort – verfügbar zu machen, was wegen der endlichen Geschwindigkeit aller irdischen Vorgänge Zeit kostet. Da sich elektrische Transporte jedoch mit Lichtgeschwindigkeit (nahezu 300 000 km/s) fortbewegen, spielt der Zeitbedarf nur in seltenen Fällen eine Rolle. Juristen denken beim Übermitteln weniger an die Ortsänderung als an die Änderung der Verfügungsgewalt. Zum Speichern oder Übermitteln muss die physikalische Form der Information meist mehrmals verändert werden, was sich auf den Inhalt auswirken kann, aber nicht soll. **Verändern** heißt inhaltlich verändern: eingeben, suchen, auswählen, verknüpfen, sortieren, prüfen, sperren oder löschen. Tätigkeiten, die mit Listen, Karteien, Rechenschemata zu tun haben oder die mit geringen Abweichungen häufig wiederholt werden, sind mit Rechnerhilfe schneller und sicherer zu bewältigen. Rechner finden sich nicht nur in Form grauer Kästen auf oder neben Schreibtischen, sondern auch versteckt in Fotoapparaten, Waschmaschinen, Heizungsregelungen, Autos, Motorrädern und Telefonen. Diese versteckten Rechner werden *Embedded Systems* (Eingebettete Systeme) genannt.

Das Wort *Computer* stammt aus dem Englischen, wo es vor hundert Jahren eine Person bezeichnete, die berufsmäßig rechnete, einen Rechenknecht oder eine Rechenmagd. Heute versteht man nur noch die Maschinen darunter. Das englische Wort wiederum geht auf lateinisch *computare* zurück,

[1] Schon geht es los mit den Fußnoten: Bei genauem Hinsehen gibt es Unterschiede zwischen Information, Nachricht und Daten, siehe Abschnitt 2.15 *Exkurs über Informationen* auf Seite 282.

was berechnen, veranschlagen, erwägen, überlegen bedeutet. Die Franzosen sprechen vom *ordinateur*, die Spanier vom *ordenador*, dessen lateinischer Ursprung *ordo* Reihe, Ordnung bedeutet. Die Portugiesen – um sich von den Spaniern abzuheben – gebrauchen das Wort *computador*. Die Schweden nennen die Maschine *dator*, analog zu *Motor*, die Finnen *tietokone*, was *Wissensmaschine* bedeutet. Hierzulande sprach man eine Zeit lang von *Elektronengehirnen*, weniger respektvoll von *Blechbregen*. Im Deutschen verbreitet ist das Wort *Rechner*, wobei heute niemand mehr an eine Person denkt. *Rechnen* bedeutete urpsünglich *ordnen, richten*, gehört in eine vielköpfige indoeuropäische Wortfamilie und war nicht auf Zahlen beschränkt.

Die Wissenschaft von der Informationsverarbeitung ist die **Informatik**, englisch *Computer Science*, französisch *Informatique*. Ihre Wurzeln sind die **Mathematik** und die **Elektrotechnik**; kleinere Wurzelausläufer reichen auch in Wissenschaften wie Physiologie und Linguistik. Sie zählt zu den Ingenieurwissenschaften. Die früheste mir bekannte Erwähnung des Wortes *Informatik* findet sich in der Firmenzeitschrift SEG-Nachrichten (Technische Mitteilungen der Standard Elektrik Gruppe) 1957 Nr. 4, S. 171: KARL STEINBUCH, Informatik: Automatische Informationsverarbeitung. STEINBUCH berichtet in einem Referat von 1970, dass das Wort *Informatik* etwa im Jahre 1955 der Firma Standard Elektrik Lorenz AG geschützt und mit dem *Informatik-System Quelle* (Versandhaus Quelle) der Öffentlichkeit vorgestellt worden sei. Damit war das Wort geboren; die Wissenschaft von der Computerey wußte aber noch nicht, wie sie sich im deutschsprachigen Raum nennen soll. Es wurde sogar bezweifelt, dass sie eine eigene Wissenschaft sei.

Am Morgen des 26. Februar 1968 – nach dem 3. Internationalen Kolloquium über aktuelle Probleme der Rechentechnik an der TU Dresden unter der Verantwortung von NIKOLAUS JOACHIM LEHMANN – einigten sich die dort anwesenden Größen der deutschen Computerwissenschaft unter der Führung von FRIEDRICH L. BAUER nach französischem Beispiel auf die Bezeichnung *Informatik*. Diese Entscheidung war ebenso glücklich wie wichtig für die weitere Entwicklung dieser Wissenschaft im deutschen Sprachbereich: gegenüber Politik, Forschungsträgern, anderen Wissenschaften und in der Öffentlichkeit konnten nun ihre Vertreter unter einer einheitlichen und einprägsamen Bezeichnung auftreten. Kurze Zeit später wurden die *Gesellschaft für Informatik* und in Karlsruhe das *Institut für Informatik*, Direktor KARL NICKEL, gegründet. Mit dem Wintersemester 1969/70 beginnt in Karlsruhe die Informatik als eigener, voller Studiengang. Kurz darauf wird die Karlsruher Fakultät für Informatik aus der Taufe gehoben.

Der Begriff Informatik ist somit rund fünfzig Jahre alt, Computer gibt es seit siebzig Jahren, Überlegungen dazu stellten CHARLES BABBAGE vor rund zweihundert und GOTTFRIED WILHELM LEIBNIZ vor vierhundert Jahren an. Die Bedeutung der Information war dagegen schon im Altertum bekannt. Der Läufer von Marathon setzte 490 vor Christus sein Leben daran, eine Information so schnell wie möglich in die Heimat zu übermitteln. Neu in unserer Zeit ist die Möglichkeit, Informationen maschinell zu verarbeiten.

Informationsverarbeitung ist nicht an Computer gebunden. Die Informatik beschränkt sich insbesondere *nicht* auf das Herstellen von Computer-programmen. Der Computer hat jedoch die Aufgaben und die Möglichkeiten der Informatik ausgeweitet. Unter **Technischer Informatik** – im Scherz Lötkolben-Informatik genannt – versteht man den elektrotechnischen Teil. Den Gegenpol bildet die **Theoretische Informatik** – nicht zu verwechseln mit der Informationstheorie – die sich mit formalen Sprachen, Grammatiken, Semantik, Automaten, Entscheidbarkeit, Vollständigkeit und Komplexität von Problemen beschäftigt. Computer und Programme sind in der **Angewandten Informatik** zu Hause. Die Grenzen innerhalb der Informatik sowie zu den Nachbarwissenschaften sind jedoch unscharf und durchlässig.

Die heute in Hochschulen und größeren Unternehmen selbstverständlichen Rechenzentren waren anfangs umstritten. Abgesehen von den Fachleuten, die deren Daseinsberechtigung grundsätzlich bezweifelten (es ging ja auch um viel Geld), hatten manche Außenstehende die falsche Vorstellung, ein Rechenzentrum sei eine Einrichtung, bei der man abends sein Problem abliefere und am nächsten Morgen die Lösung abhole, ohne sich mit niedrigen Künsten wie numerischer Mathematik oder Programmiertechnik Finger und Hirn zu besudeln. Die Entwicklung – vor allem das Netz – hat die Aufgaben der Rechenzentren deutlicher hervortreten lassen.

Computer sind **Automaten**, Maschinen, die auf bestimmte Eingaben mit bestimmten Tätigkeiten und Ausgaben antworten. Dieselbe Eingabe führt immer zu derselben Ausgabe; darauf verlassen wir uns. Deshalb ist es im Grundsatz unmöglich, mit Computern Zufallszahlen zu erzeugen (zu würfeln). Zwischen einem Briefmarkenautomaten (Postwertzeichengeber) und einem Computer besteht jedoch ein wesentlicher Unterschied. Ein Briefmarkenautomat nimmt nur Münzen entgegen und gibt nur Briefmarken aus, mehr nicht. Es hat auch mechanische Rechenautomaten gegeben, die für spezielle Aufgaben wie die Berechnung von Geschossbahnen oder Gezeiten oder für die harmonische Analyse eingerichtet waren. Das Verhalten von mechanischen Automaten ist durch ihre Mechanik unveränderlich vorgegeben.

Bei einem Computer hingegen wird das Verhalten durch ein **Programm** bestimmt, das im Gerät gespeichert ist und leicht ausgewechselt werden kann. Derselbe Computer kann sich wie eine Schreibmaschine, eine Rechenmaschine, eine Zeichenmaschine, ein Telefon-Anrufbeantworter, ein Schachspieler oder wie ein Lexikon verhalten, je nach Programm. Er ist ein Universal-Automat. Das Wort *Programm* ist lateinisch-griechischen Ursprungs und bezeichnet ein öffentliches Schriftstück wie ein Theater- oder Parteiprogramm. Im Zusammenhang mit Computern ist an ein Arbeitsprogramm zu denken. Die englische Schreibweise ist *programme*, Computer ziehen jedoch das amerikanische *program* vor. Die Gallier reden häufiger von einem *logiciel* als von einem *programme*, wobei *logiciel* das gesamte zu einer Anwendung gehörende Programmpaket meint – bestehend aus mehreren Programmen samt Dokumentation.

Mit dem Charakter der Computer als Automaten hängt zusammen, dass sie entsetzlich dumm sind und des gesunden Menschenverstandes ermangeln. Darüber hinaus sind sie kleinlich bis zum geht nicht mehr. Ein fehlendes Komma kann sie zum Stillstand bringen. Man muss ihnen ausführlich und in exakter Sprache sagen, was sie tun sollen. Ein häufiges Problem im Umgang mit Computern ist, dass sie genau das tun, was ihnen aufgetragen wird, nicht das, was sich der Benutzer denkt oder wünscht. Andererseits zeigen sie eine Eselsgeduld. Es macht ihnen gar nichts aus, denselben Fehler millionenmal zu wiederholen. Insofern ergänzen sich Mensch und Computer hervorragend.

Ebenso wie man die Größe von Massen, Kräften oder Längen misst, werden auch **Informationsmengen** gemessen. Nun liegen Informationen in unterschiedlichen Formen vor. Sie lassen sich jedoch alle auf Folgen von zwei Zeichen zurückführen, die mit 0 und 1 oder H (high) und L (low) bezeichnet werden. Sie dürfen auch *Anna* und *Otto* dazu sagen, es müssen nur zwei verschiedene Zeichen sein. Diese einfache Darstellung wird **binär** genannt, zu lateinisch *bini* = je zwei. Die **Binärdarstellung** beliebiger Informationen durch zwei Zeichen darf nicht verwechselt werden mit dem **Dualsystem** von Zahlen, bei der die Zahlen auf Summen von Potenzen zur Basis 2 zurückgeführt werden. Das eine ist eine Darstellung oder Kodierung und gehört in die Informatik, das andere ist ein Zahlensystem und gehört in die Mathematik.

Warum bevorzugen Computer binäre Darstellungen von Informationen? Als die Rechenmaschinen noch mechanisch arbeiteten, verwendeten sie das Dezimalsystem, denn es ist einfach, Zahnräder mit 20 oder 100 Zähnen herzustellen. Viele elektronische Bauelemente hingegen kennen – von Wackelkontakten abgesehen – nur zwei Zustände wie ein Schalter, der entweder offen oder geschlossen ist. Mit binären Informationen hat es die Elektronik leichter. In der Anfangszeit hat man aber auch dezimal arbeitende elektronische Rechner gebaut. Hätten wir brauchbare Schaltelemente mit drei oder vier Zuständen, würden wir auch ternäre oder quaternäre Darstellungen verwenden.

Eine 0 oder 1 stellt eine Binärziffer dar, englisch binary digit, abgekürzt Bit. Ein **Bit** ist das Datenatom. Hingegen ist 1 bit (kleingeschrieben) die Maßeinheit für die Entscheidung zwischen 0 und 1 im Sinne der Informationstheorie von CLAUDE ELWOOD SHANNON. Kombinationen von acht Bits spielen eine große Rolle, sie werden daher zu einem **Byte** oder **Oktett** zusammengefasst. Die Festlegung des Bytes zu acht Bit stammt von der Firma IBM aus dem Jahr 1964; davor waren auch andere Werte in Gebrauch. Auf dem Papier wird ein Byte oft durch ein Paar hexadezimaler Ziffern – ein **Hexpärchen** – wiedergegeben. Das **Hexadezimalsystem** (Sedezimalsystem) – das Zahlensystem zur Basis 16 – wird uns häufig begegnen, in Linux/UNIX auch das **Oktalsystem** zur Basis 8. Durch ein Byte lassen sich $2^8 = 256$ unterschiedliche Zeichen darstellen. Das reicht für unsere europäischen Buchstaben, Ziffern und Satzzeichen. Ebenso wird mit einem Byte eine Farbe aus 256 unterschiedlichen Farben ausgewählt. 1024 Byte ergeben 1 Kilobyte, 1024 Kilobyte sind 1 Megabyte, 1024 Megabyte sind 1 Gigabyte, 1024 Gigabyte machen 1 Terabyte (mit *einem* r, aus dem Griechischen). Die nächste Stufen heißen Petabyte, Exabyte und Zettabyte.

Der Computer verarbeitet Informationen in Einheiten eines **Maschinenwortes**, das je nach der Breite der Datenregister des Prozessors 1 bis 16 Bytes (8 bis 128 Bits) umfasst. Der durchschnittliche Benutzer kommt mit dieser Einheit selten in Berührung.

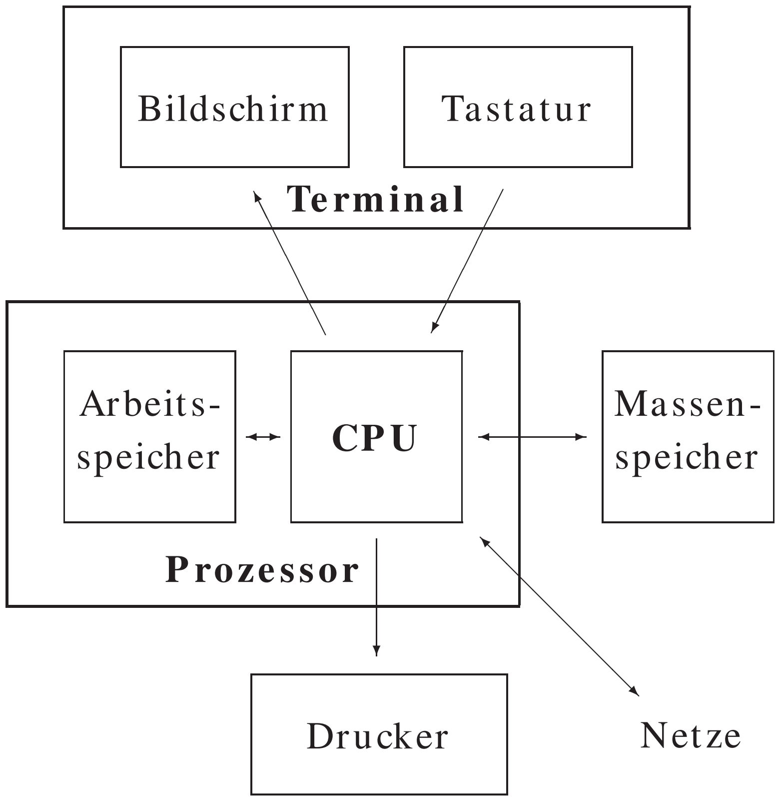

Abb. 1.1: Aufbau eines Rechners

1.2 Woraus besteht ein Rechner?

Der Benutzer sieht von einem Rechner vor allem den **Bildschirm**[2] (E: screen, F: écran) und die **Tastatur** (E: keyboard, F: clavier), auch Hackbrett genannt. Die beiden Geräte werden zusammen als **Terminal** (E: terminal, F: terminal) bezeichnet und stellen die Verbindung zwischen Benutzer und Rechner dar. Mittels der Tastatur spricht der Benutzer zum Rechner, auf dem Bildschirm erscheint die Antwort. Eine etwaige Maus oder Rollkugel zählt zur Tastatur.

Der eigentliche Rechner, die **Prozessoreinheit** (Zentraleinheit, E: central unit, F: unité centrale) ist in die Tastatur eingebaut wie beim Schneider CPC 464 oder Commodore C64, in das Bildschirmgehäuse wie beim ersten Apple Macintosh oder in ein eigenes Gehäuse. Seine wichtigsten Teile sind der **Zentralprozessor** (CPU, E: central processing unit, F: processeur central) und der **Arbeitsspeicher** (E: memory, F: mémoire centrale, mémoire vive, mémoire secondaire).

Um recht in Freuden arbeiten zu können, braucht man noch einen **Massenspeicher** (E: mass storage, F: mémoire de masse), der seinen Inhalt nicht vergisst, wenn der Rechner ausgeschaltet wird. Nach dem heutigen Stand der Technik arbeiten die meisten Massenspeicher mit magnetischen Datenträgern ähnlich wie Ton- oder Videobandgeräte. Tatsächlich verwendeten die

[2]Aus der Fernsehtechnik kommend wird der Bildschirm oft Monitor genannt. Da das Wort hier nicht ganz trifft und auch ein Programm bezeichnet, vermeide ich es.

ersten Personal Computer Tonbandkassetten. Weit verbreitet sind scheibenförmige magnetische Datenträger in Form von **Disketten** (E: floppy disk, F: disquette) und **Festplatten** (E: hard disk, F: disque dur).

Disketten, auch Schlappscheiben genannt, werden nach Gebrauch aus dem **Laufwerk** (E: drive, F: dérouleur) des Rechners herausgenommen und im Schreibtisch vergraben oder mit der Post verschickt. Festplatten verbleiben in ihrem Laufwerk. CDs und DVDs sind ebenfalls entfernbare, transportable Datenträger.

Da man gelegentlich etwas schwarz auf weiß besitzen möchte, gehört zu den meisten Rechnern ein **Drucker** (E: printer, F: imprimante). Ferner ist ein Rechner, der etwas auf sich hält, heutzutage durch ein **Netz** (E: network, F: réseau) mit anderen Rechnern rund um die Welt verbunden. Damit ist die Anlage vollständig.

Was um den eigentlichen Rechner (Prozessoreinheit) herumsteht, wird als **Peripherie** (E: periphery, F: périphérique) bezeichnet. Die peripheren Geräte sind über **Schnittstellen** (Datensteckdosen, E: interface, F: interface) angeschlossen.

In Abb. 1.1 sehen wir das Ganze schematisch dargestellt. In der Mitte die CPU, untrennbar damit verbunden der Arbeitsspeicher. Um dieses Paar herum die Peripherie, bestehend aus Terminal, Massenspeicher, Drucker und Netzanschluss. Sie können aber immer noch nichts damit anfangen, allenfalls heizen. Es fehlt noch die Intelligenz in Form eines **Betriebssystems** (E: operating system, F: système d'exploitation) wie Linux oder UNIX.

1.3 Was muss man wissen?

Die ersten Gedanken des Anfängers werden darum kreisen, wie man dem Rechner vernünftige Reaktionen entlockt. Keine Angst: durch Tastatureingaben (außer Kaffee und ähnlichen Programming Fluids) ist ein Rechner nicht zu zerstören. Die in ihm gespeicherten Daten sind allerdings empfindlich. Zum Arbeiten mit einem Rechner muss man drei Dinge lernen:

- den Umgang mit der **Hardware**
- den Umgang mit dem **Betriebssystem**,
- den Umgang mit einem **Anwendungsprogramm**, zum Beispiel einer Textverarbeitung oder Datenbank.

Darüber hinaus sind **Englischkenntnisse** und Übung im **Maschinenschreiben** nützlich. Das Lernen besteht zunächst darin, sich einige hundert Begriffe anzueignen. Das ist in jedem Wissensgebiet so. Man kann nicht über Primzahlen, Wahrscheinlichkeitsamplituden, Sonette oder Sonaten nachdenken oder reden, ohne sich vorher über die Begriffe klar geworden zu sein.

Die **Hardware** (E: hardware, F: matériel) umschließt alles, was aus Kupfer, Eisen, Kunststoffen, Glas und dergleichen besteht, was man anfassen kann. Dichterfürst FRIEDRICH VON SCHILLER hat den Begriff Hardware trefflich gekennzeichnet:

Leicht beieinander wohnen die Gedanken,
doch hart im Raume stoßen sich die Sachen.

Die Verse stehen in *Wallensteins Tod* im 2. Aufzug, 2. Auftritt. Was sich hart im Raume stößt, gehört zur Hardware, was leicht beieinander wohnt, die Gedanken, ist **Software** (E: software, F: logiciel). Die Gedanken stecken in den Programmen und den Daten.

Die reine Hardware – ohne Betriebssystem – tut nichts anderes als elektrische Energie in Wärme zu verwandeln. Sie ist ein Ofen, mehr nicht. Das **Betriebssystem** ist ein Programm, das diesen Ofen befähigt, Daten einzulesen und in bestimmter Weise zu antworten. Hardware plus Betriebssystem machen den Rechner aus. Diese Kombination wird von vielen als **System** bezeichnet. Manche sagen auch Plattform dazu. Eine bestimmte Hardware kann mit verschiedenen Betriebssystemen laufen, umgekehrt kann dasselbe Betriebssystem auch auf unterschiedlicher Hardware laufen (gerade das ist eine Stärke von Linux/UNIX).

Bekannte Betriebssysteme sind DOS von IBM, Microsoft oder Novell, Windows von Microsoft sowie IBM OS/2 für IBM-PCs und ihre Verwandtschaft, MacOS für die Macintoshs, VMS für die VAXen der Digital Equipment Corporation (DEC) sowie die UNIX-Familie einschließlich Linux und FreeBSD für eine ganze Reihe von mittleren Rechnern verschiedener Hersteller.

Um eine bestimmte Aufgabe zu erledigen – um einen Text zu schreiben, ein Gleichungssystem zu lösen oder ein Getriebe zu konstruieren – braucht man ein **Anwendungsprogramm** (E: application program, F: logiciel d'application). Dieses erwirbt man fertig, zum Beispiel ein Programm zur Textverarbeitung oder zur Tabellenkalkulation, oder schreibt es selbst. In diesem Fall muss man eine **Programmiersprache** (E: programming language, F: langage de programmation) beherrschen. Die bekanntesten Sprachen sind BASIC, COBOL, FORTRAN, JAVA, PASCAL und C/C++. Es gibt mehr als tausend[3]. Bei der Universität Genf wird eine Liste der Programmiersprachen geführt:

`http://cui.unige.ch/langlist`

Das nötige Wissen kann man auf mehreren Wegen erwerben und auf dem laufenden halten:

- Kurse, Vorlesungen
- Lehrbücher, Skripten
- Zeitschriften
- Electronic Information
- Lernprogramme

[3]Zum Vergleich: es gibt etwa 6000 lebende natürliche Sprachen. Die Bibel – oder Teile von ihr – ist in rund 2000 Sprachen übersetzt.

Gute **Kurse** oder **Vorlesungen** verbinden Theorie und Praxis, das heißt Unterricht und Übungen am Rechner. Zudem kann man Fragen stellen und bekommt Antworten. Nachteilig ist der feste Zeitplan. Die schwierigen Fragen tauchen immer erst nach Kursende auf. Viele Kurse sind auch teuer.

Seit 1974 gibt es einen Europäischen Computer-Führerschein (European Computer Driving Licence, ECDL) samt zugehörigem Ausbildungs- und Prüfungswesen sowie Webseiten. Der Prüfungsplan (Syllabus) sieht Microsoft-lastig aus und wendet sich eher an Büroberufe als an technische oder naturwissenschaftliche Benutzer. Für diesen Kreis sind vor allem die Themen Computer-Grafik und -Algebra sowie Programmieren zu ergänzen. Aber der im Netz zu findende Syllabus ist eine Hilfe beim Abchecken der eigenen Fähigkeiten.

Bei Büchern und verwandten Werken sind zu unterscheiden:

- Lehrbücher
  - Einführungen, Primer
  - Einzelwerke, Monografien
- Nachschlagewerke
  - Glossare, Lexika
  - Referenz-Handbücher, Standards, Normen

Breit angelegte **Lehrbücher** führen durch ein Wissensgebiet, treffen eine Auswahl, werten oder diskutieren und verzichten zu Gunsten der Verständlichkeit auf viele Einzelheiten. **Einzelwerke** behandeln ein enges Thema ausführlich. Sie gehen in eine Tiefe, die den Anfänger verwirren würde. Zu einer Reihe von Linux/UNIX-Kommandos, Programmierfragen und Netzdiensten gibt es solche vertiefenden Werke, siehe Anhang L *Zum Weiterlesen* ab Seite 358.

Glossare und **Lexika** sind nach Stichwörtern alphabetisch geordnet und dienen der schnellen Information, dem Einordnen eines Begriffes in einen größeren Zusammenhang und dem Erschließen verwandter Stichwörter, gegebenenfalls in weiteren Sprachen. Außer in Buchform erscheinen viele Glossare im Netz. Die **Wikipedia** hat sich zu einer allgemeinen Enzyklopädie im Netz entwickelt und erläutert auch viele Begriffe aus der Informatik:

- `en.wikipedia.org/` mit einer Million Einträgen,
- `de.wikipedia.org/` mit 350.000 Einträgen,
- `fr.wikipedia.org/` mit 250.000 Einträgen.

Mit einer gewissen Vorsicht genossen – die man auch einem Skriptum entgegenbringen sollte – ist die Wikipedia eine brauchbare Informationsquelle.

Referenzen müssen vollständig und eindeutig sein. Sie beschreiben alle Einzelheiten und helfen bei allgemeinen Schwierigkeiten gar nicht. Will man wissen, welche Werkzeuge Linux/UNIX zur Textverarbeitung bereit hält, braucht man ein Lehrbuch. Will man hingegen wissen, wie man den Editor `vi` veranlasst, nach einer Zeichenfolge zu suchen, so schlägt man im Referenz-Handbuch nach. Auf Linux/UNIX-Systemen ist das Referenz-Handbuch online (auf dem Bildschirm) verfügbar.

Die Einträge im Linux/UNIX-Referenz-Handbuch sind knapp gehalten. Bei einfachen Kommandos wie `pwd` oder `who` sind sie dennoch auf den ersten Blick verständlich. Zu Kommandos wie `vi`, `sh` oder `xdb`, die umfangreiche Aufgaben erledigen, gehören schwer verständliche Einträge, die voraussetzen, dass man wesentliche Züge des Kommandos bereits kennt.

Ohne Rechner bleibt das Bücherwissen trocken und abstrakt. Man sollte daher die Bücher in der Nähe eines Terminals lesen, sodass man sein Wissen sofort ausprobieren kann[4].

Zeitschriften berichten über Neuigkeiten. Manchmal bringen sie auch Kurse in Fortsetzungsform. Ein Lehrbuch oder ein Referenz-Handbuch ersetzen sie nicht. Sie eignen sich zur Ergänzung und Weiterbildung, sobald man über ein Grundwissen verfügt. Von einer guten Computerzeitschrift darf man verlangen, dass sie über Email erreichbar ist und ihre Informationen im Netz verfügbar macht. Falls sie sehr gut ist, berücksichtigt sie dabei auch sehschwache Leser.

Electronic Information besteht aus Mitteilungen in den Rechnernetzen. Das sind Bulletin Boards (Schwarze Bretter), Computerkonferenzen, Electronic Mail, Netnews, Veröffentlichungen per Anonymous FTP, Webseiten und ähnliche Dinge. Sie sind aktueller als Zeitschriften, die Diskussionsmöglichkeiten gehen weiter. Neben viel nutzlosem Zeug stehen hochwertige Beiträge von Fachleuten aus Universitäten und Computerfirmen. Ein guter Tipp sind die FAQ-Sammlungen (Frequently Asked Questions; Foire Aux Questions; Fragen, Antworten, Quellen der Erleuchtung) in den Netnews. Hauptproblem ist das Filtern der Informationsflut. Im Internet erscheinen täglich (!) mehrere 10.000 Beiträge, die Anzahl der Webseiten hat die Millionengrenze weit überschritten.

Es gibt **Lernprogramme** zu Hardware, Betriebssystemen und Anwendungsprogrammen. Man könnte meinen, dass sich gerade der Umgang mit dem Rechner mit Hilfe des Rechners lernen lässt. Moderne Rechner mit **Hypertext**[5], bewegter farbiger Grafik, Dialogfähigkeit und Tonausgabe bieten tatsächlich Möglichkeiten, die dem Buch verwehrt sind. Der Aufwand für ein

---

[4]Es hei3t, dass von der Information, die man durch Hören aufnimmt, nur 30 % im Gedächtnis haften bleiben. Beim Sehen sollen es 50 % sein, bei Sehen und Hören zusammen 70 %. Vollzieht man etwas eigenhändig nach – begreift man es im wörtlichen Sinne – ist der Anteil noch höher. Hingegen hat das maschinelle Kopieren von Informationen keine Wirkungen auf das Gedächtnis und kann nicht als Ersatz für die klassischen Wege des Lernens gelten.

[5]Hypertext ist ein Text, bei dem Sie erklärungsbedürftige Wörter anklicken und dann die Erklärung auf den Bildschirm bekommen. In Hypertext wäre diese Fußnote eine solche Erklärung. Der Begriff wurde Anfang der 60er Jahre von THEODOR HOLME (TED) NELSON in den USA geprägt. Siehe Abschnitt 2.7.14 *Hypertext* auf Seite 175. Mit dem Xanadu-Projekt hat er auch so etwas wie das World Wide Web vorweggenommen.

Lernprogramm, das diese Möglichkeiten ausnutzt, ist allerdings beträchtlich, und deshalb sind manche Lernprogramme nicht gerade ermunternd. Es gibt zwar Programme – sogenannte Autorensysteme (E: authoring system, F:) – die das Schreiben von Lernsoftware erleichtern, aber Arbeit bleibt es trotzdem. Auch gibt es vorläufig keinen befriedigenden Ersatz für Unterstreichungen und Randbemerkungen, mit denen eifrige Leser ihren Büchern eine persönliche Note geben. Erst recht ersetzt ein Programm nicht die Ausstrahlung eines guten Pädagogen.

Über den modernen Wegen der Wissensvermittlung hätte ich beinahe einen jahrzehntausendealten, aber immer noch aktuellen Weg vergessen: **Fragen**. Wenn Sie etwas wissen wollen oder nicht verstanden haben, fragen Sie, notfalls per Email. Die meisten Linux/UNIX-**Wizards** (*wizard*: person who effects seeming impossibilities; man skilled in occult arts; person who is permitted to do things forbidden to ordinary people) sind nette Menschen und freuen sich über Ihren Wissensdurst. Möglicherweise bekommen Sie verschiedene Antworten – es gibt in der Informatik auch Glaubensfragen – doch nur so kommen Sie voran.

Weiß auch Ihr Wizard nicht weiter, können Sie sich an die Öffentlichkeit wenden, das heißt an die schätzungsweise zehn Millionen Usenet-Teilnehmer. Den Weg dazu finden Sie unter dem Stichwort *Netnews*. Sie sollten allerdings vorher Ihre Handbücher gelesen haben und diesen Weg nicht bloß aus Bequemlichkeit wählen. Sonst erhalten Sie *RTFM*[6] als Antwort.

1.4 Wie läuft eine Sitzung ab?

Die Arbeit mit dem Rechner vollzieht sich meist im Sitzen vor einem Terminal und wird daher **Sitzung** (E: session, F: session) genannt. Mittels der Tastatur teilt der Benutzer dem Rechner seine Wünsche mit, auf dem Bildschirm antwortet die Maschine. Diese Arbeitsweise wird als **interaktiv** oder als **Terminal-Dialog** bezeichnet. Die Tastatur sieht ähnlich aus wie eine Schreibmaschinentastatur (weshalb Fähigkeiten im Maschinenschreiben nützlich sind), hat aber ein paar Tasten mehr. Oft gehört auch eine Maus (E: mouse, F: souris) oder eine Rollkugel (E: trackball, F: boule de commande) dazu.

Falls Sie mit einem Personal Computer arbeiten, müssen Sie ihn als erstes einschalten. Bei größeren Anlagen, an denen mehrere Leute gleichzeitig arbeiten, hat dies ein wichtiger und vielgeplagter Mensch für Sie erledigt, der **System-Verwalter**, Administrator oder System-Manager. Sie sollten seine Freundschaft suchen[7].

Nach dem Einschalten lädt der Rechner sein Betriebssystem, er bootet, wie man so sagt. **Booten** (E: to boot, F: lancer le système) heißt eigentlich Bootstrappen und das hinwiederum, sich an den eigenen Stiefelbändern

[6] Read The Fantastic Manual

[7] Laden Sie ihn gelegentlich zu Kaffee und Kuchen oder einem Viertele Wein ein.

oder Schnürsenkeln (bootstraps) aus dem Sumpf der Unwissenheit herausziehen wie weiland der Lügenbaron KARL FRIEDRICH HIERONYMUS FREIHERR VON MÜNCHHAUSEN an seinem Zopf[8]. Zu Beginn kann der Rechner nämlich noch nicht lesen, muss aber sein Betriebssystem vom Massenspeicher lesen, um lesen lernen zu können.

Ist dieser heikle Vorgang erfolgreich abgeschlossen, gibt der Rechner einen **Prompt** (E: prompt, F: invite) auf dem Bildschirm aus. Der Prompt ist ein Zeichen oder eine kurze Zeichengruppe – beispielsweise ein Pfeil, ein Dollarzeichen oder C geteilt durch größer als – die besagt, dass der Rechner auf Eingaben wartet. Der Prompt wird auch Systemanfrage, Bereitzeichen oder Eingabeaufforderung genannt.

Nun dürfen Sie in die Tasten greifen. Bei einem Mehrbenutzersystem erwartet der Rechner als erstes Ihre **Anmeldung**, das heißt die Eingabe des Benutzer-Namens (E: user name, F: nom d'utilisateur), unter dem Sie der Verwalter eingetragen hat. Man sagt auch, er habe Ihnen ein **Konto** (E: user account, F: compte d'accès) auf der Maschine eingerichtet. Als nächstes wird die Eingabe eines Passwortes verlangt. Das **Passwort** (auch Passphrase genannt, E: password, F: mot de passe) ist der Schlüssel zum Rechner. Es wird auf dem Bildschirm nicht angezeigt. Bei der Eingabe von Namen und Passwort sind oft keine Korrekturen zugelassen, Groß- und Kleinschreibung wird unterschieden. War Ihre Anmeldung in Ordnung, heißt der Rechner Sie herzlich willkommen und promptet wieder. Die Arbeit beginnt. Auf einem DOS-PC geben Sie beispielsweise `dir` ein, auf einer Linux/UNIX-Anlage `ls`. Jede Eingabe wird mit der **Eingabe-Taste** (auch mit Return, Enter, CR oder einem geknickten Pfeil nach links bezeichnet; E: return key, F: touche de retour) abgeschlossen[9].

Zum Eingewöhnen führen wir eine kleine Sitzung durch, möglichst im Beisein eines Systemkundigen. Suchen Sie sich ein freies Linux/UNIX-Terminal. Auf die Aufforderung zur Anmeldung (`login:`) tippen Sie Ihren Benutzernamen ein, Eingabe-Taste nicht vergessen, dann Ihr Passwort. Nach dem Willkommensgruß des Systems geben wir folgende Linux/UNIX-Kommandos ein (Eingabe-Taste!) und versuchen, ihre Bedeutung mithilfe der Online-Referenz (`man`-Kommandos) näherungsweise zu verstehen:

```
who
man who
date
man date
pwd
```

[8]Siehe GOTTFRIED AUGUST BÜRGER, Wunderbare Reisen zu Wasser und zu Lande, Feldzüge und lustige Abenteuer des Freiherrn von Münchhausen, wie er dieselben bei der Flasche im Zirkel seiner Freunde selbst zu erzählen pflegt. Insel Taschenbuch 207, Insel Verlag Frankfurt (Main) (1976), im 4. Kapitel

[9]Manche Systeme unterscheiden zwischen Return- und Enter-Taste, rien n'est simple. Auf Tastaturen für den kirchlichen Gebrauch trägt die Taste die Bezeichnung *Amen*.

```
man pwd
ls
ls -l /bin
man ls
exit
```

Falls auf dem Bildschirm links unten das Wort `more` erscheint, betätigen Sie die Zwischenraum-Taste (E: space bar, F:). `more` ist ein Pager, ein Programm, das einen Text seiten- oder bildschirmweise ausgibt.

Die Grundform eines **Linux/UNIX-Kommandos** ist:

```
Kommando -Optionen Argumente
```

Statt **Option** findet man auch die Bezeichnung Parameter, Flag oder Schalter. Eine Option modifiziert die Wirkungsweise des Kommandos, beispielsweise wird die Ausgabe des Kommandos `ls` ausführlicher, wenn wir die Option `-l` (long) dazuschreiben. **Argumente** sind Datei-Namen oder andere Informationen, die das Kommando benötigt, oben der Verzeichnisname `/bin`. Bei den Namen der Linux/UNIX-Kommandos haben sich ihre Schöpfer etwas gedacht, nur was, bleibt hin und wieder im Dunkeln. Hinter manchen Namen steckt auch eine ganze Geschichte, wie man sie in der Newsgruppe `comp.society.folklore` im Netz erfährt. Das Kommando `exit` beendet die Sitzung. Es ist ein internes Shell-Kommando und im Handbuch unter der Beschreibung der Shell `sh` zu finden.

Jede Sitzung muss ordnungsgemäß beendet werden. Es reicht nicht, sich einfach vom Stuhl zu erheben. Laufende Programme – zum Beispiel ein Editor – müssen zu Ende gebracht werden, auf einer Mehrbenutzeranlage meldet man sich mit einem Kommando ab, das `exit`, `quit`, `logoff`, `logout`, `stop`, `bye` oder `end` lautet. Arbeiten Sie mit Fenstern, so findet sich irgendwo am Rand das Bild eines Knopfes (E: button, F: bouton) namens *exit*. Ihren eigenen PC dürfen Sie selbst herunterfahren und ausschalten, ansonsten erledigt das wieder der Verwalter. Das Ausschalten des Terminals einer Mehrbenutzeranlage hat für den Rechner keine Bedeutung, die Sitzung läuft weiter!

Stundenlanges Arbeiten am Bildschirm belastet die Augen, stundenlanges Bücherlesen oder Autofahren genauso. Eine gute Information zu diesem Thema findet sich in der Universität Gießen unter dem URL:

```
www.uni-giessen.de/~gkw1/patient/arbeitsplatz.html
```

1.5 Wo schlägt man nach?

Wenn es um Einzelheiten geht, ist das zu jedem Linux/UNIX-System gehörende und einheitlich aufgebaute **Referenz-Handbuch** – auf Papier, CD oder Bildschirm – die wichtigste Hilfe[10]. Es gliedert sich in folgende **Sektionen**:

- 1 Kommandos und Anwendungsprogramme

[10] Real programmers don't read manuals, sagt das Netz.

- 1M Kommandos zur Systemverwaltung (maintenance)
- 2 Systemaufrufe
- 3C Subroutinen der Standard-C-Bibliothek
- 3M Mathematische Bibliothek
- 3S Subroutinen der Standard-I/O-Bibliothek
- 3X Besondere Bibliotheken
- 4 Datei-Formate oder Geräte-Dateien
- 5 Vermischtes (z. B. Datei-Hierarchie, Zeichensätze) oder Datei-Formate
- 6 Spiele
- 7 Gerätefiles oder Makros
- 8 Systemverwaltung
- 9 Glossar oder Kernroutinen

Subroutinen sind in diesem Zusammenhang vorgefertigte Funktionen für eigene Programme, Standardfunktionen oder Unterprogramme mit anderen Worten. Die erste Seite jeder Sektion ist mit `intro` betitelt und führt in den Inhalt der Sektion ein. Beim Erwähnen eines Kommandos wird gelegentlich die Sektion des Handbuchs in Klammern angegeben, da das gleiche Stichwort in mehreren Sektionen mit unterschiedlicher Bedeutung vorkommen kann, beispielsweise `cpio(1)` und `cpio(4)`. Die Einordnung eines Stichwortes in eine Sektion variiert etwas zwischen verschiedenen Linux/UNIX-Abfüllungen. Die Eintragungen zu den Kommandos oder Stichwörtern sind wieder gleich aufgebaut:

- Name (Name des Kommandos)
- Synopsis, Syntax (Gebrauch des Kommandos)
- Remarks (Anmerkungen)
- Description (Beschreibung des Kommandos)
- Return Value (Rückgabewert des Programms)
- Examples (Beispiele)
- Hardware Dependencies (hardwareabhängige Eigenheiten)
- Author (Urheber des Kommandos)
- Files (vom Kommando betroffene Dateien)
- See Also (ähnliche oder verwandte Kommandos)
- Diagnostics (Fehlermeldungen)
- Bugs (Mängel, soweit bekannt)
- Caveats, Warnings (Warnungen)
- International Support (europäische Absonderlichkeiten)

Bei vielen Kommandos finden sich nur Name, Synopsis und Description. Zu einigen kommt eine Beschreibung mit, die ausgedruckt mehr als hundert Seiten A4 umfasst. Der Zweck des Kommandos wird meist verheimlicht; deshalb versuche ich, diesen Punkt zu erhellen. Was hilft die Beschreibung eines Schweißbrenners, wenn Sie nicht wissen, was und warum man schweißt?

Einige Kommandos oder Standardfunktionen haben keinen eigenen Eintrag, sondern sind mit anderen zusammengefasst. So findet man das Kommando `mv` unter der Eintragung für das Kommando `cp` oder die Standardfunktion `gmtime` bei der Standardfunktion `ctime`. In solchen Fällen muss man das Sachregister, den Index des Handbuchs befragen. Auf manchen Systemen findet sich auch ein Kommando `apropos`, das in den man-Seiten nach Schlüsselwörtern sucht.

Mittels des Kommandos `man` holt man die Einträge aus dem gespeicherten Referenz-Handbuch (On-line-Manual, man-Seiten, man-pages) auf den Bildschirm oder Drucker. Das On-line-Manual sollte zu den auf dem System vorhandenen Kommandos passen, während das papierne Handbuch veraltet oder verschwunden sein kann. Versuchen Sie folgende Eingaben:

```
man pwd
man time
man 2 time
man -k time
man man
man man | col -b > manual.txt
man man | col -b | lp
```

Die Zahlenangabe bei der dritten Eingabe bezieht sich auf die Sektion. Mit der vierten Zeile erfährt man möglicherweise etwas zum Schlüsselwort *time*. Falls nicht, weisen Sie Ihren Verwalter auf das Kommando `catman` hin. Die letzten beiden Eingabezeilen geben die Handbuchseiten zum Kommando `man` in eine Datei oder auf den Default-Drucker aus (fragen Sie Ihren Arzt oder Apotheker oder besser Ihren Verwalter, für das Drucken gibt es viele Wege). Drucken Sie aber nicht das ganze Handbuch aus, die meisten Seiten braucht man nie.

1.6 Warum verwendet man Computer (nicht)?

Philosophische Interessen sind bei Ingenieuren häufig eine Alterserscheinung, meint der Wiener Computerpionier HEINZ ZEMANEK. Ich glaube, das nötige Alter zu haben, um dann und wann das Wort *warum* in den Mund nehmen oder in die Tastatur hacken zu dürfen. Junge Informatiker äußern diese Frage auch gern. Bei der Umstellung einer hergebrachten Tätigkeit auf Computer steht oft die **Zeitersparnis** (= Kostenersparnis) im Vordergrund. Zumindest wird sie als Begründung für die Umstellung herangezogen. Das ist weitgehend falsch. Während der Umstellung muss doppelgleisig gearbeitet werden, und hernach erfordert das Rechnersystem eine ständige Pflege.

Einige Arbeiten gehen mit Rechnerhilfe schneller von der Hand, dafür verursacht der Rechner selbst Arbeit. Auf Dauer sollte ein Gewinn herauskommen, aber die Erwartungen sind oft überzogen.

Nach drei bis zehn Jahren Betrieb ist ein Rechner veraltet. Die weitere Benutzung ist unwirtschaftlich, das heißt man könnte mit dem bisherigen Aufwand an Zeit und Geld eine leistungsfähigere Anlage betreiben oder mit einer neuen Anlage den Aufwand verringern. Dann stellt sich die Frage, wie die alten Daten weiterhin verfügbar gehalten werden können. Denken Sie an die Lochkartenstapel oder Disketten verflossener Jahrzehnte, die heute nicht mehr lesbar sind, weil es die Maschinen nicht länger gibt. Oft muss man auch mit der Anlage die Programme wechseln. Der Übergang zu einem neuen System ist von Zeit zu Zeit unausweichlich, wird aber von Technikern und Kaufleuten gleichermaßen gefürchtet. Auch dieser Aufwand ist zu berücksichtigen. Mit Papier und Tinte war das einfacher; einen Brief unserer Urgroßeltern können wir heute noch lesen.

Deutlicher als der Zeitgewinn ist der **Qualitätsgewinn** der Arbeitsergebnisse. In einer Buchhaltung sind dank der Unterstützung durch Rechner die Auswertungen aktueller und differenzierter als früher. Informationen – zum Beispiel aus Einkauf und Verkauf – lassen sich schneller, sicherer und einfacher miteinander verknüpfen als auf dem Papierweg. Manuskripte lassen sich bequemer ändern und besser formatieren als zu Zeiten der mechanischen Schreibmaschine. Von technischen Zeichnungen lassen sich mit minimalem Aufwand Varianten herstellen. Mit Simulationsprogrammen können Entwürfe getestet werden, ehe man an echte und kostspielige Versuche geht. Literaturrecherchen decken heute eine weit größere Menge von Veröffentlichungen ab als vor vierzig Jahren. Gro'se Datenmengen waren früher gar nicht oder nur mit Einschränkungen zu bewältigen. Solche Aufgaben kommen beim Suchen oder Sortieren sowie bei der numerischen Behandlung von Problemen aus der Wettervorhersage, der Strömungslehre, der Berechnung von Flugbahnen oder Verbrennungsvorgängen vor. Das Durchsuchen umfangreicher Datensammlungen ist eine Lieblingsbeschäftigung der Rechner.

Noch eine Warnung. Die Arbeit wird durch Rechner nur selten einfacher. Mit einem Bleistift können die meisten umgehen. Die Benutzung eines Texteditors erfordert eine **Einarbeitung**, die Ausnutzung aller Möglichkeiten eines leistungsfähigen Textsystems eine lange Vorbereitung und ständige **Weiterbildung**. Ein Schriftstück wie das vorliegende wäre vor fünfzig Jahren nicht am Schreibtisch herzustellen gewesen; heute ist das mit Rechnerhilfe kein Hexenwerk, setzt aber eine eingehende Beschäftigung mit mehreren Programmen (Editor, LaTeX, RCS, `make`, `dvips`, *xdvi*, `xfig` und eine Handvoll kleinerer Linux/UNIX-Werkzeuge) und Fragen zur Gestaltung von Schriftwerken voraus.

Man darf nicht vergessen, dass der Rechner ein Werkzeug ist. Er bereitet Daten auf, interpretiert sie aber nicht. Er übernimmt keine **Verantwortung** und handelt nicht nach ethischen Grundsätzen. Er rechnet, aber wertet nicht. Das ist keine technische Unvollkommenheit, sondern eine grundsätzliche Eigenschaft. Die Fähigkeit zur Verantwortung setzt die **Willensfreiheit** voraus und diese beinhaltet den eigenen Willen. Ein Rechner, der anfängt, einen eigenen Willen zu entwickeln, ist ein Fall für die Werkstatt.

Der Rechner soll den Menschen ebensowenig ersetzen wie ein Hammer die Hand ersetzt, sondern ihn ergänzen. Das hört sich banal an, aber manchmal ist die Aufgabenverteilung zwischen Mensch und Rechner schwierig zu erkennen. Es ist bequem, die Entscheidung samt der Verantwortung der Maschine zuzuschieben. Es gibt auch Aufgaben, bei denen der Rechner einen Menschen ersetzen kann – wenn nicht heute, dann künftig – aber dennoch nicht soll. Nehmen wir zwei Extremfälle. Rufe ich die Telefonnummer 0721/19429 an, so antwortet ein Automat und teilt mir den Pegelstand des Rheins bei Karlsruhe mit. Das ist ok, denn ich will nur die Information bekommen. Ruft man dagegen die Telefonseelsorge an, erwartet man, dass ein Mensch zuhört, wobei das Zuhören wichtiger ist als das Übermitteln einer Information. So klar liegen die Verhältnisse nicht immer. Wie sieht es mit dem Rechner als Lehrer aus? Darf ein Rechner Studenten prüfen? Soll ein Arzt eine Diagnose vom Rechner stellen lassen? Ist ein Rechner zuverlässiger als ein Mensch? Ist die Künstliche Intelligenz in allen Fällen der Natürlichen Dummheit überlegen? Soll man die Entscheidung über Krieg und Frieden dem Präsidenten der USA überlassen oder besser seinem Rechner? Und wenn der Präsident zwar entscheidet, sich aber auf die Auskünfte seines Rechners verlässt? Wer ist dann wichtiger, der Präsident oder sein Rechner?

Je besser die Rechner funktionieren, desto mehr neigen wir dazu, die Datenwelt für maßgebend zu halten und Abweichungen der realen Welt von der Datenwelt für Störungen. Hört sich übertrieben an, ist es auch, aber wie lange noch? Fachliteratur, die nicht in einer Datenbank gespeichert ist, zählt praktisch nicht mehr. Texte, die sich nicht per Rechner in andere Sprachen übersetzen lassen, gelten als mangelhaft. Bei Meinungsverschiedenheiten über personenbezogene Daten hat zunächst einmal der Rechner recht, und wenn er Briefe an *Herrn Marianne Meier* schreibt. Das lässt sich klären, aber wie sieht es mit dem **Weltbild** aus, das die Computerspiele unseren Kindern vermitteln? Welche Welt ist wirklich? Kann man von Spielgeld leben? Haben die Mitmenschen ein so einfaches Gemüt wie die virtuellen Helden? War *Der längste Tag* nur ein Bildschirmspektakel? Brauchten wir 1945 nur neu zu booten?

Unbehagen bereitet auch manchmal die zunehmende **Abhängigkeit** vom Rechner, die bei Störfällen unmittelbar zu spüren ist – sei es, dass der Rechner streikt oder dass der Strom ausfällt. Da gibt es Augenblicke, in denen sich die Verwalter fragen, warum sie nicht Minnesänger oder Leuchtturmwärter geworden sind. Nun, der Mensch war immer abhängig. In der Steinzeit davon, dass es genügend viele nicht zu starke Bären gab, später davon, dass das Wetter die Ernte begünstigte, und heute sind wir auf die Rechner angewiesen. Im Unterschied zu früher – als der erfahrene Bärenjäger die Bärenlage überblickte – hat heute der Einzelne nur ein unbestimmtes Gefühl der Abhängigkeit von Dingen, die er nicht kennt und nicht beeinflussen kann.

Mit den Rechnern wird es uns vermutlich ähnlich ergehen wie mit der Elektrizität: wir werden uns daran gewöhnen. Wie man für Stromausfälle eine Petroleumlampe und einen Campingkocher bereithält, sollte man für

Rechnerausfälle etwas Papier, einen Bleistift und ein gutes, zum Umblättern geeignetes Buch zurücklegen.

1.7 Begriffe

Folgende Begriffe sollten klarer geworden sein:

- Anwendungsprogramm
- Benutzer, System-Verwalter
- Betriebssystem
- binäre Darstellung
- Bit, Byte, Oktett
- booten
- Dualsystem, Oktalsystem, Hexadezimalsystem
- Hardware, Software
- Informatik
- Kommando, Eingabe-Taste
- Konto, Benutzername, Passwort
- Prompt
- Prozessor, CPU, Arbeitsspeicher, Massenspeicher
- Programmiersprache
- Rechner (Computer)
- Referenz-Handbuch, man-Seiten, `man`
- Sitzung, anmelden, abmelden
- Speichern, Übermitteln und Verändern von Daten
- Terminal, Bildschirm, Tastatur
- Linux/UNIX-Kommando, Option, Argument

Folgende Linux/UNIX-Kommandos sollten bekannt sein:

- `exit`
- `man`
- `who`
- `date`
- `pwd`
- `ls`

1.8 Memo

(folgt *as soon as possible*)

1.9 Fragen

(folgen *as soon as possible*)

Seid einig, einig, einig!
Schiller, Tell

2 Linux/UNIX

2.1 Grundbegriffe

2.1.1 Wozu braucht man ein Betriebssystem?

In der frühen Kindheit der Computer – schätzungsweise vor 1950 – hatten die Maschinen kein Betriebssystem. Die damaligen Computer waren jedoch trotz ihrer gewaltigen räumlichen Abmessungen logisch sehr übersichtlich, die wenigen Benutzer kannten sozusagen jedes Bit persönlich. Beim Programmieren mußte man sich auch um jedes Bit einzeln kümmern. Wollte man etwas auf der Fernschreibmaschine (so hieß das I/O-Subsystem damals) ausgeben, so schob man Bit für Bit über die Treiberstufen zu den Elektromagneten. In heutiger Sprechweise enthielt jedes Anwendungsprogramm sein eigenes Betriebssystem.

Die Programmierer waren damals schon so arbeitsscheu (effektivitätsbewußt) wie heute und bemerkten bald, daß dieses Vorgehen nicht zweckmäßig war. Viele Programmteile wiederholten sich in jeder Anwendung. Man faßte diese Teile auf einem besonderen Lochkartenstapel oder Lochstreifen zusammen, der als **Vorspann** zu jeder Anwendung eingelesen wurde. Der nächste Schritt war, den Vorspann nur noch nach dem Einschalten der Maschine einzulesen und im Speicher zu belassen. Damit war das Betriebssystem geboren und die Trennung von den Anwendungen vollzogen.

Heutige Computer sind räumlich nicht mehr so eindrucksvoll, aber logisch um Größenordnungen komplexer. Man faßt viele Einzelheiten zu übergeordneten Objekten zusammen, man abstrahiert in mehreren Stufen. Der Benutzer sieht nur die oberste Schicht der Software, die ihrerseits mit darunterliegenden Software-Schichten verkehrt. Zuunterst liegt die Hardware. Ein solches **Schichtenmodell** finden wir bei den Netzen wieder. In Wirklichkeit sind die Schichten nicht sauber getrennt, sondern verzahnt, teils aus historischen Gründen, teils wegen Effektivität, teils aus Schlamperei. Neben dem Schichtenmodell werden **objektorientierte Ansätze** verfolgt, in denen alle harten und weichen Einheiten abgekapselte Objekte sind, die über Nachrichten miteinander verkehren. Aber auch hier bildet sich eine Hierarchie aus.

Was muß ein Betriebssystem als Minimum enthalten? Nach obigem das, was alle Anwendungen gleichermaßen benötigen. Das sind die Verbindungen zur Hardware (CPU, Speicher, I/O) und die Verwaltung von Prozessen und Daten. Es gibt jedoch Bestrebungen, auch diese Aufgaben in Anwendungsprogramme zu verlagern und dem Betriebssystem nur noch koordinierende und kontrollierende Tätigkeiten zu überlassen. Vorteile eines solchen **Mikro-Kerns** sind Übersichtlichkeit und Anpassungsfähigkeit.

Die Grundfunktionen eines Rechners lassen sich nach dem Gesagten durch folgende Mittel verwirklichen, wobei sich Geschwindigkeit und Anpassungsfähigkeit widersprechen:

- Hardware (schnell, unflexibel)
- Software
  - Betriebssystem-Kern
  - Systemmodule und Systemaufrufe des Betriebssystems
  - Standardfunktionen einer Programmiersprache, Funktionen höherer Bibliotheken

Wo die Grenzen gezogen werden, steht dem Entwickler in Grenzen frei.

Wenn ein UNIX-Programmierer heute Daten nach `stdout` schreibt, setzt er mehrere Megabyte System-Software in Bewegung, die andere für ihn erstellt haben. Als Programmierer dürfte man nur noch im pluralis modestatis reden.

2.1.2 Verwaltung der Betriebsmittel

Ein Betriebssystem vermittelt zwischen der Hardware und den Benutzern. Aus Benutzersicht verdeckt es den mühsamen und schwierigen unmittelbaren Verkehr mit der Hardware. Der Benutzer braucht sich nicht darum zu sorgen, daß zu bestimmten Zeiten bestimmte elektrische Impulse auf bestimmten Leitungen ankommen, er gibt vielmehr nur das Kommando zum Lesen aus einer Datei namens `xyz`. Für den Benutzer stellen Hardware plus Betriebssystem eine **virtuelle Maschine** mit einem im Handbuch beschriebenen Verhalten dar. Was auf der Hardware wirklich abläuft, interessiert nur den Entwicklungsingenieur. Daraus folgt, daß dieselbe Hardware mit einem anderen Betriebssystem eine andere virtuelle Maschine bildet. Ein PC mit PC-DOS ist ein PC-DOS-Rechner, derselbe PC mit Linux ist ein UNIX-Rechner mit deutlich anderen Eigenschaften. Im Schichtenmodell stellt jede Schicht eine virtuelle Maschine für ihren oberen Nachbarn dar, die oberste Schicht die virtuelle Maschine für den Benutzer.

Aus der Sicht der Hardware sorgt das Betriebssystem dafür, daß die einzelnen **Betriebsmittel** (Prozessor, Speicher, Ports für Ein- und Ausgabe) den Benutzern bzw. deren Programmen in einer geordneten Weise zur Verfügung gestellt werden, so daß sie sich nicht stören. Die Programme dürfen also nicht selbst auf die Hardware zugreifen, sondern haben ihre Wünsche dem Betriebssystem mitzuteilen, das sie möglichst sicher und zweckmäßig weiterleitet[1]

[1]Ein Nachteil von PC-DOS ist, daß ein Programmierer direkt die Hardware ansprechen kann und sich so um das Betriebssystem herummogelt.

Neben den harten, körperlich vorhandenen Betriebsmitteln kann man auch Software als Betriebsmittel ansehen. Für den Benutzer macht es unter UNIX keinen Unterschied, ob er einen Text auf einen Massenspeicher schreibt oder dem Electronic Mail System übergibt, das aus ein paar Drähten und viel Software besteht. Schließlich gibt es virtuelle Betriebsmittel, die für den Benutzer oder seinen Prozess scheinbar vorhanden sind, in Wirklichkeit aber durch Hard- und Software vorgegaukelt werden. Beipielsweise wird unter UNIX der immer zu kleine Arbeitsspeicher scheinbar vergrößert, indem man Massenspeicher zu Hilfe nimmt. Dazu gleich mehr. Auch zwischen harten und virtuellen Druckern sind vielfältige Beziehungen herstellbar. Der Zweck dieser Scheinwelt[2] ist, den Benutzer von den Beschränkungen der harten Welt zu befreien. Die Kosten dafür sind eine erhöhte Komplexität des Betriebssystems und Zeit. Reichlich reale Betriebsmittel sind immer noch das Beste.

An fast allen Aktivitäten des Computers ist der zentrale Prozessor beteiligt. Ein Prozessor erledigt zu einem Zeitpunkt immer nur einen Auftrag. Der Verteilung der **Prozessorzeit** kommt daher eine besondere Bedeutung zu. Wenn in einem leistungsfähigen Betriebssystem wie UNIX mehrere Programme (genauer: Prozesse) gleichzeitig Prozessorzeit verlangen, teilt das Betriebssystem jedem nacheinander eine kurze Zeitspanne zu, die nicht immer ausreicht, den jeweiligen Prozess zu Ende zu bringen. Ist die Zeitspanne (im Millisekundenbereich) abgelaufen, beendet das Betriebssystem den Prozess vorläufig und reiht ihn wieder in die Warteschlange ein. Nach Bedienung aller anstehenden Prozesse beginnt das Betriebssystem wieder beim ersten, so daß bei den Benutzern der Eindruck mehrerer gleichzeitig laufender Prozesse entsteht. Man spricht von Quasi- oder Pseudo-Parallelität. Dieser Vorgang läßt sich durch eine gleichmäßig rotierende **Zeitscheibe** veranschaulichen, von der jeder Prozess einen Sektor bekommt. Die Sektoren brauchen nicht gleich groß zu sein. Diese Form der Auftragsabwicklung wird **präemptives** oder **verdrängendes Multi-Tasking** genannt (lat. *praeemere* = durch Vorkaufsrecht erwerben). Das Betriebssystem hat sozusagen ein Vorkaufsrecht auf die Prozessorzeit und verdrängt andere Prozesse nach Erreichen eines Zeitlimits.

Einfachere Betriebssysteme (MacOS bis Version 9, Microsoft Windows bis Version 98) verwalten zwar auch eine Warteschlange von Prozessen, vollenden aber einen Auftrag, ehe der nächste an die Reihe kommt. Die Prozesse können sich **kooperativ** zeigen und den Platz an der Sonne freiwillig räumen, um ihren Mitbewerbern eine Chance zu geben; das Betriebssystem erzwingt dies jedoch nicht. Versucht ein nicht-kooperativer Prozess, die größte Primzahl zu berechnen, warten die Mitbenutzer lange. Noch einfachere Betriebssysteme (PC-DOS) richten nicht einmal eine Warteschlange ein.

Den Algorithmus zur Verteilung der Prozessorzeit (scheduling algorithm)

[2]In UNIX kann ein Benutzer, den es nicht gibt (ein Dämon), eine Datei, die es nicht gibt (eine Datenbank-View), auf einem Drucker, den es nicht gibt (ein logischer Drucker), ausgeben, und es kommt am Ende ein reales Blatt Papier heraus.

kann man verfeinern. So gibt es Prozesse, die wenig Zeit beanspruchen, diese aber sofort haben möchten (Terminaldialog), andere brauchen mehr Zeit, aber nicht sofort (Hintergrundprozesse). Ein Prozess, das auf andere Aktionen warten muß, zum Beispiel auf die Eingabe von Daten, sollte vorübergehend aus der Verteilung ausscheiden. Man muß sich vor Augen halten, daß die Prozessoren heute mit hundert Millionen Takten und mehr pro Sekunde arbeiten. Mit einem einzelnen Bildschirmdialog langweilt sich schon ein Prozessor für zwo fuffzich.

Das Programm, das der Prozessor gerade abarbeitet, muß sich im Arbeitsspeicher befinden. Wenn der Prozessor mehrere Programme gleichzeitig in Arbeit hat, sollten sie auch gleichzeitig im Arbeitsspeicher liegen, denn ein ständiges Ein- und Auslagern vom bzw. zum Massenspeicher kostet Zeit. Nun sind die Arbeitsspeicher selten so groß, daß sie bei starkem Andrang alle Programme fassen, also kommt man um das Auslagern doch nicht ganz herum. Das Auslagern des momentan am wenigsten dringend benötigten Programms als Ganzes wird als **Swapping** oder Speicheraustauschverfahren bezeichnet. Programm samt momentanen Daten kommen auf die Swapping Area (Swap-Datei) des Massenspeichers (Platte). Dieser sollte möglichst schnell sein, Swappen auf Band ist der allerletzte Ausweg. Bei Bedarf werden Programm und Daten in den Arbeitsspeicher zurückgeholt. Ein einzelnes Programm mit seinen Daten darf nicht größer sein als der verfügbare Arbeitsspeicher.

Bei einer anderen Technik werden Programme und Daten in Seiten (pages) unterteilt und nur die augenblicklich benötigten Seiten im Arbeitsspeicher gehalten. Die übrigen Seiten liegen auf dem Massenspeicher auf Abruf. Hier darf eine Seite nicht größer sein als der verfügbare Arbeitsspeicher. Da ein Programm aus vielen Seiten bestehen kann, darf seine Größe die des Arbeitsspeichers erheblich übersteigen. Dieses **Paging** oder Seitensteuerungsverfahren hat also Vorteile gegenüber dem Swapping.

Bei starkem Andrang kommt es vor, daß der Prozessor mehr mit Aus- und Einlagern beschäftigt ist als mit nutzbringender Arbeit. Dieses sogenannte **Seitenflattern** (trashing) muß durch eine zweckmäßige Konfiguration (Verlängerung der einem Prozess minimal zur Verfügung stehenden Zeit) oder eine Vergrößerung des Arbeitsspeichers verhindert werden. Auch ein Swapping oder Paging übers Netz ist durch ausreichend Arbeitsspeicher oder lokalen Massenspeicher zu vermeiden, da es viel Zeit kostet und das Netz belastet.

2.1.3 Verwaltung der Daten

Die Verwaltung der Daten des Systems und der Benutzer in einem **Datei-System** ist die zweite Aufgabe des Betriebssystems. Auch hier schirmt das Betriebssystem den Benutzer vor dem unmittelbaren Verkehr mit der Hardware ab. Wie die Daten physikalisch auf den Massenspeichern abgelegt sind, interessiert ihn nicht, sondern nur die logische Organisation, beispielsweise in einem Baum von Verzeichnissen. Für den Benutzer ist eine Datei eine zusammengehörige Menge von Daten, die er über den Dateinamen anspricht. Daß die Daten einer Datei physikalisch über mehrere, nicht zusammenhängende Bereiche auf der Festplatte verstreut sein können, geht nur das Betriebssystem etwas an. Eine Datei kann sogar über mehrere Platten, unter Umständen auf mehrere Computer verteilt sein. Im schlimmsten Fall existiert die Datei, mit dem der Benutzer zu arbeiten wähnt, überhaupt nicht, sondern wird aus Teilen verschiedener Dateien bei Bedarf zusammengesetzt. Beim Arbeiten mit Datenbanken kommt das vor. Zum Benutzer hin sehen alle UNIX-Datei-Systeme gleich aus, zur Hardware hin gibt es jedoch Unterschiede. Einzelheiten siehe im Referenz-Handbuch unter `fs(4)`.

2.1.4 Einteilung der Betriebssysteme

Nach ihrem Zeitverhalten werden Betriebssysteme eingeteilt in:

- Batch-Systeme
- Dialog-Systeme
- Echtzeit-Systeme

wobei gemischte Formen die Regel sind.

In einem Stapel- oder **Batch-System** werden die Aufträge (Jobs) in eine externe Warteschlange eingereiht und unter Beachtung von Prioritäten und weiteren, der Effizienz und Gerechtigkeit dienenden Gesichtspunkten abgearbeitet, ein Auftrag nach dem anderen. Einige Tage später holt der Benutzer seine Ergebnisse ab. Diese Arbeitsweise war früher – vor UNIX – die einzige und ist heute noch auf Großrechenanlagen verbreitet. Zur Programmentwicklung mit wiederholten Testläufen und Fehlerkorrekturen ist sie praktisch nicht zu gebrauchen.

Bei einem **Dialog-System** arbeitet der Benutzer an einem Terminal in unmittelbarem Kontakt mit der Maschine. Die Reaktionen auf Tastatur-Eingaben erfolgen nach menschlichen Maßstäben sofort, nur bei Überlastung der Anlage kommen sie zäher. Alle in die Maschine eingegebenen Aufträge sind sofort aktiv und konkurrieren um Prozessorzeit und die weiteren Betriebsmittel, die nach ausgeklügelten Gesichtspunkten zugewiesen werden. Es gibt keine externe Warteschlange für die Aufträge. UNIX ist in erster Linie ein Dialogsystem.

In einem **Echtzeit-System** bestimmt der Programmierer oder System-Manager das Zeitverhalten völlig. Für kritische Programmteile wird eine maximale Ausführungsdauer garantiert. Das Zeitverhalten ist unter allen Umständen vorhersagbar. UNIX ist infolge der Pufferung der Datenströme zunächst kein Echtzeit-System. Es gibt aber Erweiterungen, die UNIX für Echtzeit-Aufgaben geeignet machen, siehe Abschnitt 2.12 *Echtzeit-Erweiterungen* auf Seite 241.

Nach der Anzahl der scheinbar gleichzeitig bearbeiteten Aufträge – wir haben darüber schon gesprochen – unterscheidet man:

- Single-Tasking-Systeme
- Multi-Tasking-Systeme
  - kooperative Multi-Tasking-Systeme

  – präemptive Multi-Tasking-Systeme

Nach der Anzahl der gleichzeitig angemeldeten Benutzer findet man eine Einteilung in:

- Single-User-Systeme
- Multi-User-Systeme

Ein Multi-User-System ist praktisch immer zugleich ein Multi-Tasking-System, andernfalls könnten sich die Benutzer nur gemeinsam derselben Aufgabe widmen. Das ist denkbar, uns aber noch nie über den Weg gelaufen (Teilhaber- oder Transaktionsbetrieb, soll bei Buchungs- oder Auskunftssyztemen vorkommen). Ein Multi-User-System enthält vor allem Vorrichtungen, die verhindern, daß sich die Benutzer in die Quere kommen (Benutzerkonten, Zugriffsrechte an Dateien).

Schließlich gibt es, bedingt durch den Wunsch nach immer mehr Rechenleistung, die Vernetzung und die Entwicklung von Computern mit mehreren Zentralprozessoren, seit einigen Jahren:

- Einprozessor-Systeme
- Mehrprozessor-Systeme

Ein Sonderfall der Mehrprozessor-Systeme sind **Netz-Betriebssysteme**, die mehrere über ein Netz verteilte Prozessoren wie einen einzigen Computer verwalten, im Gegensatz zu Netzen aus selbständigen Computern mit jeweils einer eigenen Kopie eines Betriebssystems, das Netzfunktionen unterstützt.

Stellt man die Einteilungen in einem vierdimensionalen Koordinatensystem dar, so besetzen die wirklichen Systeme längst nicht jeden Schnittpunkt, außerdem gibt es Übergangsformen. PC-DOS ist ein Dialogsystem mit Single-Tasking-Fähigkeiten für einen einzelnen Prozessor und einen einzelnen Benutzer. IBM-OS/2 ist ein Dialogsystem mit Multi-Tasking-Fähigkeiten, ebenfalls für einen einzelnen Prozessor und einen einzelnen Benutzer. UNIX ist ein Dialogsystem mit Multi-Tasking-Fähigkeiten für mehrere Benutzer und verschiedene Prozessorentypen, in jüngerer Zeit erweitert um Echtzeit-Funktionen und Unterstützung mehrerer paralleler Prozessoren. Das Betriebssystem Hewlett-Packard RTE VI/VM für die Maschinen der HP 1000-Reihe war ein echtes Echtzeit-System mit einer einfachen Batch-Verwaltung. Die IBM 3090 lief unter dem Betriebssystem MVS mit Dialog- und Batch-Betrieb (TSO bzw. Job Control Language). Novell NetWare ist ein Netz-Betriebssystem, das auf vernetzten PCs anstelle von PC-DOS oder OS/2 läuft, wohingegen das Internet aus selbständigen Computern unter verschiedenen Betriebssystemen besteht.

Um einen Brief zu schreiben oder die Primzahlen bis 100000 auszurechnen, reicht PC-DOS. Soll daneben ein Fax-Programm sende- und empfangsbereit sein und vielleicht noch die Mitgliederliste eines Vereins sortiert werden, braucht man IBM OS/2 oder Microsoft Windows NT. Wollen mehrere Benutzer gleichzeitig auf dem System arbeiten, muß es UNIX sein. Arbeitet man in internationalen Netzen, ist UNIX der Standard. UNIX läuft zur Not auf einem einfachen PC mit Disketten, aber für das, was man heute von UNIX verlangt, ist ein PC mit einem Intel 80386, 8 MB Arbeitsspeicher und einer 200-MB-Festplatte die untere Grenze.

2.1.5 Laden des Betriebssystems

Ein Betriebssystem wie PC-DOS oder UNIX wird auf Bändern, CD-ROMs, Disketten oder über das Netz geliefert. Die Installation auf den Massenspeicher gehört zu den Aufgaben des System-Managers und wird im Abschnitt 2.14 *Systemverwaltung* auf Seite 253 beschrieben. Es ist mehr als ein einfacher Kopiervorgang.

Uns beschäftigt hier die Frage, wie beim Starten des Systems die Hardware, die zunächst noch gar nichts kann, das Betriebssystem vom Massenspeicher in den Arbeitsspeicher lädt. Als kaltes **Booten** oder **Kaltstart** bezeichnet man einen Start vom Einschalten des Starkstroms an, als warmes Booten oder **Warmstart** einen erneuten Startvorgang einer bereits laufenden und daher warmen Maschine. Beim Warmstart entfallen einige der ersten Schritte (Tests).

Nach dem Einschalten wird ein einfaches Leseprogramm entweder Bit für Bit über eine besondere Tastatur eingegeben oder von einem permanenten Speicher (Boot-ROM) im System geholt. Mittels dieses Leseprogramms wird anschließend das Betriebssystem vom Massenspeicher gelesen, und dann kann es losgehen.

Beim Booten wird das Betriebssystem zunächst auf einem entfernbaren Datenträger (Band, CD-ROM, Diskette) gesucht, dann auf der Festplatte. Auf diese Weise läßt sich in einem bestehenden System auch einmal ein anderes Betriebssystem laden, zum Beispiel Linux statt PC-DOS, oder bei Beschädigung des Betriebssystems auf der Platte der Start von einer Diskette oder einem Band durchführen.

2.2 Das Besondere an UNIX

2.2.1 Die präunicische Zeit

Der Gedanke, Rechenvorgänge durch mechanische Systeme darzustellen, ist alt. Dass wir heute elektronische Systeme bevorzugen – und vielleicht in Zukunft optische Systeme – ist ein technologischer Fortschritt, kein grundsätzlicher. Umgekehrt hat man Zahlen schon immer dazu benutzt, Gegebenheiten aus der Welt der Dinge zu vertreten.

Wenn in der Jungsteinzeit ein Hirte – sein Name sei ÖTZI – sichergehen wollte, dass er abends genau so viel Stück Vieh heimbrachte, wie er morgens auf die Weide getrieben hatte, stand ihm nicht einmal ein Zahlensystem zur Verfügung, das nennenswert über die Zahl zwei hinausging. Da er nicht dumm war, wusste er sich zu helfen und bildete die Menge seines Viehs umkehrbar eindeutig auf eine Menge kleiner Steinchen ab, die er in einem Beutel bei sich trug. Blieb abends ein Steinchen übrig, fehlte ein Stück Vieh. Die Erkenntnis, dass Mengen andere Mengen in Bezug auf eine bestimmte

Eigenschaft (hier die Anzahl) vertreten können, war ein gewaltiger Sprung und der Beginn der Angewandten Mathematik.

Mit **Zahlensystemen** taten sich die Menschen früher schwer. Die Griechen – denen die Mathematik viel verdankt – hatten zwei zum Rechnen gleichermaßen ungeeignete Zahlensysteme. Das milesische System bildete die Zahlen 1 bis 9, 10 bis 90, 100 bis 900 auf das Alphabet ab, die Zahl 222 schrieb sich also $\sigma\kappa\beta$. Das attische oder akrophonische System verwendete die Anfangsbuchstaben der Zahlwörter, die Zehn (deka) wurde als Δ geschrieben. Einen Algorithmus wie das Sieb des ERATHOSTENES konnte nur ein Grieche ersinnen, dessen Zahlensystem vom Rechnen abschreckte.

Die Römer, deren Zahlenschreibweise wir heute noch allgemein kennen und für bestimmte Zwecke verwenden – siehe die Seitennumerierung zu Anfang des Buches oder das Verkehrszeichen Nr. E.5.3 der BinSchStrO – hatten auch nur bessere Strichlisten. Über ihre Rechenweise ist wenig bekannt. Sicher ist, dass sie wie ÖTZI Steinchen (calculi) gebrauchten.

Erst mit dem **Stellenwertsystem** der Araber und Inder wurde das Rechnen einfacher. Mit dem Einspluseins, dem Einmaleins und ein paar Regeln löst heute ein Kind arithmetische Aufgaben, deren Bewältigung im Altertum Bewunderung erregt oder im Mittelalter zu einer thermischen Entsorgung geführt hätte. Versuchen Sie einmal, römische Zahlen zu multiplizieren. Dann lernen Sie das Stellenwertsystem zu schätzen.

Seither sind Fortschritte erzielt worden, die recht praktisch sind, aber am Wesen des Umgangs mit Zahlen nichts ändern. Wir schieben keine Steinchen mehr über Rechentafeln, sondern Bits durch Register. Macht das einen Unterschied?

2.2.2 Entstehung

A long time ago in a galaxy far, far away ... so entstand UNIX nicht. Seine Entwicklung ist dennoch ungewöhnlich und auch heute noch für Überraschungen gut. Ende der sechziger Jahre schrieben sich zwei Mitarbeiter der Bell-Labs des AT&T-Konzerns, KEN THOMPSON und DENNIS RITCHIE, ein Betriebssystem zu ihrem eigenen Gebrauch[3]. Vorläufer reichen bis in den Anfang der sechziger Jahre zurück. Ihre Rechenanlage war eine ausgediente DEC PDP 7. Im Jahr 1970 prägte ein dritter Mitarbeiter, BRIAN KERNIGHAN, den Namen UNIX (Plural: UNICES oder deutsch auch UNIXe) für das Betriebssystem, außerdem wurde die PDP 7 durch eine PDP 11/20[4] ersetzt, um ein firmeninternes Textprojekt durchzuführen.

Der Name UNIX geht auf die indoeuropäische Wurzel **oinos* zurück,

[3]Eine authentische Zusammenfassung findet sich in The Bell System Technical Journal, Vol. 57, July-August 1978, Nr.6, Part 2, p. 1897 - 2312. Ähnlich auch im WWW: `www.bell-labs.com/history/unix/`.

[4]Die PDP 11 hatte einen Adressraum von 64 KByte. Ein PC/AT hatte einen Adressraum von wenigstens 16 MByte. Wenn UNIX allmählich zu einem Speicherfresser wird, liegt das nicht am Konzept, sondern daran, dass immer mehr hineingepackt wird.

karlsruherisch *oins*, neuhochdeutsch *eins*, mit Verwandten in allen indoeuropäischen Sprachen, die außer der baren Zahl *einzigartig, außerordentlich* bedeuten. UNIX hatte einen Vorgänger namens MULTICS (Multiplexed Information and Computing Service)[5], der bereits viele Ideen vorwegnahm, aber für die damaligen Hardware- und Programmiermöglichkeiten wohl etwas zu anspruchsvoll war und erfolglos blieb. KEN THOMPSON magerte MULTICS ab, bis es zuverlässig im Ein-Benutzer-Betrieb lief, daher UNIX. Inzwischen hat UNIX wieder zugenommen und ist – im Widerspruch zu seinem Namen – *das* Mehr-Benutzer-System.

DENNIS RITCHIE entwickelte auch eine neue Programmiersprache, die C getauft wurde (ein Vorgänger hieß B). Das UNIX-System wurde 1973 weitgehend auf diese Sprache umgeschrieben, um es besser erweitern und auf neue Computer übertragen zu können.

1975 wurde UNIX erstmals – gegen eine Schutzgebühr – an andere abgegeben, hauptsächlich an Universitäten. Vor allem die University of California in Berkeley beschäftigte sich mit UNIX und erweiterte es. Die Berkeley-Versionen – mit der Abkürzung **BSD** (Berkeley Software Distribution) versehen – leiten sich von der Version 7 von AT&T aus dem Jahr 1979 her. Die BSD hat einen bedeutenden Einfluss auf das UNIX ausgeübt, wie wir es heute kennen.

Seit 1983 wird UNIX von AT&T als **System V** vermarktet. Die lange Entwicklungszeit ohne den Einfluss kaufmännischer Interessen ist UNIX gut bekommen. AT&T vergab Lizenzen für die Nutzung der Programme, nicht für den Namen. Deshalb musste jeder Nutzer sein UNIX anders nennen: Hewlett-Packard wählte HP-UX, Siemens SINIX, DEC ULTRIX, Sun SunOS und Solaris, Apple A/UX, Silicon Graphics wählte Irix, sogar IBM nahm das ungeliebte, weil fremde Kind unter dem Namen AIX auf. Von den NeXT-Rechnern ist NeXTstep übriggeblieben, das auf unterschiedlicher Hardware läuft, unter anderem auf den HP 9000/7*. Openstep ist eine Fortführung in Form von Open Source im Netz. Im Jahr 2001 erlebte NeXTstep eine Wiedergeburt als Mac OS X. Eine frühe Portierung von UNIX auf PCs hieß XENIX und machte seinerzeit die Hälfte aller UNIX-Installationen aus. UNIX ist heute also zum einen ein geschützter Name, ursprünglich dem AT&T-Konzern gehörend, und zum anderen ein Gattungsname für miteinander verwandte Betriebssysteme von AIX bis XINU.

Die UNIX-Abfüllung von Hewlett-Packard – **HP-UX** – entstand 1982 und wurzelt in UNIX System III und Berkeley 4.1 BSD. Hewlett-Packard hat eigene Beiträge zur Grafik, Kommunikation, Datenverwaltung und zu Echtzeitfunktionen geleistet. In Europa gilt Siemens-Nixdorf mit **SINIX** als führender UNIX-Hersteller.

Im Jahr 1991 hat AT&T die UNIX-Geschäfte in eine Tochtergesellschaft namens Unix System Laboratories (USL) ausgelagert, mit der der amerikanische Netzhersteller Novell 1992 ein gemeinsames Unternehmen Univel gegründet hat. Anfang 1993 schließlich hat Novell USL übernommen. Ende 1993 hat Novell den Namen *UNIX* und die Spezifikationen der 1988 gegrün-

[5] `www.multicians.org/`

deten Open Software Foundation (OSF) vermacht, den Programmcode jedoch der Firma SCO.

Die **Open Software Foundation** (OSF) arbeitet ebenfalls an einer neuen, von AT&T unabhängigen Verwirklichung eines UNIX-Systems unter dem Namen **OSF/1**. Dieses System wird von den Mitgliedern der OSF (IBM, Hewlett-Packard, DEC, Bull, Siemens u. a.) angeboten werden. Die Open Software Foundation ist heute nach ihrem Zusmmenschluss mit X/Open im Jahre 1996 unter dem Namen **Open Group** ein Konsortium mehrerer Firmen mit dem Ziel, die Zusammenarbeit zwischen verschiedenen Software-Produkten zu verbessern. Die Open Group gibt die **Single UNIX Specification** heraus (siehe unten) und ist Inhaberin des geschützten Namens *UNIX*.

Die Väter von UNIX in den Bell Labs von AT&T – vor allem ROB PIKE und KEN THOMPSON – haben sich nicht auf ihren Lorbeeren ausgeruht und ein neues experimentelles Betriebssystem namens **Plan9** entwickelt, das in bewährter Weise seit Herbst 1992 an Universitäten weitergegeben wird. Es behält die Vorteile von UNIX wie das Klarkommen mit heterogener Hardware bei, läuft auf vernetzten Prozessoren (verteiltes Betriebssystem), kennt 16-bit-Zeichensätze und versucht, den Aufwand an Software zu minimieren, was angesichts der Entwicklung des alten UNIX und des X Window Systems als besonderer Vorzug zu werten ist.

Seit 1985 läuft an der Carnegie-Mellon-Universität in Pittsburgh ein Projekt mit dem Ziel, einen von Grund auf neuen UNIX-Kernel unter Berücksichtigung moderner Erkenntnisse und Anforderungen zu entwickeln. Das System namens **Mach** arbeitet bereits auf einigen Anlagen (zum Beispiel unter dem Namen NeXTstep). Ob es einen eigenen Zweig begründen wird wie seinerzeit das Berkeley-System und ob dieser im Lauf der Jahre wieder in die Linie von AT&T einmünden wird, weiß niemand zu sagen.

Das amerikanische Institute of Electrical and Electronics Engineers (IEEE) hat seit 1986 einen Standard namens **POSIX** (IEEE Standard 1003.1-1988 Portable Operating System Interface for Computer Environments) geschaffen, der die grundsätzlichen Anforderungen an UNIX-Systeme beschreibt, genauer gesagt an POSIX-konforme Betriebssysteme, wie sie auch immer heißen mögen. Im Jahr 1990 wurde POSIX mit leichten Änderungen als **International Standard ISO/IEC 9945-1:1990** angenommen. Der Standard ist leider nur gegen Bares vom IEEE zu haben. POSIX wird von der US-Regierung und der europäischen x/OPEN-Gruppe unterstützt und soll dazu führen, dass Software ohne Änderungen auf allen POSIX-konformen Systemen läuft. POSIX beschreibt eine Sammlung von Funktionen, die ein konformes Betriebssystem mindestens zur Verfügung stellen muss und die ein konformes Anwendungsprogramm höchstens benötigen darf. In welcher Form (Systemaufrufe oder Bibliotheksfunktionen) oder Sprache (C, FORTRAN, ADA) die Funktionssammlung verwirklicht wird, lässt der Standard offen. Trotzdem besteht eine enge Verbindung zwischen POSIX und C nach ISO/IEC 9899. POSIX selbst ist also *kein* Betriebssystem.

Die Bemühungen um die Einheit der UNIX-Welt führten 1998 zur Bildung der *Austin-Group* aus Vertretern des *Open Group*-Konsortiums, der IEEE und der Internationalen Normenorganisation ISO/IEC. Die Gruppe hat die **Single**

UNIX Specification erarbeitet, die gegenwärtig in Version 3 kostenfrei zum Lesen oder Herunterladen im Web steht (`www.unix-systems.org/`) und gemeinsamer Standard der drei beteiligten Einrichtungen ist. Ob die Spezifikation einmal POSIX ablöst, muss die Zukunft zeigen.

Neben diesen kommerziellen UNIXen gibt es mehr oder weniger freie Abkömmlinge. An einigen Universitäten sind UNIX-Systeme ohne Verwendung des ursprünglichen UNIX von AT&T entstanden, um sie uneingeschränkt im Unterricht oder für Experimente einsetzen zu können. Die bekanntesten sind MINIX, FreeBSD, NetBSD, OpenBSD und Linux. MINIX war vor allem als Demonstrationsobjekt für den Unterricht gedacht. FreeBSD ist für den PC optimiert und hat heute noch eine enge Verbindung zu BSD. NetBSD legt großen Wert auf klare Strukturen und gute Portierbarkeit. OpenBSD leitet sich von NetBSD her und betont Sicherheitsaspekte. Linux schließlich ist zum populärsten UNIX geworden, vielleicht weil sein Schöpfer LINUS B. TORVALDS im richtigen Zeitpunkt das Internet in die Entwicklung und Verbreitung einbezogen hat. Der Name *Linux* ist als Markenname geschützt. Es ist auf dem Weg, zum Maß aller universellen Betriebssysteme zu werden. Nachdem Installationswerkzeuge und grafische Benutzer-Oberflächen für Linux und die BSD-Verwandtschaft verfügbar sind, gibt es keinen Grund, für weniger Leistung mehr Geld auszugeben.

Da man zum Arbeiten außer dem Kern des Betriebssystems und der Shell noch eine Vielzahl von Anwendungen oder Werkzeugen braucht und das Zusammensuchen dieses Bündels mühsam ist, haben Firmen oder Organisationen die Mühe übernommen und stellen die Bündel als **Distribution** ins Netz oder auf CD/DVDs im Handel zur Verfügung. Bekannte Distributionen sind Debian, SuSE, Red Hat und Mandrake. Global gibt es etwa zweihundert, teilweise mit speziellen Zielsetzungen. Näheres siehe Abschnitt 2.13.2.3 *Distributionen* auf Seite 246. Weil die Distributionen bei allen Vorzügen auch die Gefahr der Entwicklung in verschiedene Richtungen bergen, hat sich das **Linux Standard Base Project** (LSB, `www.linuxbase.org/`) zum Ziel gesetzt, verbindliche Richtlinien und Tests für die Konformität zu entwickeln. Das Projekt verfolgt ähnliche Ziele wie POSIX, aber auf einem anderen Feld.

Das **GNU-Projekt** der Free Software Foundation Inc., einer Stiftung, verfolgt das Ziel, der UNIX-Welt Software im Quellcode ohne Kosten zur Verfügung zu stellen. Treibende Kraft ist RICHARD MATTHEW STALLMAN, the *Last of the True Hackers*. Der Gedanke hinter dem Projekt ist, dass jeder UNIX-Programmierer Software schreibt und braucht und unter dem Strich besser fährt, wenn er sich als Geber und Nehmer an GNU beteiligt. Einige große Programme (C-Compiler, Gnuplot, Ghostscript) sind bereits veröffentlicht, siehe Abschnitt 2.2.6 *GNU is not UNIX* auf Seite 36. Der erste Betriebssystem-Kernel namens **Hurd** kam 1996 heraus. Viel aus dem GNU-Projekt findet sich auch bei Linux[6] wieder, an dem inzwischen ganze Heer-

[6] Linux ist das moderne Betriebssystem, das den Zeitgeist widerspiegelt und den gehobenen Bedürfnissen einer neuen Generation entspricht: kreativ im Gebrauch, innovativ und zeitgemäß. Ich arbeite trotzdem damit.

scharen freiwilliger Programmierer unentgeltlich mitarbeiten. Programmieren kann auch ein Steckenpferd sein, wie Angeln oder Schrauben.

Schließlich gehört heute zu einem UNIX-System die grafische, netzfähige Benutzeroberfläche **X Window System**, die zwar vom Kern her gesehen nur eine Anwendung und daher nicht notwendig ist, für den Benutzer jedoch das Erscheinungsbild von UNIX bestimmt.

Weiteres zur UNIX-Familie findet man im World Wide Web (WWW) unter folgenden Uniform Resource Locators (URLs):

- `www.pasc.org/abstracts/posix.htm`
- `www.freebsd.org/`
- `wwww.netbsd.org/`
- `www.openbsd.org/`
- `www.linux.org/`
- `plan9.bell-labs.com/plan9/`
- `www.cs.cmu.edu/afs/cs.cmu.edu/ project/mach/public/www/mach.html`
- `www.cs.utah.edu/projects/flexmach/ mach4/html/Mach4-proj.html`
- `www.gnu.org/` (Free Software Foundation)
- `www.opengroup.org/` (OSF, X Window System)

sowie auf den Seiten der kommerziellen Hersteller.

Alle diese UNIX-Systeme sind in den wesentlichen Zügen gleich. Sie bauen aber auf verschiedenen Versionen von UNIX auf – vor allem unterscheiden sich der AT&T-Zweig und der Berkeley-Zweig – und weichen daher in Einzelheiten (Dateinamen, Einordnung von Dateien in Verzeichnisse, Kommando-Optionen) voneinander ab. Dennoch sind die Unterschiede gering im Vergleich zu den Unterschieden zwischen grundsätzlich fremden Systemen.

Trotz aller Verwandschaft der UNIXe ist Vorsicht geboten, wenn es heißt, irgendeine Hard- oder Software sei für UNIX verfügbar. Das ist bestenfalls die halbe Wahrheit. Bei Hardware kann die Verbindung zum UNIX-System schon an mechanischen Problemen scheitern. Eine Modemkarte für einen IBM-PC passt weder mechanisch noch elektrisch in eine HP 9000/712. Ausführbare Programme sind für einen bestimmten Prozessor kompiliert und nicht übertragbar, sie sind nicht binärkompatibel. Nur der Quellcode von Programmen, die für UNIX-Systeme geschrieben worden sind, lässt sich zwischen UNIX-Systemen austauschen, unter Umständen mit leichten Anpassungen.

2.2.3 Vor- und Nachteile

Niemand behauptet, UNIX sei das beste aller Betriebssysteme. Im Gegenteil, manche Computerhersteller halten ihre eigenen (proprietären) Betriebssysteme für besser[7]. Aber: ein IBM-Betriebssystem läuft nicht auf einem HP-Rechner und umgekehrt. UNIX hingegen stammt von einer Firma, die nicht als Computerhersteller aufgefallen ist, wird von vielen Personen und Firmen weiter entwickelt und läuft auf den Maschinen zahlreicher Hersteller[8]. Man braucht also nicht jedesmal umzulernen, wenn man den Computer wechselt. Bei Autos ist man längst so weit.

Diese gute **Portierbarkeit** rührt daher, dass UNIX mit Ausnahme der Treiber in einer höheren Programmiersprache – nämlich C – geschrieben ist. Zur Portierung auf eine neue Maschine braucht man also nur einige Treiber und einen C-Compiler in der maschinennahen und unbequemen Assemblersprache zu schreiben. Der Rest wird fast unverändert übernommen.

Eng mit dem Gesagten hängt zusammen, dass UNIX die Verbindung von Hardware unterschiedlicher Hersteller unterstützt. Man kann unter UNIX an einen Rechner von Hewlett-Packard Terminals von Wyse und Drucker von NEC anschließen. Das ist revolutionär. Eine Folge dieser Flexibilität ist, dass die Eigenschaften der gesamten Anlage in vielen **Konfigurations-Dateien** beschrieben sind, die Speicherplatz und Prozessorzeit verbrauchen. Stimmen die Eintragungen in diesen Dateien nicht, gibt es Störungen. Und meist ist an einer Störung eine Datei mehr beteiligt, als man denkt. Die Konfigurations-Dateien sind Klartext und lassen sich mit jedem Editor bearbeiten; sie enthalten auch erklärenden Kommentar. Die System-Manager hüten sie wie ihre Augäpfel, es steckt viel Arbeit darin.

Zweitens enthält UNIX einige Gedanken, die wegweisend waren. Es war von Anbeginn ein System für mehrere Benutzer (**Multiuser-System**). Andere Punkte sind die Datei-Hierarchie, die Umlenkung von Ein- und Ausgabe, Pipes, der Kommando-Interpreter, das Ansprechen der Peripherie als Dateien, leistungs- und erweiterungsfähige Werkzeuge und Dienstprogramme (Programme zum Erledigen häufig vorkommender Aufgaben). Bei UNIX hat es nie eine Unterscheidung zwischen Arbeitsplatzrechnern (Workstations) und Servern gegeben. Je nachdem welche Programme gestartet werden, arbeitet jeder UNIX-Rechner als Server für bestimmte Aufgaben. Diese frühe Anlage wesentlicher Eigenschaften trägt zur Stabilität heutiger UNIX-Systeme bei. Man muss den Weitblick der Väter von UNIX bewundern.

Die Stärken von UNIX liegen in der Programmierumgebung, in der Kommunikation und in der Verarbeitung anspruchsvoller Texte. Auch die Offenheit einiger UNIX-Abfüllungen ist ein großes Plus. Wer will, kann genau nachvollziehen, was ein bestimmtes Programm tut oder lässt. Schwächen von UNIX sind das Fehlen von standardisierten Grafik- und Datenbankfunktionen sowie eine nur mittlere Sicherheit.

Wenn UNIX nichts sagt, geht es ihm gut, oder es ist mausetot. Wer viel am Bildschirm arbeitet, ist für die Schweigsamkeit jedoch dankbar. Es gibt auch technische Gründe für die Zurückhaltung: wohin sollten die Meldungen eines Kommandos in einer Pipe oder bei einem Hintergrundprozess gehen,

[7]Real programmers use IBM OS/370.

[8]Der Linux-Kernel 2.6.0 unterstützt mindestens 15 verschiedene Plattformen

ohne andere Prozesse zu stören? Die Vielzahl und der Einfallsreichtum der Programmierer, die an UNIX mitgearbeitet haben und noch weiterarbeiten, haben stellenweise zu einer etwas unübersichtlichen und den Anfänger verwirrenden Fülle von Werkzeugen geführt. Statt *einer* Shell gibt es gleich ein Dutzend Geschmacksrichtungen. Nach heutiger Erkenntnis hat UNIX auch Schwächen theoretischer Art, siehe ANDREW S. TANENBAUM.

UNIX gilt als schwierig. Aber komplexe Aufgaben lösen andere Betriebssysteme auch nicht einfacher als UNIX. UNIX ist sicherer als PC-DOS oder ähnliche Betriebssysteme. Den Reset-Knopf brauche ich vielleicht einmal im Vierteljahr, und das meist im Zusammenhang mit Systemumstellungen, die heikel sein können. Anwendungsprogramme eines gewöhnlichen Benutzers haben gar keine Möglichkeit, das System abstürzen zu lassen (sagen wir vorsichtshalber fast keine). Nicht ohne Grund arbeiten viele Server im Internet mit UNIX.

2.2.4 UNIX-Philosophie

Unter UNIX stehen über tausend **Dienstprogramme** (utility, utilitaire) zur Verfügung. Dienstprogramme werden mit dem Betriebssystem geliefert, gehören aber nicht zum Kern, sondern haben den Rang von Anwendungen. Sie erledigen immer wieder und überall vorkommende Arbeiten wie Anzeige von Dateiverzeichnissen, Kopieren, Löschen, Editieren von Texten, Sortieren und Suchen. Die Dienstprogramme von UNIX erfüllen jeweils *eine* überschaubare Funktion. Komplizierte Aufgaben werden durch Kombinationen von Dienstprogrammen gemeistert. Eierlegende Wollmilchsäue widersprechen der reinen UNIX-Lehre, kommen aber vor, siehe `emacs(1)`.

Der Benutzer wird mit unnötigen Informationen verschont. No news are good news. Rückfragen, ob man ein Kommando wirklich ernst gemeint hat, gibt es selten. UNIX-Werkzeuge hindern den Benutzer nicht daran, etwas zu tun, sondern unterstützen ihn. In früheren Jahrzehnten ging es vor allem darum, auf langsamer Hardware schnell zu arbeiten; der Wunsch kommt auch heute noch vor. Ob das Tun sinnvoll ist oder nicht, bleibt dem Benutzer überlassen, er wird möglichst wenig gegängelt. UNIX rechnet mit dem mündigen Anwender. Ein gewisser Gegensatz zu anderen Welten ist zu erkennen.

Von Anbeginn hat UNIX Wert auf Portabilität und den Umgang mit heterogener Hardware gelegt, mitunter zu Lasten der Effizienz. Dazu gehört, dass Konfigurationen und Daten möglichst in einfachen ASCII-Textfiles abgelegt werden. Um ein UNIX-Subsystem – sagen wir die Secure Shell oder den inet-Dämon – zu konfigurieren, brauche ich nur meinen Lieblingseditor. Ich kann auch einfach eine fremde Konfiguration oder fremde Skripte abkupfern und anpassen. Konfigurationswerkzeuge mit grafischer Benutzeroberfläche mögen elegant aussehen, aber wehe, sie funktionieren nicht hundertprozentig. Dann steht der Benutzer in einem ziemlich dunklen Wald. Und die Kombination mit anderen Werkzeugen ist nahezu unmöglich.

UNIX geht davon aus, dass alle Benutzer guten Willens sind und fördert ihre Zusammenarbeit. Es gibt aber Hilfsmittel zur Überwachung. Schwarze Schafe entdeckt ein gewissenhafter System-Manager bald und sperrt sie ein.

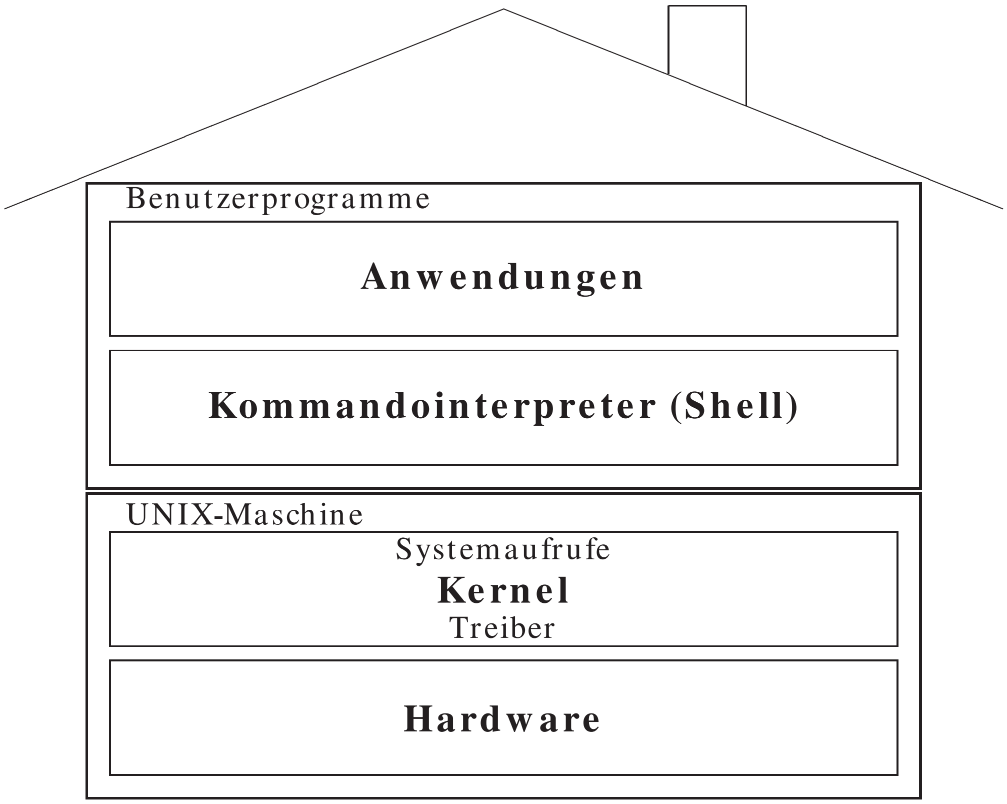

Abb. 2.1: Schematischer Aufbau von UNIX

Der US-amerikanische Autor HARLEY HAHN schreibt *Unix is the name of a culture*. Übertrieben, aber ein eigener Stil im Umgang mit Benutzern und Daten ist in der UNIX-Welt und dem von ihr geprägten Internet zu erkennen.

2.2.5 Aufbau

Man kann sich ein UNIX-System als ein Gebäude mit mehreren Stockwerken vorstellen (Abbildung 2.1 auf Seite 35). Im Keller steht die **Hardware**. Darüber sitzt im Erdgeschoss der **UNIX-Kern** (kernel, noyau), der mit der Hardware über die Treiberprogramme (driver, pilote) und mit den höheren Etagen über die Systemaufrufe (system call, fonction système) verkehrt. Außerdem enthält der Kern die Prozessverwaltung und das Datei-System. Der betriebsfähige Kern ist ein einziges Programm, das im Datei-System finden ist (`hpux`, `vmux`, `vmlinuz`). Hardware und UNIX-Kern bilden die UNIX-Maschine. Die Grenze des Kerns nach oben (die Systemaufrufe) ist in der **UNIX System V Interface Definition** (SVID) beschrieben. Was aus den oberen Stockwerken kommt, sind für den UNIX-Kern **Benutzer-** oder **Anwendungsprogramme**, auch falls sie zum Lieferumfang von UNIX gehören. Die Anwendungsprogramme sind austauschbar, veränderbar, ergänzbar. Für den Benutzer im Dachgeschoss ist die Sicht etwas anders. Er verkehrt mit der Maschine über einen Kommandointerpreter, die **Shell**. Sie nimmt seine Wünsche entgegen und sorgt für die Ausführung. UNIX ist für ihn in erster Linie die Shell. Allerdings könnte sich ein Benutzer eine eigene Shell schreiben oder Programme, die ohne Shell auskommen. Dieses Doppelgesicht des Kommandointerpreters spiegelt seine Mittlerrolle zwischen Benutzer und Betriebssystem-Kern wider.

Die Verteilung der Aufgaben zwischen Kern und Anwendungen ist in manchen Punkten willkürlich. Eigentlich sollte ein Kern nur die unbedingt notwendigen Funktionen enthalten. Ein Monolith von Kern, der alles macht, ist bei den heutigen Anforderungen kaum noch zu organisieren. In MINIX und OS/2 beispielsweise ist das Datei-System eine Anwendung, also nicht Bestandteil des Kerns. Auch die Arbeitsspeicherverwaltung – das Memory Management – lässt sich auslagern, so dass nur noch Steuerungs- und Sicherheitsfunktionen im Kern verbleiben.

Wer tiefer in den Aufbau von UNIX oder verwandten Betriebssystemen eindringen möchte, sollte mit den im Anhang L *Zum Weiterlesen* auf Seite 358 genannten Büchern von ANDREW S. TANENBAUM beginnen. Der Quellcode zu dem dort beschriebenen Betriebssystem MINIX ist ebenso wie für Linux auf Papier, Disketten und im Netz verfügbar, einschließlich Treibern und Systemaufrufen. Weiter ist in der Universität Karlsruhe, Institut für Betriebs- und Dialogsysteme ein Betriebssystem KBS für einen Kleinrechner (Atari) entwickelt worden, das in der Zeitschrift c't Nr. 2, 3 und 4/1993 beschrieben ist und zu dem ausführliche Unterlagen erhältlich sind. Dieses System ist zwar kein UNIX, sondern etwas kleiner und daher überschaubarer, aber die meisten Aufgaben und einige Lösungen sind UNIX-ähnlich.

2.2.6 GNU is not UNIX

Die Gnus (Connochaetes) sind eine Antilopenart in Süd- und Ostafrika, von den Buren Wildebeest genannt. Nach ALFRED BREHM sind es höchst absonderliche, gesellig lebende Tiere, in deren Wesen etwas Komisches, Feuriges, Überspanntes steckt.

Im Zusammenhang mit UNIX-Software stößt man oft auf das **GNU-Projekt** der **Free Software Foundation**. Es bezweckt, Benutzern Software ohne Einschränkungen juristischer oder finanzieller Art zur Verfügung zu stellen. Die Software ist durch Copyright[9] geschützt, darf aber unentgeltlich benutzt, verändert und weitergegeben werden, jedoch immer nur zusammen mit dem Quellcode, so dass andere Benutzer die Programme ebenfalls anpassen und weiterentwickeln können. Einzelheiten siehe die GNU **General Public License** (GPL). Strenggenommen ist zwischen Software aus dem GNU-Projekt und Software beliebiger Herkunft, die unter GNU-Regeln zur Verfügung gestellt wird, zu unterscheiden. Für den Anwender ist das nebensächlich. Der Original Point of Distribution ist `prep.ai.mit.edu`, aber die GNU-Programme werden auch auf vielen anderen FTP-Servern gehalten. Eine kleine Auswahl aus dem Projekt:

- `emacs` – ein mächtiger Editor,

[9]Die GNU-Leute bezeichnen ihre besondere Art des Copyrights als Copyleft, siehe `www.gnu.org/copyleft/copyleft.html`.

- gnuchess – ein Schachspiel,
- gcc – ein ANSI-C-Compiler (auch für PC-DOS, siehe djgpp),
- g++ – ein C++-Compiler,
- gawk – eine Alternative zu awk(1),
- flex – eine Alternative zu lex(1),
- bison – eine Alternative zu yacc(1),
- ghostscript – ein PostScript-Interpreter,
- ghostview – ein PostScript-Pager,
- ispell – ein Rechtschreibungsprüfer,
- gzip – ein wirkungsvoller Datei-Kompressor,
- f2c – ein FORTRAN-zu-C-Konverter,
- gtar – ein Archivierer wie tar(1),
- bash – die Bourne-again-Shell,
- gimp – das GNU Image Manipulation Program,
- recode – ein Filter zur Umwandlung von Zeichensätzen

Besonders wertvoll ist der Zugang zum Quellcode, so dass man die Programme ergänzen und portieren kann. Die Werkzeuge sind nicht nur kostenfrei, sondern zum Teil auch besser als die entsprechenden originalen UNIX-Werkzeuge. Die Gedanken hinter dem Projekt, das 1984 von RICHARD MATTHEW STALLMAN begründet wurde, sind im GNU Manifesto im WWW nachzulesen (www.gnu.org/gnu/manifesto.html).

Der geneigte Leser kann die nächsten Zeilen überspringen, bis er den Abschnitt 2.8 *Programmer's Workbench* auf Seite 188 verdaut hat. Die GNU-Programme sind *immer* als Quellen verfügbar, oft zusammen mit Makefiles für verschiedene Systeme, selten als unmittelbar ausführbare Programme. Man muss also noch etwas Arbeit hineinstecken, bis man sie nutzen kann. Gute Kenntnisse von make(1) sind hilfreich. Vereinzelt nehmen auch Firmen die Kompilierung vor und verkaufen die ausführbaren Programme zu einem gemäßigten Preis. Am Beispiel des oft verwendeten Packers gzip(1) wollen wir uns ansehen, wie eine Installation auf einer UNIX-Maschine vor sich geht:

- Wir legen ein Unterverzeichnis gzip an, gehen hinein und bauen eine Anonymous-FTP-Verbindung mit ftp.rus.uni-stuttgart.de auf.
- Dann wechseln wir dort in das Verzeichnis pub/unix/gnu, das ziemlich viele Einträge enthält.
- Wir stellen den binären Übertragungsmodus (binary oder image) ein und holen uns mittels mget gzip* die gewünschten Dateien. Angeboten werden gzip...msdos.exe, gzip...shar, gzip...tar und gzip...tar.gz. Letzteres ist zwar in der Regel das beste Format, setzt jedoch voraus, dass man gzip(1) bereits hat. Wir wählen also das tar-Archiv, rund 200 Kbyte.

- Mittels `tar -xf gzip-1.2.4.tar` entpacken wir das Archiv. Anschließend finden wir ein Unterverzeichnis `gzip-1.2.4` und wechseln hinein. Obige Versionsnummer durch aktuelle Nummer ersetzen.
- Mindestens die Textfiles `README` und `INSTALL` sollte man lesen, bevor es weitergeht.
- Mittels `./configure` wird ein angepasstes `Makefile` erzeugt. Man sollte es sich ansehen, allerdings nur äußerst vorsichtig editieren, falls unvermeidbar.
- Dann folgt ein schlichtes `make`. Läuft es ohne Fehlermeldungen durch, gehört man zu den Glücklichen dieser Erde.
- Hier kann man noch `make check` aufrufen, gibt es nicht immer.
- Zum Kopieren in die üblichen Verzeichnisse (`/usr/local/bin` usw.) gibt man als Superuser `make install` ein.
- Zu guter Letzt räumt man mittels `make clean` auf. Nun haben wir `gzip(1)` sowie `gunzip(1)` und können weitere GNU-Werkzeuge in gzippter Form holen. Zugriffsrechte prüfen.

Die Installation geht nicht immer so glatt über die Bühne. Die häufigsten Überraschungen beim Einrichten von GNU- oder anderer freier Software sind:

- Fehlende include-Dateien oder Funktionsbibliotheken (irgendwoher beschaffen),
- die Dateien sind zwar vorhanden, liegen aber im falschen Verzeichnis (in diesem Fall weiche Links anlegen),
- die Dateien sind am richtigen Ort, die Zugriffsrechte reichen nicht aus (Zugriffsrechte ändern oder als Superuser kompilieren),
- es werden zusätzlich einige Hilfsprogramme wie `groff(1)`, `gmake(1)` oder `patch(1)` aus dem GNU-Projekt benötigt (per FTP holen und hoffen, dass sie sich problemlos kompilieren lassen),
- es ist zwar alles wie erforderlich eingerichtet, aber die Typen der Argumente und Rückgabewerte sind anders, als sie die GNU-Software erwartet. Dann passen irgendwelche Versionen nicht zueinander, und es ist Hand- und Hirnarbeit angesagt,
- dem Compiler muss eigens per Option befohlen werden, sich ANSI- oder POSIX-konform zu verhalten. Ein Hinweis darauf sollte in einer `README`-Datei zu finden sein, aber manchmal muss man die Quellen und die Include-Dateien lesen, um darauf zu kommen.

Ein allgemeines Rezept läßt sich nicht angeben. Gelegentlich hatte ich mit dem Editieren der Makefiles Erfolg, manchmal auch nicht. Dann kann man sich noch nach der neuesten Version der GNU-Software umschauen oder eine Email an den Autor schreiben. Es kommen aber auch angenehme Überraschungen vor – die Kompilierung und Einrichtung von `gmake(1)` und `gzip(1)` gingen bei mir ohne Probleme über die Bühne – und die GNU-Software ist den Versuch der Einrichtung allemal wert. Zudem lernt man einiges über das Programmieren portabler Software und die Struktur von Programmen.

2.3 Daten in Bewegung: Prozesse

2.3.1 Was ist ein Prozess?

Wir müssen – wenn wir uns präzise ausdrücken wollen – unterscheiden zwischen einem Programm und einem Prozess, auch Auftrag oder Task genannt. Ein Programm *läuft* nicht, sondern ruht als Datei im Datei-System, siehe Abschnitt *Daten in Ruhe: Dateien* ab Seite 54. Beim Aufruf wird es in den Arbeitsspeicher kopiert, mit Daten ergänzt und bildet dann einen **Prozess** (process, processus), der Prozessorzeit anfordert und seine Tätigkeit entfaltet. Man kann den Prozess als die grundlegende, unteilbare Einheit ansehen, in der Programme ausgeführt werden. Inzwischen unterteilt man jedoch in bestimmten Zusammenhängen Prozesse noch feiner (threads), wie auch das Atom heute nicht mehr unteilbar ist. Ein Prozess ist eine kleine, abgeschlossene Welt für sich, die mit der Außenwelt nur über wenige, genau kontrollierte Wege oder Kanäle Verbindung hält. Prozesse und Dateien sind Grundkonzepte von UNIX. Alle gängigen Betriebssysteme verwenden diese beiden Grundkonzepte, in der Ausgestaltung unterscheiden sich die Systeme.

Ein UNIX-Prozess besteht aus drei Teilen:

- einem (virtuellen) Speicher-Adressraum,
- System-Ressourcen, zum Beispiel offene Dateien,
- eine Folge (Sequenz, Thread im allgemeinen Sinn) von Anweisungen, die nacheinander von der CPU ausgeführt werden.

Diese drei Teile gehören dem Prozess, kein fremder Prozess hat darin etwas verloren, worüber das Betriebssystem wacht. Der Speicher-Adressraum ist virtuell, das heißt er steht zunächst nur auf dem Papier und wird bei Bedarf auf wirklichen Speicher abgebildet. Er gliedert sich seinerseits wieder in drei Teile oder Segmente:

- ein **Code-Segment** (auch Text-Segment genannt, obwohl aus unlesbarem Maschinencode bestehend),
- ein **Benutzerdaten-Segment** und
- ein **Systemdaten-Segment**.

Der Prozess bekommt eine eindeutige Nummer, die **Prozess-ID** (PID). Das Code-Segment wird bei der Erzeugung des Prozesses mit dem Prozesscode gefüllt und bleibt dann vor weiteren schreibenden Zugriffen geschützt. Das Benutzerdaten-Segment wird vom Prozess beschrieben und gelesen, das Systemdaten-Segment darf vom Prozess gelesen und vom Betriebssystem beschrieben und gelesen werden. Im Benutzerdaten-Segment finden sich unter anderem die dem Prozess zugeordneten Puffer. Unter die Systemdaten fallen Statusinformationen über die Hardware und über offene Dateien. Durch die Verteilung der Rechte wird verhindert, daß ein wildgewordener Prozess das ganze System lahmlegt[10]. Ein einzelner Prozess kann hängen bleiben, abstürzen oder aussteigen, das System ist dadurch noch lange nicht gefährdet. Ein Booten wegen eines Systemabsturzes ist unter UNIX äußerst selten vonnöten.

Die gerade im System aktiven Prozesse listet man mit dem Kommando `ps(1)` mit der Option `-ef` auf. Die Ausgabe sieht ungefähr so aus:

```
     UID   PID  PPID     STIME TTY      TIME COMMAND
    root     0     0   Jan 22  ?        0:04 swapper
    root     1     0   Jan 22  ?        1:13 init
    root     2     0   Jan 22  ?        0:00 pagedaemon
    root     3     0   Jan 22  ?        0:00 statdaemon
    root    31     1   Jan 22  ?        0:18 /etc/cron
    root    46     1   Jan 22  console  0:00 sleep
    root    48     1   Jan 22  console  0:00 sleep
    root    59     1   Jan 22  ?        0:00 /etc/delog
 wualex1  1279     1   Feb  4  console  0:00 -ksh [ksh]
    root  1820     1  00:48:00 tty0p2   0:00 /etc/getty
      lp  1879     1  00:51:17 ?        0:00 lpsched
    root  2476     1  13:02:40 tty1p0   0:00 /etc/getty
    root  2497     1  13:04:31 tty0p1   0:00 /etc/getty
    root  2589     1  13:31:34 tty1p1   0:00 /etc/getty
 wualex1  2595  1279  13:32:39 console  0:00 -ksh [ksh]
 wualex1  2596  2595  13:32:40 console  0:00 ps -ef
 wualex1  2597  2595  13:32:40 console  0:00  [sort]
```

Die Spalten bedeuten folgendes:

- UID User-ID, Besitzer des Prozesses
- PID Prozess-ID, Prozessnummer
- PPID Parent Process ID, Nummer des Elternprozesses
- STIME Start Time des Prozesses
- TTY Kontroll-Terminal des Prozesses (*nicht* der Terminaltyp!)
- TIME Dauer der Ausführung des Prozesses
- COMMAND zugehöriger Programmaufruf (Kommandozeile)

Im obigen Beispiel ist der jüngste Prozess `sort` mit der Nr. 2597; er ist zusammen mit `ps -ef` Teil einer Pipe, um die Ausgabe nach der PID sortiert auf den Bildschirm zu bekommen. Beide sind Kinder einer Shell `ksh` mit der Nr. 2595. Die eckigen Klammern um den Namen weisen darauf hin, daß der

[10]In einfacheren Betriebssystemen ist es möglich, daß ein Programm während der Ausführung seinen im Arbeitsspeicher stehenden Code verändert. Man könnte ein Programm schreiben, daß sich selbst auffrißt.

Prozess bei seiner Erzeugung möglicherweise einen anderen Namen hatte, was meist unwichtig ist. Die Shell ihrerseits ist Kind der Login-Shell mit der Nr. 1279, die aus einem `getty`-Prozess mit derselben Nummer entstanden ist. Elternprozess aller `getty`-Prozesse ist der Dämon `init` mit der PID 1, der Urahne der meisten Prozesse auf dem System. Dieser und noch wenige andere Dämonen wurden vom `swapper` mit der PID 0 erzeugt, der den Verkehr zwischen Arbeits- und Massenspeicher regelt. Man bemerkt ferner die Dämonen `cron(1M)` und `lpsched(1M)` sowie zwei schlafende Prozesse (Nr. 46 und 48), die die Druckerports offen halten. Will man wissen, ob ein bestimmter Prozess (im Beispiel `tar(1)` noch am Leben ist, filtert man die umfangreiche Ausgabe von `ps(1) -ef` durch das Kommando `grep(1)`:

```
ps -ef | grep tar
```

Eine Erklärung der hier vorkommenden Begriffe folgt auf den nächsten Seiten.

Da jeder Benutzer sich mittels obigem Kommando alle Prozesse ansehen kann und in der letzten Spalte die jeweiligen Kommandozeilen mit allen Optionen und Argumenten zu lesen sind, ist anzuraten, niemals sensible Daten wie geheime Schlüssel in der Kommandozeile mit einzugeben. Kommandos wie `crypt(1)` erfragen den Schlüssel im Dialog, so dass er nicht in der Prozessauflistung erscheint.

Alle Prozesse, die von einem gemeinsamen Vorfahren abstammen, gehören zu einer **Prozessgruppe**. Sie verkehren mit der Außenwelt über dasselbe **Kontroll-Terminal** und empfangen bestimmte **Signale** als Gruppe. Der gemeinsame Vorfahre ist der **Prozessgruppenleiter**. Über den Systemaufruf `setpgrp(2)` ernennt sich ein Prozess zum Leiter einer neuen Gruppe. Ohne diese Möglichkeit gäbe es im System nur eine Prozessgruppe, da alle Prozesse auf `init` zurückgehen.

2.3.2 Prozesserzeugung (exec, fork)

Nehmen wir an, wir hätten bereits einen Prozess. Dieser kopiert sich, dann haben wir zwei gleiche Prozesse, einen **Elternprozess** und einen **Kindprozess**. Das Codesegment des Kindprozesses wird nun mit dem Code des neuen Kommandos oder Programmes überlagert. Dann wird der Kindprozess ausgeführt, während der Elternprozess wartet. Ist der Kindprozess fertig, wird er im Speicher gelöscht und sein Ende dem Elternprozess mitgeteilt, der nun weitermacht. Der Kindprozess kann seinerseits – solange er lebt – wieder Kinder bekommen, so daß einem lebhaften Familienleben nichts im Wege steht (Abbildung 2.2 auf Seite 42). Durch das Kopieren erbt der Kindprozess beide Datensegmente des Elternprozesses und kann damit arbeiten. Eine Rückvererbung von den Kindern auf die Eltern gibt es im Gegensatz zum bürgerlichen Recht (BGB, 5. Buch) nicht, die Vererbung ist eher biologisch aufzufassen. Programmiertechnisch bedeutet die Prozesserzeugung den Aufruf eines selbständigen Programmes (Hauptprogrammes) mittels der Systemaufrufe `exec(2)` und `fork(2)` aus einem anderen Programm heraus. Diese Art der Prozesserzeugung ist flexibel, aber nicht gerade effektiv. Wenn es auf

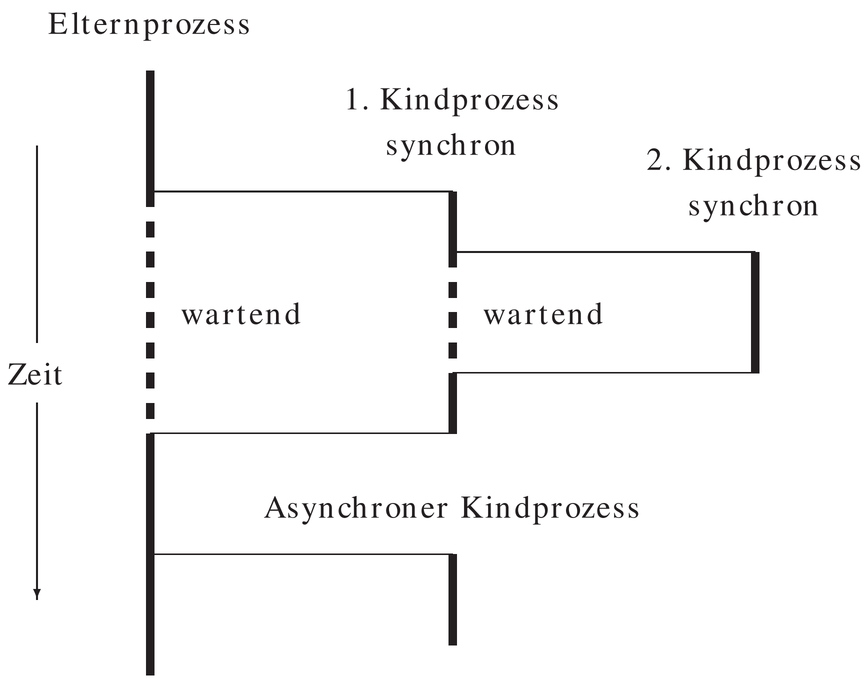

Abb. 2.2: Synchrone und asynchrone Prozesse

Geschwindigkeit ankommt, versucht man daher, die Anzahl der zu erzeugenden Prozesse niedrig zu halten. Beispielsweise werden auf hoch belasteten WWW-Servern einige `httpd`-Prozesse auf Vorrat erzeugt (preforking server), die beim Eintreffen von Anfragen nur aufzuwachen brauchen.

Wie gelangen wir nun zu dem ersten Prozess im System, der zwangsläufig als Vollwaise auf die Welt kommen muß? Beim Einschalten des Computers läuft ein besonderer Vorgang ab, der den Prozess Nr. 0 mit Namen `swapper` erzeugt. Der ruft sogleich einige zum Betrieb erforderliche Prozesse ins Leben, darunter der `init`-**Prozess** mit der Nr. 1. `init` erzeugt für jedes Terminal einen `getty`-Prozess, ist also unter Umständen Elternteil einer zahlreichen Nachkommenschaft. Die `getty`-Prozesse nehmen die Anmeldungen der Benutzer entgegen und ersetzen sich ohne Erzeugung eines Kindprozesses durch den `login`-Prozess, der die Anmeldung prüft. Bei Erfolg ersetzt sich der `login`-Prozess durch den Kommandointerpreter, die Shell. Der Elternprozess dieser ersten Shell ist also `init`, ihre Prozess-ID und ihre Startzeit ist die des zugehörigen `getty`-Prozesses. Alle weiteren Prozesse der Sitzung sind Kinder, Enkel, Urenkel usw. der Sitzungsshell. Am Ende einer Sitzung stirbt die Sitzungsshell ersatzlos. Der `init`-Prozess – der Urahne – erfährt dies und erzeugt aufgrund eines `respawn`-Eintrages in `/etc/inittab(4)` wieder einen neuen `getty`-Prozess.

Wenn der Eltern-Prozess mit seiner Arbeit wartet, bis sein Abkömmling fertig ist, spricht man beim Kind von einem **synchronen** oder **Vordergrund-Prozess**. Das ist der Regelfall. Man kann aber auch als letztes Zeichen der Kommandozeile das et-Zeichen `&` geben, dann ist der Elternprozess sofort zu neuen Taten bereit:

```
myprogram &
```

Der Kind-Prozess läuft **asynchron** oder im **Hintergrund**. Sinnvoll ist das nur, falls der Elternprozess nicht die Ergebnisse seines Kindes benötigt. Ein im Vordergrund gestarteter Prozess läßt sich auf einigen UNIX-Systemen – abhängig von Kernel und Shell – mit der Tastenkombination `control-z` unterbrechen und dann mit dem Kommando `bg pid` in den Hintergrund schicken. Umgekehrt holt ihn `fg pid` wieder in den Vordergrund.

Der Benutzer, der einen Prozess aus seiner Sitzung gestartet hat, ist der Besitzer des Prozesses und verfügt über ihn, insbesondere darf er ihn gewaltsam beenden. Das Terminal, von dem der Prozess aus gestartet wurde, ist sein **Kontroll-Terminal** `/dev/tty`, über das er seinen Dialog abwickelt.

Das System führt eine **Prozesstabelle**, in der für jeden Prozess alle zugehörigen Informationen von der Prozess-ID bis zu den Zeigern auf die Speichersegmente liegen. Das Kommando `ps(1)` greift auf diese Tabelle zu.

2.3.3 Selbständige Prozesse (nohup)

Stirbt ein Elternprozess, so sterben automatisch alle etwa noch lebenden Kindprozesse; traurig, aber wahr. Mit dem Ende einer Sitzungsshell ist auch das Ende aller in der Sitzung erzeugten Prozesse gekommen. Man möchte gelegentlich jedoch Rechenprogramme zum Beispiel über Nacht laufen lassen, ohne die Sitzung während der ganzen Zeit fortzusetzen.

Mit dem Vorkommando `nohup(1)` (no hang up) vor dem Programmaufruf erreicht man, daß das Programm bei Beendigung der Sitzung weiterläuft, es ist von der Sitzung abgekoppelt. Gleichzeitig muß man es natürlich im Hintergrund (&) laufen lassen, sonst läßt sich die Sitzung nicht vom Terminal aus beenden. Tatsächlich bewirkt das Vorkommando `nohup(1)`, daß das Signal Nr. 1 (SIGHUP, hangup) ignoriert wird. Der Aufruf sieht so aus:

```
nohup program &
```

Für `program` ist der Name des Programmes einzusetzen, das von der Sitzung abgekoppelt werden soll. Will man `nohup` auf Kommandofolgen oder Pipes anwenden, sind sie in ein Shellskript zu verpacken. Die Ausgabe eines nogehuppten Programmes geht automatisch in eine Datei namens `nohup.out`, falls sie nicht umgelenkt wird.

Starten wir das Shellskript `traptest` mit `nohup traptest &` und beenden unsere Sitzung (zweimal `exit` geben), so können wir in einer neuen Sitzung mit `ps -ef` feststellen, daß `traptest` in Form einer Shell und eines `sleep`-Prozesses weiterlebt. Besitzer und ursprüngliches Kontroll-Terminal werden angezeigt. Wir sollten die Shell möglichst bald mit `kill(1)` beenden.

2.3.4 Priorität (nice)

Ein Dialog-Prozess sollte unverzüglich antworten, sonst nervt er. Bei einem Rechenprozess, der nächtelang im Hintergrund läuft, kommt es dagegen auf eine Stunde mehr oder weniger nicht an. Deshalb werden den Prozessen unterschiedliche **Prioritäten** eingeräumt. In der Schlange der auf Prozessorzeit wartenden Prozesse kommt ein Prozess hoher Priorität vor einem Prozess niedriger Priorität, das heißt er kommt früher an die Reihe. Es bedeutet nicht, daß ihm mehr Prozessorzeit zugeteilt wird.

Die Priorität eines Prozesses, die man sich mit `ps -elf` oder `ps -al` anzeigen läßt, setzt sich aus zwei Teilen zusammen. Der Benutzer kann einem Prozess beim Aufruf einen `nice`-Faktor mitgeben. Ein hoher Wert des Faktors führt zu einer niedrigen Priorität. Den zweiten Teil berechnet das System unter dem Gesichtspunkt möglichst hoher Systemeffizienz. In einem Echtzeit-System wäre eine solche eigenmächtige Veränderung der Priorität untragbar.

Die `nice`-Faktoren haben Werte von 0 bis +39, unter Debian GNU/Linux von -20 bis +19. Der Standardwert eines Vordergrundprozesses ist 20, unter Debian GNU/Linux 0. Mit dem Aufruf:

```
nice myprocess
```

setzt man den `nice`-Faktor des Prozesses `myprocess` um 10 herauf, seine Priorität im System wird schlechter. Negative Werte als Argumente von `nice` verbessern den `nice`-Faktor über den Standardwert hinaus und sind dem System-Manager für Notfälle vorbehalten. Der `nice`-Faktor kann nur beim Prozessstart verändert werden, die Priorität eines bereits laufenden Prozesses läßt sich nicht mehr beeinflussen, es sei denn, Ihr System kenne das Kommando `renice`. In Verbindung mit `nohup` ist `nice` gebräuchlich:

```
nohup nice program &
nohup time nice program &
nohup nice time program &
```

Im letzten Fall wird auch die Priorität des `time`-Oberprogrammes herabgesetzt, was aber nicht viel bringt, da es ohnehin die meiste Zeit schläft.

Im GNU-Projekt findet sich ein Kommando `renice(1)`, das die Priorität eines laufenden Prozesses zu ändern ermöglicht. Weitere Überlegungen zur Priorität stehen im Abschnitt 2.12 *Echtzeit-Erweiterungen* auf Seite 241.

2.3.5 Dämonen

2.3.5.1 Was ist ein Dämon?

Das griechische Wort $\delta\alpha\iota\mu\omega\nu$ (daimon) bezeichnet alles zwischen Gott und Teufel, Holde wie Unholde; die UNIX-**Dämonen** sind in der Mitte angesiedelt, nicht immer zu durchschauen und vorwiegend nützlich. Es sind Prozesse, die nicht an einen Benutzer und ein Kontroll-Terminal gebunden sind. Das System erzeugt sie auf Veranlassung des System-Managers, meist beim Starten des Systems. Wie Heinzelmännchen erledigen sie im stillen Verwaltungsaufgaben und stellen Dienstleistungen zur Verfügung. Beispiele sind der Druckerspooler `lpsched(1M)`, Netzdienste wie `inetd(1M)` oder `sendmail(1M)` und der Zeitdämon `cron(1M)`. Dämonen, die beim Systemstart von dem Shellskript `/etc/rc` ins Leben gerufen worden sind, weisen in der Prozessliste als Kontroll-Terminal ein Fragezeichen auf. Mittlerweile ist der Start der Dämonen beim Booten etwas komplizierter geworden und auf eine Kette von Shellskripts in den Verzeichnissen `/etc` und `/sbin` verteilt.

2.3.5.2 Dämon mit Uhr (cron)

Im System waltet ein Dämon namens `cron(1M)`. Der schaut jede Minute[11] in `/var/spool/cron/crontabs` und `/var/spool/cron/atjobs` nach, ob zu dem jeweiligen Zeitpunkt etwas für ihn zu tun ist. Die Dateien in den beiden Verzeichnissen sind den Benutzern zugeordnet. In den `crontabs` stehen periodisch wiederkehrende Aufgaben, in den `atjobs` einmalige.

In die periodische Tabelle trägt man Aufräumungsarbeiten oder Datenübertragungen ein, die regelmäßig wiederkehrend vorgenommen werden sollen. In unserer Anlage werden beispielsweise jede Nacht zwischen 4 und 5 Uhr sämtliche Sitzungen abgebrochen, Dateien mit dem Namen `core` gelöscht, die `tmp`-Verzeichnisse geputzt und der Druckerspooler neu installiert. Das dient zum Sparen von Plattenplatz und dazu, daß morgens auch bei Abwesenheit der System-Manager die Anlage möglichst störungsfrei arbeitet. Für das Ziehen von Backup-Kopien wichtiger Dateien auf eine zweite Platte ist die Tabelle ebenfalls gut.

Jeder Benutzer, dem der System-Manager dies erlaubt hat, kann sich eine solche Tabelle anlegen, Einzelheiten siehe im Handbuch unter `crontab(1)`. Die Eintragungen haben folgende Form:

```
50 0 * * * exec /usr/bin/calendar
```

Das bedeutet: um 0 Uhr 50 an jedem Tag in jedem Monat an jedem Wochentag führe das Kommando `exec /usr/bin/calendar` aus. Für den Benutzer wichtiger ist die Tabelle der einmaligen Tätigkeiten. Mit dem Kommando `at(1)`, wieder die Erlaubnis des System-Managers vorausgesetzt, startet man ein Programm zu einem beliebigen späteren Zeitpunkt durch den Dämon `cron(1M)`. Der Aufruf (mehrzeilig!) sieht so aus:

```
at 2215 Aug 29
$HOME/program
control-d
```

In diesem Fall wird am 29. August um 22 Uhr 15 Systemzeit (also mitteleuropäische Sommerzeit) das Programm `program` aus dem Homeverzeichnis gestartet. Weitere Zeitformate sind möglich, siehe Handbuch unter `at(1)`. Mittels `at -l` erhält man Auskunft über seine at-Jobs.

Weiterhin kann man dem `cron` seinen **Terminkalender** anvertrauen. Jeden Morgen beim Anmelden erfährt man dann die Termine des laufenden und des kommenden Tages, wobei das Wochenende berücksichtigt wird. Um die Eingabe der Termine kommt man allerdings nicht herum, *de nihilo nihil* oder *Input ist aller Output Anfang*. Einzelheiten im Handbuch unter

[11] Die UNIX-Uhr zählt Sekunden seit dem 1. Januar 1970, 00:00 Uhr UTC und läuft bis 2038.

`calendar(1)`. Ein solcher **Reminder Service** kann im Netz zur Koordination von Terminen mehrerer Benutzer eingesetzt werden, `calendar(1)` ist jedoch zu schlicht dafür.

Das Kommando `leave(1)` ist ein **Wecker**. Mit `leave hh:mm` kann man sich 5 Minuten vor hh:mm Uhr aus seiner Bildschirmarbeit reißen lassen.

2.3.5.3 Line Printer Scheduler (lpsched)

Der Line-Printer-Scheduler-Dämon oder Druckerspooler `lpsched(1M)` verwaltet die Druckerwarteschlangen im System. Er nimmt Druckaufträge (request) entgegen, ordnet sie in die jeweilige Warteschlange ein und schickt sie zur rechten Zeit an die Drucker. Ohne seine ordnende Hand käme aus den Druckern viel Makulatur heraus. Es darf immer nur ein Druckerspooler laufen; zeigt die Prozessliste mehrere an, ist etwas schiefgegangen. Weiteres im Abschnitt 2.7.19 *Druckerausgabe* auf Seite 183. Netzfähige Drucker, die eine eigene IP-Adresse haben, sind nicht auf den `lpsched(1M)` angewiesen. Stattdessen laufen andere Dämonen wie die des HP Distributed Printing Systems (HPDPS) zur Verwaltung der Drucker und Warteschlangen.

2.3.5.4 Internet-Dämon (inetd)

Der Internet-Dämon `inetd(1M)` ist ein Türsteher, der ständig am Netz lauscht. Kommt von außerhalb eine Anfrage mittels `ftp(1)`, `lpr(1)`, `telnet(1)`, `rlogin(1)` oder ein Remote Procedure Call, wird er aktiv und ruft einen auf den jeweiligen Netzdienst zugeschnittenen Unterdämon auf, der die Anfrage bedient. Er ist ein Super-Server, der die eigentlichen Server-Prozesse nach Bedarf startet. Es darf immer nur ein Internet-Dämon laufen. Nicht jeder Netzdienst geht über den `inetd(1M)`, Email (SMTP) beispielsweise nicht.

2.3.5.5 Mail-Dämon (sendmail)

Email ist ein wichtiger Netzdienst. Ständig kommt Post herein oder wird verschickt. Die Verbindung des lokalen Mailsystems zum Netz stellt der `sendmail(1M)`-Dämon (Mail Transfer Agent) her, der wegen seiner Bedeutung unabhängig vom `inetd(1M)`-Dämon läuft. `sendmail(1M)` ist für seine nicht ganz triviale Konfiguration berüchtigt, die vor allem daher rührt, daß die Email-Welt sehr bunt ist. Es gibt eben nicht nur das Internet mit seinen einheitlichen Protokollen. Obwohl ein Benutzer unmittelbar mit `sendmail(1M)` arbeiten könnte, ist fast immer ein Dienstprogramm wie `mail(1)` oder `elm(1)` (Mail User Agent) vorgeschaltet.

2.3.6 Interprozess-Kommunikation (IPC)

2.3.6.1 IPC mittels Dateien

Mehrere Prozesse können auf dieselbe Datei auf dem Massenspeicher lesend und schreibend zugreifen, wobei es am Benutzer liegt, das Durcheinander in Grenzen zu halten. Dabei wird oft von sogenannten **Lock-Dateien** (engl. to lock = zuschließen, versperren) Gebrauch gemacht. Beipielsweise darf nur ein `elm(1)`-Prozess zum Verarbeiten der Email pro Benutzer existieren. Also schaut `elm(1)` beim Aufruf nach, ob eine Lock-Datei `/tmp/mbox.username` existiert. Falls nein, legt `elm(1)` eine solche Datei an und macht weiter. Falls ja, muß bereits ein `elm(1)`-Prozess laufen, und die weiteren Startversuche enden mit einer Fehlermeldung. Bei Beendigung des Prozesses wird die Lock-Datei gelöscht. Wird der Prozess gewaltsam abgebrochen, bleibt die Lock-Datei erhalten und täuscht einen `elm(1)`-Prozess vor. Die Lock-Datei ist dann von Hand zu löschen.

Die Kommunikation über Dateien erfordert Zugriffe auf den Massenspeicher und ist daher langsam. In obigem Fall spielt das keine Rolle, aber wenn laufend Daten ausgetauscht werden sollen, sind andere Mechanismen vorzuziehen.

2.3.6.2 Pipes

Man kann `stdout` eines Prozesses mit `stdin` eines weiteren Prozesses verbinden und das sogar mehrmals hintereinander. Eine solche Konstruktion wird **Pipe**[12], Pipeline oder Fließband genannt und durch den senkrechten Strich (ASCII-Nr. 124) bezeichnet:

```
cat filename | more
```

`cat(1)` schreibt die Datei `filename` in einem Stück nach `stdout`, `more(1)` sorgt dafür, daß die Ausgabe nach jeweils einem Bildschirm angehalten wird. `more(1)` könnte auf `cat(1)` verzichten und selbst die Datei einlesen (anders als in PC-DOS):

```
more filename
```

aber in Verbindung mit anderen Kommandos wie `ls(1)` ist die Pipe mit `more(1)` als letztem Glied zweckmäßig. Physikalisch ist eine Pipe ein Pufferspeicher im System, in den das erste Programm schreibt und aus dem das folgende Programm liest. Die Pipe ist eine Einbahnstraße. Das Piping in einer Sitzung wird von der Shell geleistet; will man es aus einem eigenen Programm heraus erzeugen, braucht man den Systemaufruf `pipe(2)`.

2.3.6.3 Named Pipe (FIFO)

Während die eben beschriebene Pipe keinen Namen hat und mit den beteiligten Prozessen lebt und stirbt, ist die **Named Pipe** eine selbständige Einrichtung. Ihr zweiter Name FIFO bedeutet *First In First Out* und kennzeichnet einen Speichertyp, bei dem die zuerst eingelagerten Daten auch als erste wieder herauskommen (im Gegensatz zum Stack, Stapel oder Keller, bei dem die zuletzt eingelagerten Daten als erste wieder herauskommen). Wir erzeugen im aktuellen Verzeichnis eine Named Pipe:

---

[12] Auf Vektorrechnern gibt es ebenfalls eine Pipe, die mit der hier beschriebenen Pipe nichts zu tun hat.

```
mknod mypipe p
```

(`mknod(1M)` liegt in `/bin`, `/sbin` oder `/etc`) und überzeugen uns mit `ls -l` von ihrer Existenz. Dann können wir mit:

```
who > mypipe &
cat < mypipe &
```

unsere Pipe zum Datentransport vom ersten zum zweiten Prozess einsetzen. Die Reihenfolge der Daten ist durch die Eingabe festgelegt, beim Auslesen verschwinden die Daten aus der Pipe (kein Kopieren). Die Pipe existiert vor und nach den beiden Prozessen und ist beliebig weiter verwendbar. Man wird sie mit `rm mypipe` wieder los.

2.3.6.4 Signale (kill, trap)

Ein Prozess kann niemals von außen beendet werden außer durch Abschalten der Stromversorgung. Er verkehrt mit seiner Umwelt einzig über rund dreißig **Signale**. Ihre Bedeutung ist im Anhang H *UNIX-Signale* auf Seite 327 nachzulesen oder im Handbuch unter `signal(2)`. Ein Prozess reagiert in dreierlei Weise auf ein Signal:

- er beendet sich (Default[13]) oder
- ignoriert das Signal oder
- verzweigt zu einem anderen Prozess.

Mit dem Kommando `kill(1)` (unglücklich gewählter Name) wird ein Signal an einen Prozess gesendet. Jedes der Kommandos

```
kill -s 15 4711
kill -s SIGTERM 4711
```

schickt das Signal Nr. 15 (SIGTERM) an den Prozess Nr. 4711 und fordert ihn damit höflich auf, seine Arbeit zu beenden und aufzuräumen. Das Signal Nr. 9 (SIGKILL) führt zum sofortigen Selbstmord des jeweiligen Prozesses. Mit der Prozess-ID 0 erreicht man alle Prozesse der Sitzung. Die Eingabe

```
kill -s 9 0
```

ist also eine etwas brutale Art, sich abzumelden. Mit `kill -l` erhält man eine Übersicht über die Signale mit Nummern und Namen, jedoch ohne Erklärungen.

Wie ein Programm bzw. Shellskript (was das ist, folgt in Abschnitt **??** *Shellskripte* auf Seite **??**) auf ein Signal reagiert, legt man in Shellskripten mit dem internen Shell-Kommando `trap` und in Programmen mit dem Systemaufruf `signal(2)` fest. Einige wichtige Signale wie Nr. 9 können nicht ignoriert oder umfunktioniert werden. Das `trap`-Kommando hat die Form

[13] Für viele Größen im System sind Werte vorgegeben, die solange gelten, wie man nichts anderes eingibt. Auch in Anwenderprogrammen werden solche Vorgaben verwendet. Sie heißen Defaultwerte, wörtlich Werte, die für eine fehlende Eingabe einspringen.

```
trap "Kommandoliste" Signalnummer
```

Empfängt die das Skript ausführende Shell das Signal mit der jeweiligen Nummer, wird die Kommandoliste ausgeführt. Das `exit`-Kommando der Shell wird als Signal Nr. 0 angesehen, so daß man mit

```
trap "echo Arrivederci; exit" 0
```

in der Datei `/etc/profile` die Sitzungsshell zu einem freundlichen Abschied veranlassen kann. Das nackte `trap`-Kommando zeigt die gesetzten Traps an. Ein Beispiel für den Gebrauch von Signalen in einem Shellskript namens `traptest`:

```
trap "print Abbruch durch Signal; exit" 15
trap "print Lass den Unfug!" 16
while :
do
sleep 1
done
```

Quelle 2.1 : Shellskript traptest mit Signalbehandlung

Setzen Sie die Zugriffsrechte mit `chmod 750 traptest`. Wenn Sie das Shellskript mit `traptest` im Vordergrund starten, verschwindet der Prompt der Sitzungsshell, und Sie können nichts mehr eingeben, weil `traptest` unbegrenzt läuft und die Sitzungsshell auf das Ende von `traptest` wartet. Allein mit der Break-Taste (Signal 2) werden Sie `traptest` wieder los. Starten wir das Shellskript mit `traptest &` im Hintergrund, kommt der Prompt der Sitzungsshell sofort wieder, außerdem erfahren wir die PID der Shell, die `traptest` abarbeitet, die PID merken! Mit `ps -f` sehen wir uns unsere Prozesse an und finden den `sleep`-Prozess aus dem Shellskript. Schicken wir nun mit `kill -16 PID` das Signal Nr. 16 an die zweite Shell, antwortet sie mit der Ausführung von `print Lass den Unfug!`. Da das Shellskript im Hintergrund läuft, kommt möglicherweise vorher schon der Prompt der Sitzungsshell wieder. Schicken wir mit `kill -15 PID` das Signal Nr. 15, führt die zweite Shell die Kommandos `print Abbruch durch Signal; exit` aus, das heißt sie verabschiedet sich. Auch hier kann der Sitzungsprompt schneller sein.

Wenn ein Prozess gestorben ist, seine Leiche aber noch in der Prozesstabelle herumliegt, wird er **Zombie** genannt. Zombies sollen nicht auf Dauer in der Prozesstabelle auftauchen. Notfalls booten.

Die weiteren Mittel zur Kommunikation zwischen Prozessen gehören in die System- und Netzprogrammierung, sie gehen über den Rahmen dieses Buches hinaus. Wir erwähnen sie kurz, um Ihnen Stichwörter für die Suche in der Literatur an die Hand zu geben.

2.3.6.5 Nachrichtenschlangen

Nachrichtenschlangen (message queue) sind keine erdgebundene Konkurrenz der Brieftauben, sondern im Systemkern gehaltene verkettete Listen mit jeweils einem Identifier, deren Elemente kurze Nachrichten sind, die durch Typ, Länge und Inhalt gekennzeichnet sind. Auf die Listenelemente kann in beliebiger Folge zugegriffen werden.

2.3.6.6 Semaphore

Semaphore[14] sind Zählvariablen im System, die entweder nur die Werte 0 und 1 oder Werte von 0 bis zu einem systemabhängigen n annehmen. Mit ihrer Hilfe lassen sich Prozesse synchronisieren. Beispielsweise kann man Schreib- und Lesezugriffe auf dieselbe Datei mit Hilfe eines Semaphores in eine geordnete Abfolge bringen (wenn ein Prozess schreibt, darf kein anderer lesen oder schreiben).

2.3.6.7 Gemeinsamer Speicher

Verwenden mehrere Prozesse denselben Bereich des Arbeitsspeichers zur Ablage ihrer gemeinsamen Daten, so ist das der schnellste Weg zur Kommunikation, da jeder Kopiervorgang entfällt. Natürlich muß auch hierbei für Ordnung gesorgt werden.

Shared Memory ist nicht auf allen UNIX-Systemen verfügbar. Man probiere folgende Eingabe, die die Manualseite zu dem Kommando `ipcs(1)` (Interprocess Communication Status) erzeugt:

```
man ipcs
```

Bei Erfolg dürften Nachrichtenschlangen, Semaphore und Shared Memory eingerichtet sein.

2.3.6.8 Sockets

Sockets sind ein Mechanismus zur Kommunikation zwischen zwei Prozessen auf derselben oder auf vernetzten Maschinen in beiden Richtungen. Die Socket-Schnittstelle besteht aus einer Handvoll Systemaufrufe, die von Benutzerprozessen in einheitlicher Weise verwendet werden. Dem Programmierer stellen sich Sockets wie Dateien dar: Er öffnet mittels eines Systemaufrufes einen Socket, erhält einen Socket-Deskriptor zurück und schreibt nach dort oder liest von dort. Ist er fertig, schließt er den Socket. Unter den Sockets liegen die Protokollstapel, das heißt die Programmmodule, die die Daten entsprechend den Schichten eines Netzprotokolls aufbereiten, und schließlich die Gerätetreiber für die Netzkarten oder sonstige Verbindungen.

2.3.6.9 Streams

Ein **Stream** ist eine Verbindung zwischen einem Prozess und einem Gerätetreiber zum Austausch von Daten in beiden Richtungen (vollduplex). Der Gerätetreiber braucht nicht zu einem physikalischen Gerät (Hardware) zu

[14]In Cuxhaven steht ein großer Semaphor, der den auslaufenden Schiffen die Windverhältnisse bei Borkum und Helgoland übermittelt.

führen, sondern kann auch ein Pseudotreiber sein, der nur bestimmte Funktionen zur Verfügung stellt. In den Stream lassen sich nach Bedarf dynamisch Programmmodule zur Bearbeitung der Daten einfügen, beispielsweise um sie einem Netzprotokoll anzupassen (was bei Sockets nicht möglich ist). Das Streams-Konzept erhöht die Flexibilität der Gerätetreiber und erlaubt die mehrfache Verwendung einzelner Module auf Grund genau spezifizierter Schnittstellen.

Die Terminalein- und -ausgabe wird in neueren UNIXen mittels Streams verwirklicht. Auch Sockets können durch Streams nachgebildet (emuliert) werden ebenso wie die Kommunikation zwischen Prozessen auf derselben Maschine oder auf Maschinen im Netz.

2.3.7 Begriffe Prozesse

Folgende Begriffe sollten klarer geworden sein:

- Dämon
- Elternprozess, Kindprozess, Vererbung
- Interprozess-Kommunikation
- Priorität
- Prozess
- Signal
- Vordergrund, Hintergrund

Folgende Kommandos sollten beherrscht werden:

- `ps`
- `kill`
- `nice`
- `nohup ...&`

2.3.8 Memo Prozesse

- Ein Prozess ist die Form, in der ein Programm ausgeführt wird. Er liegt im Arbeitsspeicher und verlangt Prozessorzeit.
- Ein Prozess wird erzeugt durch den manuellen oder automatischen Aufruf eines Programmes (Ausnahme Prozess Nr. 0).
- Ein Prozess endet entweder auf eigenen Wunsch (wenn seine Arbeit fertig ist) oder infolge eines von außen kommenden Signals.
- Prozesse können untereinander Daten austauschen. Der einfachste Weg ist eine Pipe (Einbahnstraße).
- Das Kommando `ps(1)` mit verschiedenen Optionen zeigt die Prozessliste an.
- Mit dem Kommando `kill` wird ein Signal an einen Prozess geschickt. Das braucht nicht unbedingt zur Beendigung des Prozesses zu führen.

2.3.9 Übung Prozesse

Suchen Sie sich ein freies Terminal. Melden Sie sich als `gast` oder `guest` an. Ein Passwort sollte dazu nicht erforderlich sein. Falls Sie schon als Benutzer auf der Anlage eingetragen sind, verwenden Sie besser Ihren Benutzernamen samt Passwort. Bei Schwierigkeiten wenden Sie sich an den System-Manager.

Groß- und Kleinbuchstaben sind in UNIX verschiedene Zeichen. Auch die Leertaste (space) ist ein Zeichen. Bei rechtzeitig (vor dem Tippen von RETURN oder ENTER) bemerkten Tippfehlern geht man mit der Taste BS oder BACKSPACE zurück – nur nicht beim Anmelden, da muß alles stimmen.

Nach der Eingabe eines Kommandos ist immer die RETURN- oder ENTER-Taste zu betätigen. Hierauf wird im folgenden nicht mehr hingewiesen. Erst mit dieser Taste wird das Kommando wirksam. Man kann allerdings Programme so schreiben, daß sie auf die Eingabe *eines* Zeichens ohne RETURN antworten (siehe `curses(3)`).

Lesen Sie während der Sitzung im Referenz-Handbuch die Bedeutung der Kommandos nach. Die Optionen unterscheiden sich gelegentlich bei den verschiedenen UNIXen. Wenn der Prompt (Dollarzeichen) kommt, ist der Computer bereit zum Empfangen neuer Kommandos. Geben Sie nacheinander folgende Kommandos ein, warten Sie die Antwort des Systems ab:

`who`	(wer arbeitet zur Zeit?)
`who -H`	(wer arbeitet zur Zeit?)
`who -a`	(wer arbeitet zur Zeit?)
`who -x`	(falsche Option)
`id`	(wer bin ich?)
`whoami`	(wer bin ich?)
`date`	(Datum und Uhrzeit?)
`zeit`	(lokales Kommando)
`leave hhmm`	(hhmm 10 min nach jetzt eingeben)
`cal`	(Kalender)
`cal 12 2000`	(die letzten Tage des Jahrtausends)
`cal 9 1752`	(Geschichte sollte man können)
`tty`	(mein Terminal?)
`pwd`	(Arbeitsverzeichnis?)
`ls`	(Inhalt des Arbeitsverzeichnisses?)
`man ls`	(Handbucheintrag zu ls)
`ps -ef`	(verfolgen Sie die Ahnenreihe vom Prozess Nr.1 – `init` – bis zu Ihrem neuesten Prozess `ps -ef`)
`ps -elf`	(noch mehr Informationen)
`sh`	
`sh`	
`ps -f`	(wieviele Shells haben Sie nun?)

`date`	
`ps`	
`exec date`	
`ps`	
`exec date`	
`ps`	
`exec date`	(Damit ist Ihre erste Shell weg, warum?)
wieder anmelden	
`ps`	(PID Ihrer Shell merken)
`kill PID`	(Das reicht nicht)
`kill -9 PID`	(Shell wieder weg)
erneut anmelden	
`sh`	
`PS1="XXX "`	(neuer Prompt)
`set`	
`exec date`	
`set`	(Erbfolge beachten)
`nice -19 date`	(falls Betrieb herrscht, warten Sie lange auf die Zeit)
`ls -l &`	(Prompt kommt sofort wieder, samt PID von `ls`)
`exit`	(abmelden)

2.3.10 Fragen Prozesse

- Was ist ein Prozess?
- Was bedeutet die Ausgabe des Kommandos `ps`?
- Was macht das Kontrollterminal?
- Wie wird ein Prozess erzeugt? (Eltern-/Kind-Prozess)
- Was ist ein synchroner (asynchroner) Prozess?
- Wie kann man einen Prozess von der Sitzung abkoppeln?
- Wie läßt sich die Priorität eines Prozesses beeinflussen? Was bedeutet sie?
- Was ist ein Dämon? Nennen Sie einige Dämonen.
- Was ist eine Pipe?
- Was macht das Kommando `kill`?

2.4 Daten in Ruhe: Dateien

2.4.1 Dateiarten

Eine **Datei** (E: file, F: fichier) ist eine Menge zusammengehöriger Daten, auf die mittels eines Dateinamens zugegriffen wird. Ein treffender Vergleich ist ein Buch, das unter einem Titel in einem Einband eine Menge zusammengehörender Sätze und Abbildungen vereint. Genau wie ein Buch lässt sich auch eine Datei transportieren, kopieren, katalogisieren, verlieren oder vernichten. Das englische Wort *file* geht auf lateinisch *filum* = Draht zurück und bezeichnete früher eine auf einem Draht aufgereihte Sammlung von Schriftstücken. Heute sind Bytes in einer Datei aufgereiht. Man kann eine Datei als einen Datentyp höherer Ordnung auffassen. Die Struktur ist in der Datei enthalten oder wird durch das schreibende bzw. lesende Programm einem an sich strukturlosen **Zeichenstrom** (byte stream) aufgeprägt. In UNIX ist das letztere der Fall.

Selbst auf einer kleinen Anlage kommen schnell einige tausend Dateien zusammen. Ohne ein Ordnungssystem findet man nichts wieder. Drei Systeme zur Strukturierung umfangreicher Datenmengen sind verbreitet:

- hierarchische Datei-Systeme,
- relationale Systeme (Tabellen),
- objektorientierte Systeme.

UNIX-Datei-Systeme sind hierarchisch in Form eines Baumes – Wurzel oben – organisiert; alle drei Möglichkeiten werden in Datenbanken verwendet. Wie diese logische, dem Benutzer sichtbare Struktur physikalisch auf dem Massenspeicher (Platte, Band, CD) verwirklicht wird, ist eine zweite Frage.

In einem UNIX-Datei-System kommen im wesentlichen drei Arten von Dateien vor:

- gewöhnliche Dateien (normale Datei, regular file, fichier regulair),
- Verzeichnisse (Katalog, directory, data set, folder, répertoire),
- Geräte-Dateien (special device file, fichier spécial).

Gewöhnliche Dateien enthalten Meßdaten, Texte, Programme, Grafiken, Sound. **Verzeichnisse** sind Listen von Dateien, die wiederum allen drei Gruppen angehören können. **Gerätefiles** sind eine Besonderheit von UNIX, das periphere Geräte (Laufwerke, Terminals, Drucker, mittlerweile auch Prozesse und Netzverbindungen) formal als Dateien ansieht. Die gesamte Datenein- und -ausgabe (E/A, englisch I/O für Input/Output, französisch entrée/sortie) erfolgt über einheitliche Schnittstellen. Alle Gerätefiles finden sich im Geräteverzeichnis `/dev`, das insofern eine Sonderstellung einnimmt (*device* = Gerät).

Darüber hinaus unterscheidet man bei gewöhnlichen Dateien noch zwischen **lesbaren** und **binären** Dateien. Lesbare Dateien (text file) enthalten nur lesbare ASCII-Zeichen und können auf Bildschirm oder Drucker ausgegeben werden. Binäre Dateien (binary) enthalten ausführbaren (kompilierten) Programmcode, Grafiken oder gepackte Daten und sind nicht lesbar. Der

Versuch, sie auf dem Bildschirm darzustellen oder sie auszudrucken, führt zu sinnlosen Ergebnissen und oft zu einem Blockieren des Terminals oder erheblichem Papierverbrauch. Dem Betriebssystem ist das wurscht; intelligente Lese- oder Druckprogramme unterscheiden beide Arten und weigern sich, binäre Dateien zu verarbeiten. Auch bei Übertragungen im Netz wird zwischen ASCII-Dateien und binären Dateien unterschieden.

2.4.2 Datei-System – Sicht von unten

Alle Daten sind auf dem Massenspeicher in einem **Datei-System** abgelegt, wobei auf einer Platte ein oder mehrere Datei-Systeme eingerichtet sind. In modernen Anlagen kann ein Datei-System auch über mehrere Platten gehen (spanning, logical volumes, RAID). Jedes UNIX-Datei-System besteht aus einem Boot-Block am Beginn der Platte (Block 0), einem Super-Block, einer Inode-Liste und dann einer Vielzahl von Datenblöcken, siehe Abbildung 2.3 auf Seite 55.

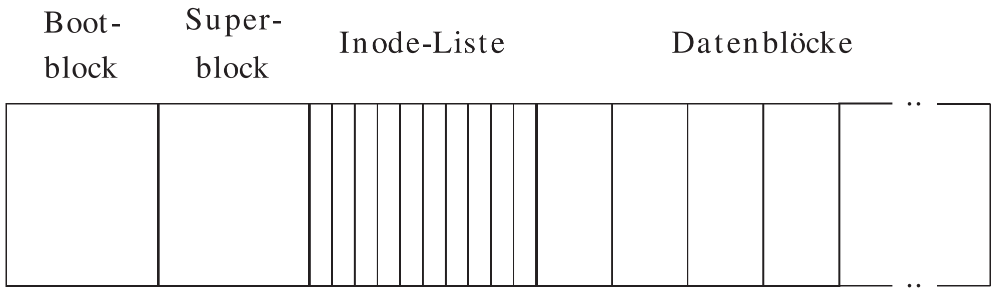

Abb. 2.3: UNIX-Datei-System, Sicht von unten

Der **Boot-Block** enthält Software zum Booten des Systems und muss der erste Block im Datei-System sein. Er wird nur im `root`-System gebraucht – also einmal auf der ganzen Anlage – ist aber in jedem Datei-System vorhanden. Er wird beim Einrichten des Datei-Systems mittels der Kommandos `mkfs(1M)` oder `newfs(1M)` angelegt.

Der **Super-Block** wird ebenfalls bei der Einrichtung des Datei-Systems geschrieben und enthält Informationen zu dem Datei-System als Ganzem wie die Anzahl der Datenblöcke, die Anzahl der freien Datenblöcke und die Anzahl der Inodes. Die Anzahl der Inodes im Datei-System ist begrenzt; die Grenze wird bei der Einrichtung festgelegt und lässt sich nachträglich kaum noch verschieben. Deshalb gut überlegen, falls man mit ungewöhnlich vielen, kleinen Dateien rechnen muss. Etwas anderes ist die Größe des Datei-Systems in Mega- oder Gigabytes. Hier kann man nachträglich erweitern, solange noch Luft auf der Platte ist, ausgenommen beim root-Datei-System.

Die **Inode-Liste** enthält alle Informationen zu den einzelnen Dateien außer den Datei-Namen. Zu jeder Datei gehört eine Inode mit einer im Datei-System eindeutigen **Nummer**, die vom System vergeben wird. Einzelheiten siehe Abschnitt 2.4.8 *Inodes und Links* auf Seite 74. Der Name wird einer

Datei sprich einer Inode-Nummer durch den Benutzer zugeordnet und steht im zuständigen Verzeichnis. Die Dateien selbst bestehen nur aus den Daten.

Der **Daten-Block** ist die kleinste Einheit, in der blockorientierte Geräte – vor allem Platten – Daten schreiben oder lesen. In den Daten-Blöcken sind die Dateien einschließlich der Verzeichnisse untergebracht. Die Blockgröße beträgt 512 Bytes oder ein Vielfaches davon. Große Blöcke erhöhen die Schreib- und Lesegeschwindigkeit, weil mit einem Plattenzugriff viele Bytes geschrieben oder gelesen werden, verschwenden aber bei kleinen Dateien Speicherplatz, weil jeder Datei mindestens ein Block zugeordnet wird. Bei den heutigen Preisen für Festplatten kann man es sich aber erlauben, von der Default-Blockgrösse (Linux: 1024 Bytes) nach oben abzuweichen. Typische Kandidaten für große Blöcke sind die Partitionen für temporäre Dateien oder Webseiten.

Ein Datei-System kann logisch beschädigt werden, weil ein logischer Schreibvorgang physikalisch auf mehrere, zeitlich auseinander liegende Plattenzugriffe verteilt sein kann. Wird während des logischen Schreibens der Strom abgeschaltet, ist das Datei-System nicht mehr konsistent. Zur Reparatur eines logisch beschädigten Datei-Systems stellt UNIX Werkzeuge bereit, vor allem `fsck(1M)`. Dieses Werkzeug geht das gesamte Datei-System durch und beseitigt Inkonsistenzen. Unter Umständen gehen dabei Daten verloren, aber das Datei-System arbeitet wenigstens wieder. Bei den heutigen großen Platten und Datei-Systemen dauert der File System Check mittels `fsck(1M)` lange. Eine bessere Lösung sind **Journaled File Systems**, die Techniken verwenden, wie sie Datenbanken einsetzen. Die haben nämlich die gleichen Konsistenzprobleme.

Hierzu wird jeder Schreibvorgang protokolliert. Erst wenn der erfolgreiche Abschluss eines Schreibvorgangs zurückgemeldet wird, wird der Eintrag im Protokoll gelöscht. Bleiben in Folge eines Stromausfalls unvollendete Schreibvorgänge übrig, werden sie bei der Reparatur mit Hilfe des Protokolls zurückgenommen. Auf diese Weise wird relativ schnell die Konsistenz und damit die Arbeitsfähigkeit eines Datei-Systems wieder hergestellt. Einzelheiten findet man für Linux-Datei-Systeme unter IBM-JFS, SGI-XFS oder ReiserFS.

Es gibt weitere Datei-Systeme, wobei mit größer werdenden Massenspeichern neben der Sicherheit die Zugriffsgeschwindigkeit eine immer wichtigere Rolle spielt. Das Datei-System ist für die Leistung einer Maschine mit durchschnittlichen Aufgaben genauso entscheidend wie die CPU. Der Benutzer sieht von den Unterschieden der Datei-Systeme jedoch fast nichts, er arbeitet nur mit der im folgenden Abschnitt erläuterten Baumstruktur.

2.4.3 Datei-System – Sicht von oben

Selbst auf einer kleinen Anlage kommt man leicht auf zehntausend Dateien. Da muss man Ordnung halten, wie in einer Bibliothek. Unter UNIX werden zum Benutzer hin alle Dateien in einer Datei-Hierarchie, einer Baumstruktur angeordnet, siehe Abbildung 2.4 auf Seite 60. An der Spitze steht ein

Verzeichnis namens `root`[15], das nur mit einem Schrägstrich bezeichnet wird. Es wird auch **Wurzel-Verzeichnis** genannt, da in ihm der ganze Dateibaum wurzelt. Dieses Verzeichnis enthält einige gewöhnliche Dateien und vor allem weitere Verzeichnisse. Die Hierarchie kann viele Stufen enthalten, der Benutzer verliert die Übersicht eher als der Computer.

Die Datei-Hierarchie hat sich im Lauf der Jahrzehnte etwas gewandelt, vor allem infolge der zunehmenden Vernetzung. Bei der Einteilung spielen folgende Gesichtspunkte eine Rolle:

- Statische, sich kaum ändernde Daten sollen von sich häufig ändernden Daten getrennt werden (Backup-Häufigkeit).
- Das Betriebssystem soll von den Anwendungsprogrammen getrennt werden (Betriebssicherheit).
- Rein lokale (private, maschinenspezifische) Daten sollen von Daten getrennt werden, die alle Mitglieder eines Computerverbundes (Cluster) gemeinsam nutzen (shared files).
- Konfigurationsdaten sollen von den ausführbaren Programmen getrennt werden, damit sie ein Update überstehen.
- Konfigurationsdaten sollen von temporären, nicht lebensnotwendigen Daten wie Protokollfiles getrennt werden.
- Daten, die im Betrieb stark wachsen (Home-Verzeichnisse, Protokollfiles, Spool-Dateien (Warteschlangen), zusätzliche Anwendungen), sollen von Daten getrennt werden, die erfahrungsgemäß kaum wachsen. Partitionen für solche Daten müssen mit viel Spielraum angelegt werden.

Die statischen Daten sind in einem Verbund großenteils gemeinsam nutzbar, so dass diese beiden Gesichtspunkte zur selben Einteilung führen, die an oberster Stelle kommt. In diesem Bereich dürfen die Benutzer gar nichts und sollten die Verwalter möglichst wenig ändern. Reale Datei-Systeme berücksichtigen die Punkte mehr oder weniger – aus historischen und lokalen Gründen – aber im wesentlichen gleichen sich alle UNIX-Datei-Systeme einer Generation.

Unter der Wurzel des Dateibaumes, dem `root`-Verzeichnis, finden sich folgende Verzeichnisse:

- Statische, gemeinsam nutzbare Verzeichnisse
  - `bin` (UNIX-Kommandos)
  - `lib` (Bibliotheken)

[15] `root` ist der Name des obersten Verzeichnisses und zugleich der Name des System-Verwalters oder -Managers. Das Verzeichnis `root` ist genau genommen gar nicht vorhanden, sondern bezeichnet eine bestimmte Adresse auf dem Massenspeicher, auf der der Dateibaum beginnt. Für den Benutzer scheint es aber wie die anderen Verzeichnisse zu existieren. Während alle anderen Verzeichnisse in ein jeweils übergeordnetes Verzeichnis eingebettet sind, ist `root` durch eine Schleife sich selbst übergeordnet.

  - opt (optionale Anwendungen, Compiler, Editoren)
  - sbin (wichtige Kommandos, beim Systemstart benötigt),
  - usr (Fortsetzung von bin, Kommandos, Bibliotheken, Dokumentation)

- Lokale Verzeichnisse
  - boot (UNIX-Kern, Boot Loader, siehe auch stand)
  - dev (Gerätefiles)
  - etc (wichtige Konfigurationsfiles, früher auch Kommandos zur Systemverwaltung)
  - home auch homes, früher users, Home-Verzeichnisse der Benutzer)
  - lost+found (Fundbüro, in jeder Partition, wird von fsck(1m) gebraucht)
  - mnt (Mounting Point, zum Beispiel für CDs)
  - stand (UNIX-Kern, Boot Loader; verkürzt aus standalone)
  - tmp (temporäre Dateien)
  - var (Dateien stark veränderlicher Größe: Protokolle, Briefkästen, Spooler, temporäre Dateien)

Die Anwendungen unter /opt sollen ihrerseits eine Verzeichnisstruktur wie unter /usr mitbringen: bin, lib, share, man usw.

Ein **Mounting Point** oder Mountpoint ist ein leeres Verzeichnis, in das die Wurzel eines weiteren Datei-Systems eingehängt werden kann. Auf diese Weise lässt sich der ursprüngliche Dateibaum nahezu unbegrenzt erweitern. Auf unseren Anlagen haben wir die Home-Verzeichnisse der Mitarbeiter und der Studenten auf jeweils eigenen Platten und Datei-Systemen untergebracht, die in durchnumerierte Mounting Points eingehängt werden. Das trägt wesentlich zur Flexibilität und zur Betriebsssicherheit bei und ist eine Voraussetzung für die Zusammenfassung mehrerer Maschinen zu einem Cluster.

Das Verzeichnis /usr enthält seinerseits zahlreiche Unterverzeichnisse, die für das System wichtig sind:

- adm (Systemverwaltung, Accounting, heute in /var/adm oder /var/log)
- bin (UNIX-Kommandos)
- conf (UNIX-Kern-Konfiguration)
- contrib (Beiträge des Computerherstellers)
- include (Header-Dateien)
- lbin (spezielle Kommandos und Dämonen)
- lib (Bibliotheken und Kommandos)

- `local` (lokale Kommandos)
- `mail` (Mailsystem, heute in `/var/mail` oder `/var/spool/mail`)
- `man` (Referenz-Handbuch, heute in `/usr/share/man`)
- `newconfig` (Default-Konfigurationen des Systems)
- `news` (News, Nachrichten an alle, heute in `/var/news` `/var/spool/news`)
- `sbin` (Kommandos zur Systemverwaltung)
- `share` (in einem Cluster gemeinsam nutzbare Dateien)
- `spool` (Drucker-Spoolsystem, `cron`-Dateien, heute in `/var/spool`)
- `tmp` (weitere temporäre Dateien, heute in `/var/tmp`)

und weitere Unterverzeichnisse für spezielle Zwecke. Im Abschnitt G *UNIX-Datei-System* des Anhangs auf Seite 325 wird die Datei-Hierarchie eingehender dargestellt. Die Einzelheiten ändern sich im Lauf der Jahre und von Hersteller zu Hersteller, aber die Konzepte bleiben. Ob die Briefkästen in `/mail`, `/var/mail` oder `/var/spool/mail` liegen, ist nebensächlich und kann mittels Soft Links an die jeweiligen Wünsche angepasst werden, aber dass jeder Benutzer eine Briefkasten-Datei hat, an dem nur er und das Mail-System Rechte haben, ist ein in der UNIX-Welt stabiles Konzept.

Die Eintragungen im Geräte-Verzeichnis `/dev` weichen von denen in anderen Verzeichnissen ab. Sie enthalten zusätzlich Angaben über den Treiber und den I/O-Port, an den das jeweilige Gerät angeschlossen ist. Dasselbe Gerät kann am selben Port mit anderem Treiber unter einem anderen Namen erscheinen. Insbesondere erscheinen Massenspeicher einmal als blockorientiertes und einmal als zeichenorientiertes Gerät. Blockorientierte Geräte übertragen nicht einzelne Zeichen, sondern Blöcke von 512 oder mehr Zeichen. Die Namen sind zwar beliebig, es haben sich aber gewisse Regeln eingebürgert:

- `console` Konsol-Terminal des Systems
- `ct` Cartridge Tape als Block Device
- `dsk` Platte als Block Device (blockorientiert)
- `lan` Netz-Interface
- `lp` Drucker (line printer)
- `mt` Magnetband als Block Device
- `null` Papierkorb (bit bucket)
- `pty` Pseudo-Terminal
- `rct` Cartridge Tape als Character (raw) Device
- `rdsk` Platte als Character Device (raw, zeichenorientiert)
- `rmt` Magnetband als Character Device
- `tty` Kontroll-Terminal eines Prozesses

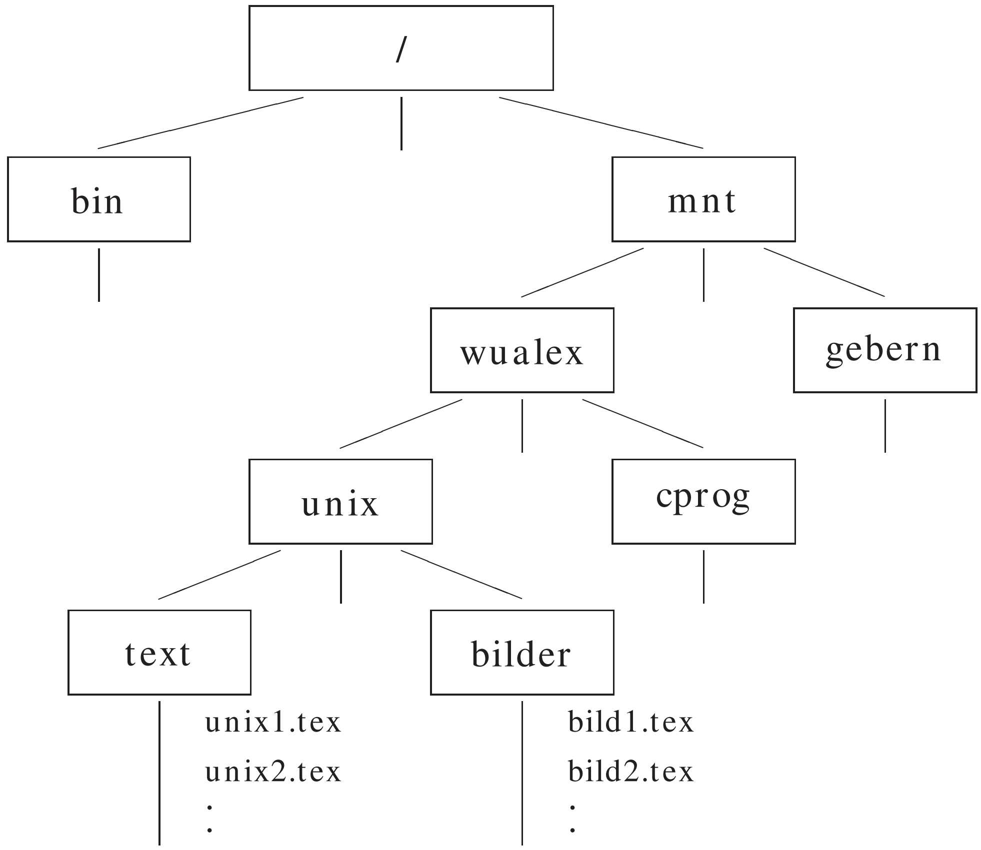

Abb. 2.4: UNIX-Datei-Hierarchie

- tty1p2 Terminal an Multiplexer 1, Port 2

Bei umfangreicher Peripherie ist das /dev-Verzeichnis in Unterverzeichnisse gegliedert. Beim Schreiben nach /dev/null verschwinden die Daten unwiederbringlich in einem Schwarzen Loch im Informationsraum, das Lesen aus /dev/null liefert ein EOF-Zeichen (end of file).

Beim Arbeiten mit Verzeichnissen werden vier Begriffe immer wieder gebraucht:

- das root- oder Wurzelverzeichnis, bereits erläutert,
- das Login-Verzeichnis,
- das Home-Verzeichnis,
- das aktuelle oder Arbeits-Verzeichnis.

Nach der Anmeldung (Login) landet der Benutzer in seinem **Login-Verzeichnis**, wie es in der Datei /etc/passwd festgelegt ist. Das Login-Verzeichnis ist fast immer, aber nicht notwendig identisch mit dem Home-Verzeichnis. Benutzer mit minderen Rechten bekommen mitunter nur das Verzeichnis /tmp oder /dev/null zugewiesen.

In seinem **Home-Verzeichnis** (Heimatverzeichnis, Hauptkatalog, E: home directory, F: répertoire principal) darf ein Benutzer nach Herzenslust Dateien und Unterverzeichnisse anlegen und löschen, Rechte vergeben und entziehen, Ordnung halten oder nicht. Bei gewöhnlichen Benutzern wird das Home-Verzeichnis als Login-Verzeichnis in /etc/passwd eingetragen. Einem Benutzernamen kann nur ein Home-Verzeichnis zugeordnet werden. Die Homes der gewöhnlichen Benutzer sind in einem Verzeichnis /home oder /homes zusammengefasst, das bei größeren Installationen in einer eigenen Partition liegt und in das Wurzelverzeichnis gemountet wird. Das Home des Systemverwalters root liegt als /root direkt im Wurzelverzeichnis.

Das Home-Verzeichnis ist zu Beginn einer Sitzung das **Arbeits-Verzeichnis** oder aktuelle Verzeichnis (E: working directory, F: répertoire courant, répertoire de travail), dessen Dateien und Unterverzeichnisse unmittelbar über ihren Namen ohne die beim Wurzelverzeichnis root beginnende Verzeichniskette verfügbar sind. Man kann jedes Unterverzeichnis seines Home-Verzeichnisses vorübergehend zum Arbeits-Verzeichnis machen, auch andere Verzeichnisse, sofern dazu berechtigt. Mit cd(1) wechselt man das Arbeits-Verzeichnis. Das Kommando pwd(1) (print working directory) nennt das Arbeits-Verzeichnis, falls man die Orientierung verloren hat.

Nach einer Faustregel soll man ein Verzeichnis weiter unterteilen, wenn es mehr als wenige hundert Eintragungen enthält. Verzeichnisnamen sollten nicht extrem lang gewählt werden. Die Namen der Dateien und Unterverzeichnisse in einem Verzeichnis sollten sich möglichst schon in den vorderen Stellen unterscheiden. Das alles trägt zur Geschwindigkeit bei, da bei Zugriffen auf eine Datei die übergeordneten Verzeichnisse sequentiell (linear) durchsucht werden.

Ihr Home-Verzeichnis sollten Sie auch sofort in Unterverzeichnisse unterteilen, sonst macht sich schnell das Chaos breit. Wenn Sie Besitzer einiger tausend Dateien sind – und das wird man schnell – kennen Sie nicht mehr jede Datei.

Der **Name** einer Datei wird entweder **absolut** angegeben, ausgehend von `root`. Dann beginnt er mit dem Schrägstrich und wird auch **Pfad** (path, chemin d'accès) genannt. Oder er wird **relativ** zum augenblicklichen Arbeits-Verzeichnis nur mit seinem letzten Namensteil (basename) angegeben:

```
/mnt/wualex/buch/unix/einleitung/vorwort.tex
vorwort.tex
```

Das Kommando `basename(1)` verkürzt einen absoluten Namen auf seinen letzten Namensteil und wird in Shellskripten gebraucht. Umgekehrt zieht das Kommando `dirname(1)` aus einem absoluten Datei-Namen alle Vorderglieder (= Namen von Verzeichnissen) heraus.

Datei-Namen dürfen 255 Zeichen[16] lang sein und sollen nur Buchstaben (keine Umlaute), Ziffern sowie die Zeichen Unterstrich, Bindestrich oder Punkt enthalten. Es ist üblich, Kleinbuchstaben zu verwenden, Großbuchstaben nur für Namen, die auffallen sollen (`README`). Die Verwendung von TAB, Backspace, Space, Stern, ESC und dergleichen ist nicht verboten, führt aber zu lustigen Effekten. Verboten sind nur der Schrägstrich, der als Trennzeichen in der Pfadangabe dient, und das Zeichen ASCII-Nr. 0, das einen String abschließt. Datei-Namen sollten mindestens vier Zeichen lang sein, um die Gefahr einer Verwechslung mit UNIX-Kommandos zu verringern. Will man sichergehen, dass ein Datei-Name auch in anderen Welten unverändert gültig ist, darf er nur Großbuchstaben (ohne Umlaute etc.), Ziffern, den Unterstrich und den Punkt enthalten. Er muss ferner der 8.3-Regel von DOS genügen, also höchstens acht Zeichen gefolgt von einem Punkt und nochmal drei Zeichen umfassen. Innerhalb eines Verzeichnisses darf ein Name nur einmal vorkommen (wie in Familien); der gleiche Name in verschiedenen Verzeichnissen benennt verschiedene Dateien, zum Beispiel `/bin/passwd(1)` und `/etc/passwd(4)`. Bei Shellkommandos, die Datei-Namen als Argument benötigen, kann man in den Datei-Namen **Jokerzeichen** verwenden, siehe Abschnitt 2.5.1.1 *Kommandointerpreter* auf Seite 89.

Die Verwendung von **Namenserweiterungen** (`file.config`, `file.dat`, `file.bak`) oder **Kennungen** (extension) ist erlaubt, aber nicht so verbreitet wie unter DOS. Programme im Quellcode bekommen eine Erweiterung (.c für C-Quellen, .f für FORTRAN-Quellen, .p für PASCAL-Quellen), ebenso im Objektcode (.o). Der Formatierer LaTeX verwendet auch Erweiterungen. Es dürfen Kennungen mit mehr als drei Zeichen oder eine Kette von Kennungen verwendet werden wie `myprogram.c.backup.old`. Sammlungen gebräuchlicher Kennungen finden sich im Netz.

Das jeweilige Arbeits-Verzeichnis wird mit einem Punkt bezeichnet, das unmittelbar übergeordnete Verzeichnis mit zwei Punkten (wie in DOS). Das Kommando `cd ..` führt also immer eine Stufe höher in der Datei-Hierarchie. Mit `cd ../..` kommt man zwei Stufen höher. Will man erzwingen, dass ein

[16] Ältere UNIX-Systeme erlauben nur 14 Zeichen.

Kommando aus dem Arbeitsverzeichnis ausgeführt wird und nicht ein gleichnamiges aus `/bin`, so stellt man Punkt und Schrägstrich voran wie bei `./cmd`.

Beim Arbeiten im Netz ist zu bedenken, dass die Beschränkungen der Datei-Namen von System zu System unterschiedlich sind. DOS gestattet beispielsweise nur acht Zeichen, dann einen Punkt und nochmals drei Zeichen (8.3). Ferner unterscheidet es nicht zwischen Groß- und Kleinschreibung. In der Macintosh-Welt sind Datei-Namen aus mehreren Wörtern gebräuchlich. Will man sicher gehen, so passt man die Datei-Namen von Hand an, ehe man sich auf irgendwelche Automatismen der Übertragungsprogramme verlässt.

In den Home-Verzeichnissen werden einige Dateien oder Verzeichnisse vom System oder von Anwendungen erzeugt und verwaltet. Sie enthalten Einstellungen, Sitzungsparameter und dergleichen. Diese Dateien interessieren den Benutzer selten. Ihr Name beginnt mit einem Punkt (dot), zum Beispiel `.profile`. Sie werden von `ls(1)` nicht angezeigt. Gibt man `ls(1)` mit der Option `-a` ein, so erscheinen auch sie. Solche Dateien (dotfile) werden als **verborgen** (hidden) bezeichnet, sind aber in keiner Weise geheim. Im Lauf der Jahre sammelt sich ein lange Latte davon an, darunter viele, die man nicht mehr braucht. Ich lösche von Zeit zu Zeit meine Dot-Dateien, deren Aufgabe ich nicht kenne und die älter sind als ein Jahr. Bisher ging es gut.

Ein Verzeichnis wird mit dem Kommando `mkdir(1)` erzeugt, mit `rmdir(1)` löscht man ein leeres Verzeichnis, mit `rm -r` (r = rekursiv) ein volles samt Unterverzeichnissen. Frage: Was passiert, wenn Sie gleich nach der Anmeldung `rm -r *` eingeben? Die Antwort nicht experimentell ermitteln!

Ein Arbeiten mit Laufwerken wie unter DOS ist unter UNIX nicht vorgesehen. Man hat es stets nur mit einer einzigen Datei-Hierarchie zu tun. Weitere Datei-Hierarchien – zum Beispiel auf Disketten oder Platten, lokal oder im Netz – können vorübergehend in die Datei-Hierarchie des Systems eingehängt werden. Dabei wird das Wurzel-Verzeichnis des einzuhängenden Datei-Systems auf ein Verzeichnis, einen Mounting Point des root-Datei-Systems abgebildet. Dieses Verzeichnis muss bereits vorhanden sein, darf nicht in Gebrauch und soll leer sein. Falls es nicht leer ist, sind die dort enthaltenen Dateien und Unterverzeichnisse so lange nicht verfügbar, wie ein Datei-System eingehängt ist. Man nennt das **mounten**, montieren, bereitstellen oder einhängen, Kommando `mount(1M)`. Mountet man das Datei-System eines entfernbaren Datenträgers (Diskette, CD) und entfernt diesen, ohne ihn vorher mittels `umount(1M)` unzumounten (zu unmounten?), gibt es Ärger. Moderne UNIXe setzen zum Mounten und Unmounten entfernbarer Datei-Systeme Automount-Dämonen ein, so dass der Benutzer sich keine Gedanken zu machen braucht. Beim Mounten treten Probleme mit den Zugriffsrechten auf. Deshalb gestatten die Verwalter dem gewöhnlichen Benutzer diese Möglichkeit nur auf besonderen Wunsch. Datei-Systeme können auch über das Netz gemountet werden, siehe Network File System (NFS). Wir mounten beispielsweise sowohl lokale Datei-Systeme von weiteren Platten und CD-Laufwerken wie auch Verzeichnisse von Servern aus unserem Rechenzentrum in die root-Verzeichnisse unserer Workstations. Das Weitermounten über das Netz gemounteter Verzeichnisse ist üblicherweise nicht gestattet. Auch werden Superuser-Rechte meist nicht über das Netz weitergereicht. Man sollte sich bewusst sein, dass die Daten von über das Netz gemounteten Datei-Systemen unverschlüsselt durch die Kabel gehen und mitgelesen werden können.

Auf einen entfernbaren Datenträger (Floppy, CD, ZIP, Band etc.) kann auf zwei Arten zugegriffen werden. In beiden Fällen muss das Laufwerk über eine Eintragung im `/dev`-Verzeichnis erreichbar sein. Meist sind dort alle nur denkbaren Peripheriegeräte angelegt – ein Zuviel schadet nicht – und man hat nur herauszufinden, wie das Laufwerk in dem Verzeichnis heißt, beispielsweise `/dev/hdd` oder `/dev/floppy`. Mitunter kostet das etwas experimentellen Aufwand. Ihr Systemschamane sollte es wissen. Muss man das Laufwerk selbst eintragen, braucht man einige Informationen (Treiber, Nummern) und das Kommando `mknod(1M)`. Fall 1: Ist auf dem Datenträger mittels `newfs(1M)` oder `mkfs(1M)` ein Datei-System eingerichtet, muss dieses in das beim Systemstart geöffnete `root`-Datei-System mit dem Kommando `mount(1M)` irgendwo in ein vorhandenes Verzeichnis gemountet werden und wird damit vorübergehend ein Zweig von diesem. Es kann sein, dass ein `automount`-Dämon diese Arbeit stillschweigend erledigt. Auf CDs ist praktisch immer ein Datei-System eingerichtet. Fall 2: Ist der Datenträger dagegen nur formatiert (Kommando `mediainit(1)` oder `fdformat(1)`), d. h. zum Lesen und Schreiben eingerichtet, ohne dass ein Datei-System angelegt wurde, so greift man mit den Kommandos `cpio(1)` oder `tar(1)` darauf zu. Kommandos wie `cd(1)` oder `ls(1)` sind dann sinnlos, es gibt auch keine Inodes. Der Datenträger ist über den Namen des Gerätefiles anzusprechen, beispielsweise `/dev/hdd`, `/dev/rfd.0` oder `/dev/rmt/0m`, mehr weiß das System nicht von ihm. Das Kommando zum Kopieren eines ganzen Unterverzeichnisses auf eine ZIP-Diskette in einem Linux-Rechner könnte so lauten:

```
tar -cf /dev/hdd ./mydir
```

Ein Datenträger mit einem Datei-System kann nur auf einem anderen System gelesen werden, wenn dieses den Datei-System-Typ kennt und mounten kann. Ein Datenträger mit einem Archiv kann nur von einem System gelesen werden, das über das entsprechende Archivierungswerkzeug (`tar(1)`) verfügt. In einem Datei-System kann ich navigieren, ein Archiv nur schreiben und lesen.

Wer sowohl unter DOS/Microsoft Windows wie unter UNIX arbeitet, mache sich den Unterschied zwischen einem Wechsel des Laufwerks (A, B, C ...) unter DOS/Microsoft Windows und dem Mounten eines Datei-Systems unter UNIX sorgfältig klar. Dazwischen liegen Welten.

Über diese beiden Möglichkeiten hinaus finden sich auf einigen Systemen Werkzeuge, um auf DOS-Datei-Systeme auf Disketten oder Festplatten zuzugreifen. Unter Linux sind sie als `mtools(1)` bekannt, unter HP-UX als `doscp(1)` und Verwandtschaft.

2.4.4 Netz-Datei-Systeme (NFS)

Datei-Systeme lassen sich über ein Netz auch von anderen UNIX-Hosts in das eigene Datei-System mounten. Das ist praktisch; ein verbreiteter Mechanismus dazu – das **Network File System** (NFS) – wurde von der Firma SUN entwickelt. Die heutigen UNIXe bringen die Dämonen meistens mit, man braucht sie nur noch zu aktivieren. Mit dem Internet hat das Ganze nichts zu tun. Mit Verschlüsselung auch nichts.

Die beteiligten Server und Klienten sollten gleiche `/etc/passwd`- und `/etc/group`-Dateien verwenden, zumindest was die Besitzer von über das Netz gemounteten Verzeichnissen und Dateien anbelangt. Andernfalls handelt man sich Probleme ein. Das lässt sich durch schlichtes Kopieren bewerkstelligen. Man kann aber auch einen Schritt weiter gehen und ein NIS-Cluster einrichten. Das ist in kleinen Netzen auch nicht schwierig, eigentlich eine folgerichtige Ergänzung zum NFS und ebenfalls von SUN entwickelt.

Auf dem **Datei-Server** müssen mindestens drei Dämonen laufen:

- `portmap` oder `rpcbind`, eine Voraussetzung für diesen und andere Dienste,
- `mountd` oder `rpc.mountd` zur Herstellung der `mount`-Verbindung,
- `nfsd` oder `rpc.nfsd` für die eigentliche Datenübertragung.

Ferner muss in der Datei `/etc/exports` festgelegt werden, welche Verzeichnisse von welchen Maschinen gemountet werden dürfen. Die Syntax von `exports` ist in der UNIX-Welt nicht ganz einheitlich, man schaut am besten auf der man-Seite seiner Maschine nach. Ein Beispiel:

```
/mnt2/homes   *.mvm.uni-karlsruhe.de(rw)
/mnt4/student rzia.rz.uni-karlsruhe.de(r)
```

Hier kann das Verzeichnis `/mnt2/homes` von allen Computern unserer Subdomäne les- und schreibbar gemountet werden, das Verzeichnis `/mnt4/student` nur lesbar von einer Maschine im Rechenzentrum.

Auf den **Klienten** müssen Mounting-Punkte eingerichtet sein und folgende Dämonen laufen:

- `portmap` oder `rpcbind`, eine Voraussetzung für diesen und andere Dienste,
- `biod` oder `nfsiod` zum schnelleren blockweisen Lesen und Schreiben.

In der Datei `/etc/fstab` oder `/etc/vfstab` stehen die Anweisungen für das beim Systemstart aufgerufene `mount`-Kommando:

```
mvm24:/mnt2/homes /mnt2/homes nfs rw,rsize=8192,wsize=8192 0 0
mvm24:/mnt4/student /mnt4/stud nfs defaults 0 0
```

hier also die Anweisung, von der Maschine `mvm24` die genannten Verzeichnisse in nicht notwendig gleichnamige lokale Mounting-Punkte einzuhängen und dabei gewisse Optionen zu beachten, siehe die man-Seite zu `fstab`. Die lokal gemounteten Verzeichnisse sind oben nicht aufgeführt, sie stehen vor den Netz-Verzeichnissen. Das ist zunächst einmal alles. Wenn das Netz gut funktioniert, bemerkt der Benutzer keinen Unterschied zwischen lokalen und fernen Dateien.

2.4.5 Zugriffsrechte

Auf einem Mehrbenutzersystem ist es untragbar, dass jeder Benutzer mit allen Dateien alles machen darf. Jede Datei einschließlich der Verzeichnisse wird daher in UNIX durch einen Satz von neun **Zugriffsrechten**[17] (permission, droit d'accès) geschützt.

Die Benutzer werden eingeteilt in den **Besitzer** (owner, propriétaire), seine **Gruppe** (group, groupe) und die **Menge der sonstigen Benutzer** (others, *ohne* den Besitzer und seine Gruppe), auch Rest der Welt genannt[18]. Diese Interpretation rührt daher, dass die Shell die Zugriffsrechte von links nach rechts liest – wie sie angezeigt werden – und nach der ersten Übereinstimmung zwischen dem aktuellen Benutzer und einer der drei Möglichkeiten aufhört. Die Rechte werden ferner nach **Lesen** (read), **Schreiben** (write) und **Suchen/Ausführen** (search/execute) unterschieden. Bei einer gewöhnlichen Datei bedeutet execute Ausführen (was nur bei Programmen Sinn ergibt), bei einem Verzeichnis Durchsuchen oder Durchqueren. Wer sich ein Verzeichnis mittels `ls(1)` ansehen will, braucht das Leserecht daran. Es ist gelegentlich sinnvoll, wenn auch keine starke Sicherheitsmaßnahme, das Durchsuchen eines Verzeichnisses zu gestatten, das Lesen jedoch zu verweigern. Jedes Zugriffsrecht kann nur vom Besitzer erteilt und wieder entzogen werden. Und wie immer vom Verwalter.

Der Besitzer einer Datei ist zunächst derjenige, der sie erzeugt hat. Mit dem Kommando `chown(1)` lässt sich jedoch der Besitz an einen anderen Benutzer übertragen (ohne dass dieser zuzustimmen braucht). Entsprechend ändert `chgrp(1)` die zugehörige Gruppe. Will man eine Datei für andere lesbar machen, so reicht es nicht, der Datei die entsprechende Leseerlaubnis zuzuordnen oder den Besitzer zu wechseln. Vielmehr müssen alle übergeordneten Verzeichnisse von / (root) an lückenlos das Suchen gestatten. Das wird oft vergessen.

Tab. 2.1: Zugriffsrechte von Dateien

Rechte	Besitzer	Gruppe	Rest der Welt
000	Rechte ändern	nichts	nichts
700	alles	nichts	nichts
750	alles	lesen + ausführen	nichts
755	alles	lesen + ausführen	lesen + ausführen
600	lesen + schreiben	nichts	nichts
640	lesen + schreiben	lesen	nichts
644	lesen + schreiben	lesen	lesen

[17] Manche Systeme unterscheiden zwischen Recht (privilege) und Berechtigung (permission). Ein Benutzer hat das Recht, eine Sitzung zu eröffnen, und die Berechtigung, eine Datei zu lesen.

[18] Unter Microsoft Windows bedeutet *Jeder* wirklich jeder und nicht eine Restmenge wie others.

Die Zugriffsrechte lassen sich in Form einer dreistelligen Oktalzahl angeben, und zwar hat

- die read-Erlaubnis den Wert 4,
- die write-Erlaubnis den Wert 2,
- die execute/search-Erlaubnis den Wert 1

Alle drei Rechte ergeben zusammen den Wert 7, höher geht es nicht. Die drei Stellen der Oktalzahl sind in folgender Weise den Benutzern zugeordnet:

- links der Besitzer (owner),
- in der Mitte seine Gruppe (group), ohne Besitzer
- rechts der Rest der Welt (others), ohne Besitzer und Gruppe

Diese Oktalzahl wird auch als Rechtevektor bezeichnet. Eine sinnvolle Kombination ist, dem Besitzer alles zu gestatten, seiner Gruppe das Lesen und Suchen/Ausführen und dem Rest der Welt nichts. Die Oktalzahl 750 bezeichnet diese Empfehlung. Oft wird auch von den Gruppenrechten kein Gebrauch gemacht, man setzt sie gleich den Rechten für den Rest der Welt, also die Oktalzahl auf 700. Das Kommando zum Setzen der Zugriffsrechte lautet:

```
chmod 750 filename
```

Setzt man die Zugriffsrechte auf 007, so dürfen der Besitzer und seine Gruppe nichts machen. Alle übrigen (Welt minus Besitzer minus Gruppe) dürfen die Datei lesen, ändern und ausführen. Der Besitzer darf nur noch die Rechte auf einen vernünftigeren Wert setzen. Mit der Option `-R` werden die Kommandos `chmod(1)`, `chown(1)` und `chgrp(1)` rekursiv und beeinflussen ein Verzeichnis samt allem, was darunter liegt. Bei `chmod(1)` ist jedoch aufzupassen: meist sind die Zugriffsrechte für Verzeichnisse anders zu setzen als für gewöhnliche Dateien. Es gibt noch weitere Formen des `chmod(1)`-Kommandos sowie die Bezeichnung der Zugriffsrechte durch Buchstaben anstelle von Oktalzahlen.

Tab. 2.2: Zugriffsrechte von Verzeichnissen

Rechte	Besitzer	Gruppe	Rest der Welt
000	Rechte ändern	nichts	nichts
700	alles	nichts	nichts
750	alles	auflisten + durchsuchen	nichts
751	alles	auflisten + durchsuchen	durchsuchen
755	alles	auflisten + durchsuchen	auflisten + durchsuchen

Aus Sicherheitsgründen soll man die Zugriffsrechte möglichst einschränken. Will ein Benutzer auf eine Datei zugreifen und darf das nicht, wird er sich schon rühren. Mittels des Kommandos:

```
ls -l filename
```

erfährt man die Zugriffsrechte einer Datei. Die Ausgabe sieht so aus:

```
-rw-r----- 1 wualex1 users 59209 May 15 unix.tex
```

Die Zugriffsrechte heißen hier also *read* und *write* für den Besitzer `wualex1`, *read* für seine Gruppe `users` und für den Rest der Welt nichts. Die Zahl 1 ist der Wert des Link-Zählers, siehe Abschnitt 2.4.8 *Inodes und Links* auf Seite 74. Dann folgen Besitzer und Gruppe sowie die Größe der Datei in Bytes. Das Datum gibt die Zeit des letzten schreibenden (den Inhalt verändernden) Zugriffs an. Schließlich der Name. Ist das Argument des Kommandos `ls(1)` der Name eines Verzeichnisses, werden die Dateien des Verzeichnisses in ASCII-alphabetischer Folge aufgelistet. Ohne Argument wird das aktuelle Verzeichnis aufgelistet. Das Kommando `ls(1)` kennt viele Optionen.

Beim **Kopieren** muss man Zugang zum Original (Sucherlaubnis für alle übergeordneten Verzeichnisse) haben und dieses lesen dürfen. Besitzer der Kopie wird der Veranlasser des Kopiervorgangs. Er kann anschließend die Zugriffsrechte der Kopie ändern, die Kopie gehört ihm. Leserecht und Kopierrecht sind untrennbar. Das Kommando zum Kopieren lautet:

```
cp originalfile copyfile
```

Falls die Datei `copyfile` schon vorhanden ist, wird es ohne Warnung überschrieben. Ist das Ziel ein Verzeichnis, wird die Kopie dort eingehängt, Schreiberlaubnis in dem Verzeichnis vorausgesetzt. Der Versuch, eine Datei auf sich selbst zu kopieren – was bei der Verwendung von Jokerzeichen oder Links vorkommt – führt zu einer Fehlermeldung.

Zum **Löschen** einer Datei oder eines Verzeichnisses braucht man ebenso wie zum Erzeugen in dem übergeordneten Verzeichnis die Schreiberlaubnis. Zugriffsrechte an der zu löschenden Datei sind nicht erforderlich. Weiteres zum Löschen einer Datei oder eines Verzeichnisses siehe Seite 82.

Die Default-Rechte werden mittels des Kommandos `umask(1)` in der Datei `/etc/profile` oder `$HOME/.profile` gesetzt. Das Kommando braucht als Argument die Ergänzung der Rechte auf 7. Beispielsweise setzt

```
umask 077
```

die Default-Rechte auf 700, ein gängiger Wert. Ohne Argument aufgerufen zeigt das Kommando den aktuellen Wert an.

Will man beim Kopieren Besitzer, Zugriffsrechte und Zeitstempel beibehalten, so ist die Option `-p` (p = preserve) hinzuzufügen. Die Option `-r` führt zu einem rekursiven Kopieren eines Verzeichnisses samt Unterverzeichnissen. Da Kopieren ein häufiger Vorgang ist, sollte man sich den `man`-Eintrag genau durchlesen.

Gelegentlich möchte man einzelnen Benutzern Rechte erteilen, nicht gleich einer ganzen Gruppe. Das geht mit obigen Zugriffsrechten nicht. Unter einigen UNIXen lässt sich jedoch einer Datei eine **Access Control List** (ACL) zuordnen, in die Sonderrechte eingetragen werden, Näheres mittels `man 5 acl`. Der POSIX-Standard sieht entsprechende Systemaufrufe vor.

Für das System ist ein Benutzer im Grunde ein Bündel von Rechten. Ob dahinter eine natürliche oder juristische Person, eine Gruppe von Personen oder ein Dämon steht, ist dem System schnurz und piepe. Es gibt Betriebssysteme wie Microsoft Windows NT, verteilte Datei-Systeme wie im Distributed Computing Environment (DCE) oder Datenbanken, die stärker differenzieren – sowohl nach der Art der Rechte wie nach der Einteilung der Benutzer - aber mit dem Satz von drei mal drei Rechten kommt man schon weit. Die stärkere Differenzierung ist schwieriger zu überschauen und birgt die Gefahr, Sicherheitslücken zu übersehen. In Netzen sind die Zugangswege und damit die Überlegungen zur Sicherheit komplexer. Der **Superuser** oder **Privileged User** mit der User-ID 0 – der Verwalter, System-Manager oder Administrator üblicherweise – ist an die Zugriffsrechte nicht gebunden. Sein Name (fast immer `root`) tut nichts zur Sache, entscheidend ist nur die User-ID. Er kann:

- Jede Datei lesen, verändern oder löschen,
- die Zugriffsrechte jeder Datei und jedes Verzeichnisses ändern,
- Benutzer-Accounts einrichten, sperren oder löschen,
- die Identität jedes Benutzers annehmen,
- Passwörter ändern oder löschen,
- privilegierte Programme einrichten oder ausführen,
- die System-Uhr vor- oder zurückstellen,
- das System jederzeit herunterfahren,
- das System rettungslos in den Sand setzen.

Er kann *nicht*:

- Passwörter entziffern, es sei denn, sie wären leicht zu erraten,
- sorgfältig verschlüsselte Daten entschlüsseln,
- gelöschte Dateien wiederherstellen, es sei denn, es gäbe ein Kopie,
- sämtliche `man`-Seiten im Kopf haben.

Wollen Sie Ihre höchst private Mail vor seinen Augen schützen, müssen Sie sie verschlüsseln, siehe Abschnitt 2.7.11 *Verschlüsseln* auf Seite 156.

Merke: Damit jemand auf eine Datei zugreifen kann, müssen zwei Bedingungen erfüllt sein:

- Er muss einen durchgehenden Suchpfad (x-Erlaubnis) vom Root-Verzeichnis (/) bis zu der Datei und
- die entsprechenden Rechte an der Datei selbst haben.

2.4.6 Set-User-ID-Bit

Vor den drei Oktalziffern der Zugriffsrechte steht eine weitere Oktalziffer, die man ebenfalls mit dem Kommando `chmod(1)` setzt. Der Wert 1 ist das **Sticky Bit** (klebrige Bit), das bei Programmen, die gleichzeitig von mehreren Benutzern benutzt werden (sharable programs), dazu führt, dass die Programme ständig im Arbeitsspeicher (inklusive Swap) verbleiben und dadurch sofort verfügbar sind. Wir haben das Sticky Bit eine Zeitlang beim Editor `vi(1)` verwendet. Auch bei Installationen von `sendmail` kommt es vor. Bei Dateien (Programmen) spielt es heute keine Rolle mehr, das heißt es wird vom Betriebssystem ignoriert. Bei einem Verzeichnis führt das Sticky Bit dazu, dass nur der Besitzer einer darin eingeordneten Datei, der Besitzer des Verzeichnisses oder `root` die Datei löschen oder umbenennen kann[19], auch wenn die übrigen Zugriffsrechte des Verzeichnisses diese Operationen für andere erlauben, Beispiel `/tmp`:

```
drwxrwxrwt   2 bin       bin   2048 Oct 27 17:37 /tmp
-rw-rw-r--   1 wualex1   mvm      0 Oct 27 17:43 /tmp/alex
```

Hier dürfen im Verzeichnis `/tmp` alle Benutzer schreiben und lesen, aber nur ihre eigenen Dateien umbenennen oder löschen. Die Datei `alex` kann nur vom Besitzer `wualex1`, von `bin` oder von `root` umbenannt oder gelöscht werden. Das Sticky Bit wird für öffentliche Verzeichnisse vom Superuser gesetzt, das Kommando `ls -l` zeigt es durch ein `t` oder `T` in der äußersten rechten Stelle der Zugriffsrechte an.

Der Wert 2 ist das **Set-Group-ID-Bit** (SGID-Bit), der Wert 4 das **Set-User-ID-Bit** (SUID-Bit), auch Magic Bit genannt. Sind diese gesetzt, so hat das Programm die Zugriffsrechte des Besitzers (owner), die von den Zugriffsrechten dessen, der das Programm aufruft, abweichen können. Ein häufiger Fall ist, dass ein Programm ausführbar für alle ist, `root` gehört und bei gesetztem SUID-Bit auf Dateien zugreifen darf, die `root` vorbehalten sind. Wohlgemerkt, nur das Programm bekommt die erweiterten Rechte, nicht der Aufrufende. Man sagt, der aus dem Programm hervorgegangene Prozess laufe effektiv mit der Benutzer-ID des Programmbesitzers, nicht wie üblich mit der des Aufrufenden. Bei einem Verzeichnis bewirkt das SGID-Bit, dass eine neu angelegte Datei die Gruppenzugehörigkeit vom übergeordneten Verzeichnis erbt. Das ist der Normalfall auf BSD-ähnlichen Systemen, während auf System-V-ähnlichen Systemen eine neue Datei die Gruppe des anlegenden Prozesses übernimmt. Durch das SGID-Bit bei einem Verzeichnis wird also auf System-V-Rechnern ein Verhalten gemäß BSD erreicht.

Ein gesetztes Set-User-ID-Bit wird durch ein `S` an der Stelle des `x` beim Besitzer (owner) angezeigt, wenn das execute-Recht nicht gesetzt ist, durch ein `s`, wenn auch das execute-Recht gesetzt ist (s = S + x). Entsprechendes gilt für das Set-Group-ID-Bit bei den Zugriffsrechten der Gruppe. Ausführliche Beispiele finden sich im Anhang auf Seite 325.

Das UNIX-Kommando `/bin/passwd(1)` gehört der `root`, ist für alle ausführbar, sein suid-Bit ist gesetzt:

[19] Der einzige Fall, in dem das Löschen ein besonderes Recht darstellt.

```
-r-sr-xr-x   1 root  bin  112640 Nov 22  /bin/passwd
```

Damit ist es möglich, dass jeder Benutzer sein Passwort in der Datei `/etc/passwd(4)` ändern darf, ohne die Schreiberlaubnis für diese Datei zu besitzen:

```
---r--r--r   1 root  other  3107  Dec 2  /etc/passwd
```

Da durch das Programm der Umfang der Änderungen begrenzt wird (nämlich auf die Änderung des eigenen Passwortes), erhält der Benutzer nicht die vollen Rechte des Superusers. Für das sgid-Bit gilt Entsprechendes. Beide können nur für ausführbare (kompilierte) Programme vergeben werden, nicht für Shellskripte, aus Sicherheitsgründen. Das Setzen dieser beiden Bits für Verzeichnisse führt auf unseren Anlagen zu Problemen, die Verzeichnisse sind nicht mehr verfügbar. Wer aufgepasst hat, könnte auf folgende Gedanken kommen:

- Ich kopiere mir den Editor `vi(1)`. Besitzer der Kopie werde ich.
- Dann setze ich mittels `chmod 4555 vi` das suid-Bit. Das ist erlaubt.
- Anschließend schenke ich mittels `chown root vi` meinen `vi` dem Superuser, warum nicht. Das ist ebenfalls erlaubt.

Nun habe ich einen von allen ausführbaren Editor, der Superuser-Rechte hat, also beispielsweise die Datei `/etc/passwd(4)` unbeschränkt verändern darf. Der Gedankengang ist zu naheliegend, als dass nicht die Väter von UNIX auch schon darauf gekommen wären. Probieren Sie es aus.

Falls Sie das Skriptum *Programmieren in C/C++* verinnerlicht haben, könnten Sie weiterdenken und sich ein eigenes Kommando `mychown` schreiben wollen. Dazu brauchen Sie den Systemaufruf `chown(2)`; die Inode-Liste, die den Namen des Datei-Besitzers enthält, ist nicht direkt mittels eines Editors beschreibbar. Leider steht im Referenz-Handbuch, dass der Systemaufruf bei gewöhnlichen Dateien das suid-Bit löscht. Sie geben nicht auf und wollen sich einen eigenen Systemaufruf `chmod(2)` schreiben: Das bedeutet, sich einen eigenen UNIX-Kern zu schreiben. Im Prinzip möglich, aber dann ist unser Buch unter Ihrem Niveau. Dieses Leck ist also dicht, aber Programme mit suid-Bit – zumal wenn sie der `root` gehören – sind immer ein bisschen verdächtig. Ein gewissenhafter Verwalter beauftragt daher den Dämon `cron(1M)` mit ihrer regelmäßigen Überwachung. Da das suid-Bit selten vergeben wird, könnte der Verwalter auch ein eingeschränktes `chmod`-Kommando schreiben und die Ausführungsrechte des ursprünglichen Kommandos eingrenzen.

2.4.7 Zeitstempel

Zu jeder UNIX-Datei gehören drei Zeitangaben, die Zeitstempel (E: timestamp) genannt und automatisch verwaltet werden:

- die Zeit des jüngsten lesenden oder schreibenden Zugriffs (access time, atime),

- die Zeit des jüngsten schreibenden Zugriffs (modification time, mtime),
- die Zeit der jüngsten Änderung des Datei-Status (status change time, ctime).

Ein Lesezugriff ändert nur den ersten Stempel (atime). Ein schreibender Zugriff aktualisiert die beiden ersten Stempel (atime und mtime), weil dem Schreibvorgang ein Lesevorgang vorausgeht. Der Datei-Status umfasst den Besitzer samt Gruppe, die Zugriffsrechte und den Linkzähler, also Informationen aus der Inode. Ein Schreibvorgang bewirkt ebenfalls eine Statusänderung. Schreiben, eine Änderung der Zugriffsrechte mittels `chmod(1)` oder das Hinzufügen von harten Links mittels `ln` ändert daher den dritten Stempel. Kurz und bündig ist das auf der man-Seite zum Systemaufruf `stat(2)` nachzulesen.

Bei Verzeichnissen gilt das Durchsuchen oder Hindurchgehen nicht als lesender Zugriff, Löschen oder Hinzufügen von Dateien bedeutet Schreiben, Auflisten mittels `ls(1)` Lesen.

Mittels des Kommandos `ls(1)` kann man sich die Zeitstempel ansehen (und man sollte auch einmal mit einer temporären Datei alle Änderungen durchspielen):

- `ls -l` zeigt die `mtime` an, also die Zeit des jüngsten schreibenden Zugriffs auf die Datei, wichtig,
- `ls -lu` zeigt die `atime` an, also die Zeit des jüngsten lesenden oder schreibenden Zugriffs auf die Datei,
- `ls -lc` zeigt die `ctime` an, also die Zeit der jüngsten Statusänderung (Inhalt, Besitzer, Zugriffsrechte, Links).

Unter Linux findet sich ein Kommando `stat(1)`, das den Systemaufruf `stat(2)` enthält und die Informationen aus der Inode – darunter die Zeitstempel – lesbar wiedergibt, ähnlich wie das folgende C-Programm, das zu einem Datei-Namen alle drei Zeitstempel anzeigt; falls `DEBUG` definiert ist, auch in Rohform als Sekunden seit UNIX Geburt:

```
/* Information ueber die Zeitstempel einer Datei */

/* #define DEBUG */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>

int main(int argc, char *argv[])

{
struct stat buffer;
struct tm *p;

if (argc < 2) {
    puts("Dateiname fehlt"); return (-1);
}
```

```
if (!access(argv[1], 0)) {
if (!(stat(argv[1], &buffer))) {

#ifdef DEBUG
        puts(argv[1]);
        printf("atime = %ld\n", buffer.st_atime);
        printf("mtime = %ld\n", buffer.st_mtime);
        printf("ctime = %ld\n\n", buffer.st_ctime);
#endif

    p = localtime(&(buffer.st_atime));
    printf("Gelesen:          %d. %d. %d  %2d:%02d:%02d\n",
      p->tm_mday, p->tm_mon + 1, p->tm_year, p->tm_hour,
                                    p->tm_min, p->tm_sec);

    p = localtime(&(buffer.st_mtime));
    printf("Geschrieben:      %d. %d. %d  %2d:%02d:%02d\n",
      p->tm_mday, p->tm_mon + 1, p->tm_year, p->tm_hour,
                                    p->tm_min, p->tm_sec);

    p = localtime(&(buffer.st_ctime));
    printf("Status geaendert: %d. %d. %d  %2d:%02d:%02d\n",
      p->tm_mday, p->tm_mon + 1, p->tm_year, p->tm_hour,
                                    p->tm_min, p->tm_sec);

}
else {
    puts("Kein Zugriff auf Inode (stat)"); return (-1);
}
}
else {
    puts("File existiert nicht (access)"); return (-1);
}
return 0;
}
```

Quelle 2.2 : C-Programm zur Anzeige der Zeitstempel einer Datei

Der Zeitpunkt der Erschaffung einer Datei – das wäre der allererste schreibende Zugriff – wird *nicht* festgehalten und ist auch aus technischer Sicht uninteressant[20]. Nur wenn eine Datei seit seiner Erzeugung nicht verändert worden ist, deckt sich die `mtime` mit dem Entstehungsdatum. Probieren Sie es aus. Änderungen am Datei-Inhalt hingegen sind für Werkzeuge wie `make(1)` und für Datenbanken wichtig, um entscheiden zu können, ob eine Datei aktuell ist.

[20]Die Frage nach den drei Zeitstempeln wird in Prüfungen immer wieder falsch beantwortet. Ich wiederhole: Der Zeitpunkt der Erschaffung einer Datei wird *nicht* in einem Zeitstempel gespeichert. Auch an keiner anderen Stelle. Sprechen Sie langsam und deutlich nach: *Eine Datei hat keinen Geburtstag.*

2.4.8 Inodes und Links

Die Verzeichnisse enthalten nur die Zuordnung Datei-Name zu einer Datei-Nummer, die als **Inode-Nummer** (Index-Node) bezeichnet wird. In der **Inode-Liste**, die vom System verwaltet wird, stehen zu jeder Inode-Nummer alle weiteren Informationen über eine Datei einschließlich der Startadresse und der Größe des Datenbereiches. Insbesondere sind dort auch die Zugriffsrechte und die Zeitstempel vermerkt. Einzelheiten sind im Handbuch unter `inode(5)` und `fs(5)` zu finden. Das Kommando `ls -i` zeigt die Inode-Nummern an. Der Verwalter kann sich mit `ncheck(1M)` eine Liste aller Inode-Nummern samt zugehöriger absoluter Datei-Namen eines Datei-Systems ausgeben lassen. Harte Links erscheinen mehrfach entsprechend der Anzahl zugehöriger Namen. Wie die Informationen der Inode in eigenen Programmen abgefragt werden, steht in Abschnitt 2.8.9.3 *Beispiel Datei-Informationen* auf Seite 213.

Diese Zweiteilung in Verzeichnisse und Inode-Liste erlaubt eine nützliche Konstruktion, die in DOS oder IBM-OS/2 bei aller sonstigen Ähnlichkeit nicht möglich ist. Man kann einer Datei sprich einer Inode-Nummer nämlich mehrere Datei-Namen, unter Umständen in verschiedenen Verzeichnissen, zuordnen. Das nennt man **linken**[21]. Die Datei, auf die mehrere Datei-Namen gelinkt sind, existiert nur einmal (deshalb macht es keinen Sinn, von einem Original zu reden), aber es gibt mehrere Zugangswege, siehe Abbildung 2.5 auf Seite 75. Zwangsläufig gehören zu gelinkten Datei-Namen dieselben Zeitstempel und Zugriffsrechte, da den Namen nur eine einzige Inode zu Grunde liegt. Das Kommando zum Linken zweier Datei-Namen lautet `ln(1)`:

```
ln oldname newname
```

Auf diese Weise spart man Speicher und braucht beim Aktualisieren nur eine einzige Datei zu berücksichtigen. Die Kopie einer Datei mittels `cp(1)` hingegen ist eine eigene Datei mit eigener Inode-Nr., deren weiterer Lebenslauf unabhängig vom Original ist. Beim Linken einer Datei wird ihr **Linkzähler** um eins erhöht. Beim Löschen eines Links wird der Zähler herabgesetzt; ist er auf Null angekommen (und hat kein Programm mehr die Datei geöffnet), wird der von der Datei belegte Speicherplatz freigegeben, die Datei ist futsch.

Bei einem Verzeichnis hat der Linkzähler immer den Wert 2, da jedes Verzeichnis einen Link auf sich selbst enthält, dargestellt durch den Punkt beim Auflisten. So ist die wichtigste Information über ein Verzeichnis – seine Inode-Nummer – doppelt gespeichert, nämlich im übergeordneten Verzeichnis und im Verzeichnis selbst. Ebenso ist in jedem Verzeichnis an zweiter Stelle die Inode-Nummer des übergeordneten Verzeichnisses abgelegt. Jedes Verzeichnis weiß selbst, wie es heißt und wohin es gehört. Das ermöglicht Reparaturen des Datei-Systems bei Unfällen. Als Benutzer kann man Verzeichnisse weder kopieren noch linken, sondern nur die in einem Verzeichnis versammelten Dateien. Old Root kann natürlich wieder mehr, siehe `link(1M)` und `link(2)`.

[21]Das Wort *linken* hat eine zweite Bedeutung im Zusammenhang mit dem Kompilieren von Programmen.

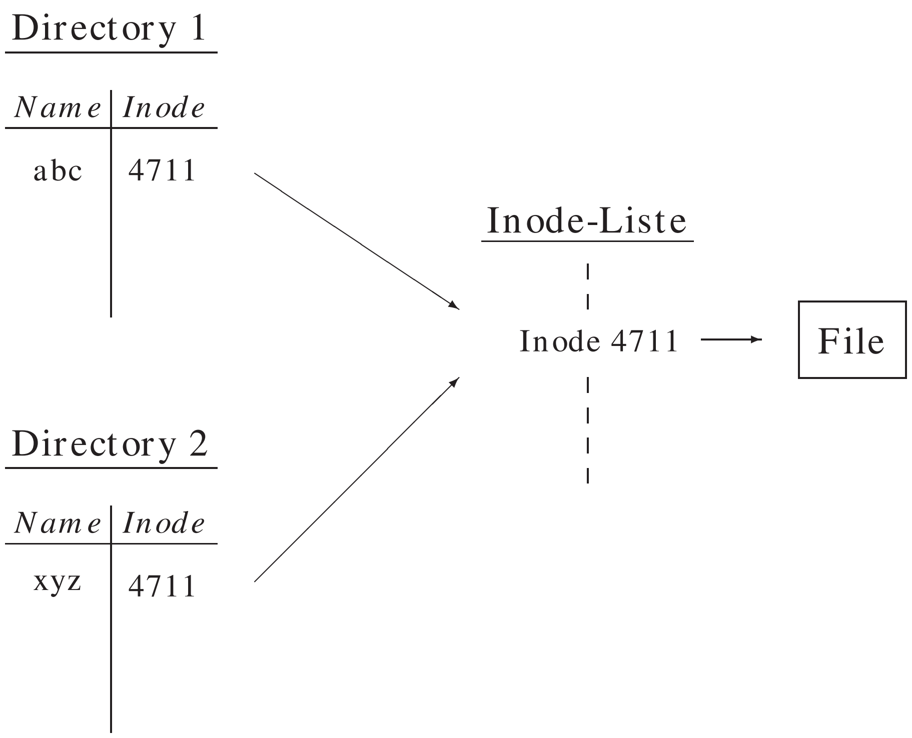

Abb. 2.5: Harter Link

Dieser **harte Link** kann sich nicht über die Grenze eines Datei-Systems erstrecken, auch falls es gemountet sein sollte. Der Grund ist einfach: jedes Datei-System verwaltet seine eigenen Inode-Nummern und hat seinen eigenen Lebenslauf. Es kann heute hier und morgen dort gemountet werden. Ein Link über die Grenze könnte dem Lebenslauf nicht folgen.

Im Gegensatz zu den eben erläuterten harten Links dürfen sich **symbolische Links**[22] oder **weiche Links** über die Grenze eines Datei-Systems erstrecken und sind auch bei Verzeichnissen erlaubt und beliebt. Sie werden mit dem Kommando `ln(1)` mit der Option `-s` erzeugt:

```
ln -s unix scriptum
```

Hier ist `unix` eine bestehende Datei oder ein Verzeichnis, `scriptum` der symbolische oder weiche Link. Dieser ist eine kleine Datei mit eigener Inode-Nummer, das einen Verweis auf einen weiteren absoluten oder relativen Datei- oder Verzeichnisnamen enthält, siehe Abbiildung 2.6 auf Seite 76. Bein Anlegen eines weichen Links prüft das System nicht, ob das Ziel existiert. Das Kommando `ls -l` zeigt weiche Links folgendermaßen an:

```
lrwx------  1 wulf  alex  4 Jun 2  scriptum -> unix
```

Das Verzeichnis `scriptum` ist ein weicher Link auf das Verzeichnis `unix`. Zugriffsrechte eines weichen Links werden vom System nicht beachtet, das

[22]Auch als Symlink, Verknüpfung, Verweis oder Pointer bezeichnet. Haben nichts mit dem C-Datentyp Pointer zu tun.

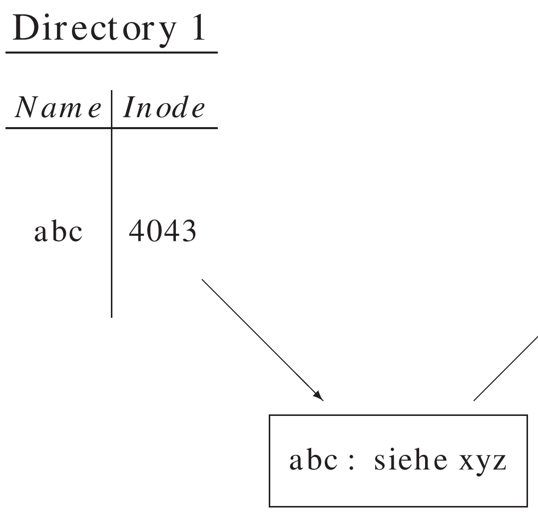

Abb. 2.6: Weicher, Symbolischer oder Soft Link

Kommando `chmod(1)` wirkt auf die zugrunde liegende Datei, `rm(1)` glücklicherweise nur auf den Link. Weiteres siehe `ln(1)` unter `cp(1)`, `symlink(2)` und `symlink(4)`.

Weiche Links dürfen geschachtelt werden. Im Falle des harten Links ist es ohnehin gleich, von welchem Namen der Inode man ausgeht, es gibt ja kein Original, sondern nur eine einzige Datei. Bei weichen Links wird auch eine Kette von Verweisen richtig verarbeitet. Insbesondere erkennt das Kopierkommando `cp(1)` die Links und verweigert ein Kopieren einer Datei auf ihren Link. Mit den Systemaufrufen `lstat(2)` und `readlink(2)` wird auf einen weichen Link direkt zugegriffen, während die Systemaufrufe `stat(2)` und `read(2)` auf die dem Link zugrunde liegende Datei zielen. Wird einem weichen Link seine Ziel-Datei weggenommen, besteht er weiter. Man kann auch einen weichen Link auf eine nicht vorhandene Datei anlegen, das Kommando `ln(1)` prüft das nicht. Zugriffe über den Link auf die Ziel-Datei sind in diesem Fall erfolglos. Hat man die Wahl zwischen einem harten und einem weichen Link, so dürfte der harte geringfügig schneller im Zugriff sein. Die Bestseller unter den Webseiten eines Webservers sollten direkt, nicht über einen weichen Link zu erreichen sein.

Weder die Datei noch ihre Inode weiß etwas von den weichen Links; der Linkzähler in der Inode kennt nur die Anzahl der harten Links. Daher sind die Links zu einer Datei etwas umständlich herauszubekommen. Geht es um harte Links, deren Vorhandensein am Linkzähler erkennbar ist, ermittelt man zuerst die Inode-Nummer und lässt dann `find(1)` suchen:

```
ls -i
find / -inum Inode-Nummer -print
```

Falls man die Suche auf Unterverzeichnisse des `root`-Verzeichnisses begrenzen kann, spart man Zeit und schont die Platten. Weiche Links lassen sich nicht mit Sicherheit vollzählig erfassen. Man muss sich alle Datei-Namen auf der Maschine anzeigen lassen und auf den Namen der in Frage stehenden Datei filtern:

```
ls -Rl / | fgrep filename
```

Auch hier sollte man den Suchraum möglichst eingrenzen. In beiden Fällen kann der Normalbenutzer durch Zugriffsrechte an Verzeichnissen ausgesperrt werden.

Eine ähnliche Aufgabe wie die Links erfüllt die `alias`-Funktion der Shell. Ein `alias` lebt und stirbt jedoch mit der Shell, während ein Link im Datei-System verankert ist und für alle Benutzer gilt.

Merke: Nach einem Kopiervorgang hat man zwei voneinander unabhängige Dateien, das Original und die Kopie. Nach einem Linkvorgang hat man zwei Namen für dieselbe Datei.

2.4.9 stdin, stdout, stderr

Drei Dateien sind für jede Sitzung automatisch geöffnet: `stdin` (in der Regel die Tastatur), `stdout` (in der Regel der Bildschirm) und `stderr` (in der Regel ebenfalls der Bildschirm). Wir erinnern uns, Geräte werden von UNIX formal als Dateien angesprochen. Andere Systeme kennen noch `stdaux` (Standard Auxiliary Device) und `stdprn` (Standard Printer Device).

Zu den Datei-Pointern `stdin`, `stdout` und `stderr` gehören die Datei-Deskriptoren 0, 1 und 2. **Datei-Pointer** sind Namen (genauer Namen von Pointern auf eine C-Struktur vom Typ FILE), **Datei-Deskriptoren** fortlaufende Nummern der für ein Programm geöffneten Dateien. In Programmen wird durch einen open-Aufruf einem Datei-Namen ein Datei-Pointer oder ein Datei-Deskriptor zugeordnet, mit dem dann die weiteren Anweisungen arbeiten. Die UNIX-Systemaufrufe (2) verwenden Datei-Deskriptoren, die C-Standardfunktionen (3) Datei-Pointer. Beispiele finden sich im C-Programm 2.40 *Datei-Informationen* auf Seite 217.

Werkzeuge soll man möglichst so schreiben, dass sie von `stdin` lesen, ihre Ausgabe nach `stdout` und ihre Fehlermeldungen nach `stderr` schreiben. Dann sind sie allgemein verwendbar und passen zu den übrigen Werkzeugen. Solche Programme werden als **Filter** bezeichnet.

Eine leere Datei wird mit der Umlenkung `> filename`, mit `cat(1)` oder `touch(1)` angelegt. Zum Leeren einer Datei kopiert man `/dev/null` dorthin.

Das Kommando `tee(1)` liest von `stdin`, schreibt nach `stdout` und gleichzeitig eine Kopie der Ausgabe in eine Datei, wie ein T-Stück sozusagen:

```
who | tee whofile
```

Über das Verbinden von `stdout` eines Prozesses mit `stdin` eines zweiten Prozesses mittels einer Pipe wurde bereits in Abschnitt 2.3.6.2 *Pipes* auf Seite 47 gesprochen.

Als **Kontroll-Terminal** `/dev/tty` eines Prozesses werden der Bildschirm und die Tastatur bezeichnet, von denen der Prozess gestartet worden ist. Über sein Kontroll-Terminal wickelt der Prozess seine Ein- und Ausgabe ab, falls nichts anderes vereinbart wird. Das ist fast dasselbe wie die Datei-Pointer `stdin` usw., aber nur fast. Während `/dev/tty` ein Gerät oder ein Special Device File ist, sind die drei Datei-Pointer zunächst einmal nur logische Quellen und Ziele. Manche Programme machen da einen Unterschied. Die Ein- und Ausgabe ist auf allen Systemen ein Gebiet voller Fallstricke.

2.4.10 Schreiben und Lesen von Dateien

Dateien werden mit einem Editor, z. B. dem `vi(1)`, geschrieben (siehe Abschnitt 2.7.3 *Editoren* auf Seite 144), von Compilern oder anderen Programmen erzeugt oder laufen einem über das Netz zu. Zum Lesen von Dateien auf dem Bildschirm stehen die Kommandos `cat(1)`, `more(1)`, `less`, `pg(1)`, `view(1)` und `vis(1)` zur Verfügung. `cat(1)` liest von `stdin` und schreibt nach `stdout`. Lenkt man die Eingabe mit `cat < filename` um, bekommt man die Datei `filename` auf den Bildschirm. Die Pager `more(1)`, `less` und `pg(1)` arbeiten ähnlich, halten aber nach jeweils einer Bildschirmseite an. `view(1)` ist der Editor `vi(1)` im Lesemodus (read-only), `vis(1)` wandelt etwaige nicht sichtbare Zeichen in ASCII-Nummern um. Der Versuch, Dateien zu lesen, die etwas anderes als in Zeilen gegliederten Text enthalten, führt in manchen Fällen zu einem Blockieren des Terminals. Begnügt man sich mit dem Lesen des Anfangs oder Endes einer Datei, leisten die Kommandos `head(1)` und `tail(1)` gute Dienste.

Will man sich den Inhalt einer beliebigen Datei genau ansehen, so schreibt man mit `od(1)`, gegebenenfalls mit der Option −c, einen Dump nach `stdout`, bei Schwierigkeiten nützlich. Ein **Dump** (F: cliché) ist eine bytegetreue Wiedergabe des Speicher- oder Datei-Inhalts ohne jede Bearbeitung. Dasselbe und mehr leisten Binär-Editoren, siehe Abschnitt 2.7.8 *Binär-Editoren* auf Seite 152.

2.4.11 Archivierer (tar, gtar)

Dateien werden oft mit drei Werkzeugen behandelt, die nichts miteinander zu tun haben, aber häufig kombiniert werden. Diese sind:

- Archivierer wie `tar(1)`,
- Packer (Komprimierer) wie `compress(1)` oder `gzip(1)`,
- Verschlüsseler wie `crypt(1)`.

Um Archivierer geht es in diesem Abschnitt, um Packer im folgenden. Verschlüsselt werden in erster Linie Textfiles, daher kommen wir im Abschnitt 2.7 *Writer's Workbench* auf Seite 134 zu diesem Thema. Mit der Verschlüsselung hängen weitere Fragen zusammen, die in Netzen eine Rolle spielen.

Zum Aufbewahren oder Verschicken von ganzen Datei-Gruppen ist es oft zweckmäßig, sie in eine einzige Datei zu verpacken. Diesem Zweck dient das Kommando `tar(1)`. Der Aufruf

```
tar -cvf name.tar name*
```

stopft alle Dateien des Arbeits-Verzeichnisses, auf die das Namensmuster (Jokerzeichen!) zutrifft, in eine Datei `name.tar`, das als Archiv bezeichnet wird. Die Option[23] `c` bedeutet *create*, mit der Option `v` wird `tar(1)` geschwätzig (verbose), und `f` weist den Archivierer an, das nächste Argument als Ziel der Packerei aufzufassen. Das zweite Argument darf auch ein Verzeichnis sein. Eine Kompression oder Verschlüsselung ist damit nicht verbunden. Bei der Wahl der Argumente ist etwas Nachdenken angebracht. Das frisch erzeugte Archiv darf nicht zum Kreis der zu archivierenden Dateien gehören, sonst beißt sich `tar(1)` unter Umständen in den Schwanz. Ferner hält sich `tar(1)` genau an die Namensvorgaben, absolute oder relative Namen werden auch als solche verpackt:

```
tar -cvf unix.tar *.tex               # (in /buch/unix)
tar -cvf unix.tar ./*.tex             # (in /buch/unix)
tar -cvf unix.tar unix/*.tex          # (in /buch)
tar -cvf unix.tar /buch/unix/*.tex    # (irgendwo)
```

archivieren zwar dieselben Dateien, aber unter verschiedenen Pfadnamen, was beim Auspacken zu verschiedenen Ergebnissen führt. Die erste und zweite Form lassen sich in einem beliebigen Verzeichnis auspacken. Die dritte Form kann an beliebiger Stelle entpackt werden und erzeugt dort ein Unterverzeichnis namens `unix`. Die vierte Form ist unflexibel und führt zu demselben absoluten Pfad wie beim Packen.

Zum Auspacken dient das Kommando (`x` = extract, `v` = verbose, `f` = file):

```
tar -xvf name.tar
```

Ist man sich unsicher, in welcher Form das Archiv angelegt worden ist, schaut man sich erst einmal das Inhaltsverzeichnis an (`t` = table of contents):

```
tar -tf /dev/st0
```

Hier ist der Datei-Name der Name einer Geräte-Datei, nämlich das SCSI-Tape Nr. 0. In diesem DAT-Laufwerk liegt das Band, dessen Verzeichnis ich mir anschauen möchte. Schon an den ersten Zeilen erkennt man, wie das Archiv angelegt wurde. Dann wechselt man vorsichtshalber in ein `tmp`-Verzeichnis und entpackt dort, wobei hier die dritte der obigen Archivierungsmöglichkeiten vorausgesetzt wird:

```
tar -xf /dev/st0 www/tmg/*
```

Im `tmp`-Verzeichnis finde ich anschließend ein Unterverzeichnis `www` mit allem, was darunter liegt. Zweckmäßig kopiert man stets das auszupackende

---

[23]Eigentlich handelt es sich bei diesen Optionen um Kommandos, Funktionen oder Schlüssel für `tar(1)`, die keinen vorangehenden Bindestrich erfordern, aber da das jeden Benutzer verwirrt, hat man den Bindestrich zugelasssen.

Archiv in ein eigenes Verzeichnis, weil hinterher unter Umständen ein umfangreicher Verzeichnisbaum an Stelle des Archivs grünt. Manchmal legt das Archiv beim Auspacken dieses Verzeichnis selbst an, am besten in einem temporären Verzeichnis ausprobieren.

Will man eine einzelne Datei aus einem `tar`-Archiv extrahieren und ist sich nicht sicher, mit welchem Pfad es archiviert wurde, liest man zuerst den Inhalt des Archivs (`t` = table of contents):

```
tar -tvf /dev/st0 | grep abc | tee tarinhalt
```

Hier wird angenommen, dass das Archiv von einem Bandgerät (SCSI-Tape Nr. 0) kommt, weiter, dass im Namen der gesuchten Datei der Substring `abc` vorkommt. Die Ausgabe schreiben wir gleichzeitig nach `stdout` (Bildschirm) und in eine Datei `tarinhalt`, zur Sicherheit. Mit dem so ermittelten Pfad gehen wir in das Kommando zum Extrahieren:

```
tar -xf /dev/st0 pfad
```

Ein `tar`-Archivfile kann mit einem beliebigen Packer verdichtet werden (erst archivieren, dann packen). Das ist im Netz üblich, um den Übertragungsaufwand zu verringern. Will man sich den Inhalt eines mit `gzip(1)` gepackten `tar(1)`-Archivs ansehen, hilft folgende Kombination (Pipe):

```
gunzip < file.tar.gz | tar -tf -
```

Im einzelnen: `gunzip` liest aus dem gepackten Archiv `file.tar.gz` und schickt seine Ausgabe per Pipe an `tar`, das von der Datei `stdin` (Bindestrich) liest und das Inhaltsverzeichnis ausgibt. Hängen wir noch `> file.inhalt` an das `tar`-Kommando an, wandert das Inhaltsverzeichnis in eine Datei und kann in Ruhe weiterverarbeitet werden.

Das GNU-Kommando `gtar(1)` archiviert und komprimiert bei entsprechender Option in einem Arbeitsgang:

```
gtar -cvzf myarchive.tar.gz filenames
```

Die vielen Möglichkeiten von `tar(1)` verwirren, sind aber logisch und beherrschbar.

2.4.12 Packer (compress, gzip)

Die meisten Dateien enthalten überflüssige Zeichen. Denken Sie an mehrere aufeinanderfolgende Leerzeichen, für die die Angabe des Zeichens und deren Anzahl aureichen würde. Um Speicherplatz und Übertragungszeit zu sparen, verdichtet man solche Dateien. Das Standard-Kommando dafür ist `compress(1)`, ein jüngeres und wirkungsvolleres Kommando `gzip(1)`. Die urpsrüngliche Datei wird gelöscht, die verdichtete Datei bekommt die Kennung `.Z` oder `.gz`. Zum Verdünnen auf die ursprüngliche Konzentration ruft man `uncompress(1)` oder `gunzip(1)` mit dem Datei-Namen auf. Das Packen ist vollkommen umkehrbar[24]. Probieren Sie folgende Kommandofolge aus (`textfile` sei ein mittelgroßes Textfile):

---

[24]Im Zusammenhang mit dem Speichern von Bildern oder Klängen gibt es auch verlustbehaftete Kompressionsverfahren.

```
cp textfile textfile1
cp textfile textfile2
ll textfile*
compress textfile1
gzip textfile2
ll textfile*
uncompress textfile1.Z
gunzip textfile2.gz
ll textfile*
cmp textfile textfile1
cmp textfile textfile2
```

`gzip(1)` kennt eine Option, mit der man zwischen minimaler Rechenzeit oder maximaler Verdichtung wählen kann. Es lohnt sich jedoch kaum, vom Defaultwert abzuweichen. Auch binäre Dateien lassen sich verdichten. Ein mehrfaches Verdichten ist nicht zu empfehlen. Grafik- oder Sound-Formate, die bereits von sich aus verdichten, lassen sich nicht nennenswert weiter verdichten. Dasselbe gilt für das Adobe-pdf-Dokumentenformat. In der DOS-Welt gibt es eine Vielzahl anderer Packprogramme, teils frei, teils gegen Bares. Einige davon archivieren und packen zugleich.

Der ultimative Packer ist `rm(1)`. Er arbeitet vollkommen unumkehrbar und hat die höchstmögliche Verdichtung. Sie sollten vor dem Gebrauch unbedingt die zugehörige man-Seite lesen.

2.4.13 Weitere Kommandos

Das Kommando `ls(1)` listet ein Verzeichnis auf und ist das UNIX-Kommando mit den meisten Optionen[25]. Die Form `ll(1)` ist gleichwertig `ls -l`, oft als Shell-Alias verwirklicht.

Mit `mv(1)` benennt man eine Datei um und verschiebt sie gegebenenfalls in ein anderes Verzeichnis, ihre Inode-Nummer bleibt:

```
mv alex blex
mv alex ../../alex
ls | xargs -i -t mv {} subdir/{}
```

In der dritten Form listet `ls(1)` das Arbeitsverzeichnis auf. Die Ausgabe wird durch eine Pipe dem Kommando `xargs(1)` übergeben, das wegen der Option `-i` (insert) die übernommenen Argumente in die beiden Klammernpaare einsetzt – und zwar einzeln – und dann das Kommando `mv(1)` aufruft, erforderlichenfalls mehrmals. Die Option `-t` (trace) bewirkt die Anzeige jeder Aktion auf `stderr`. Auf diese Weise lassen sich alle Dateien eines Verzeichnisses oder eine Auswahl davon verschieben. Ebenso lässt sich ein Verzeichnis umbennen, ohne es zu verschieben. Das folgende Shellskript ersetzt in allen Namen des aktuellen Verzeichnisses Großbuchstaben durch die entsprechenden Kleinbuchstaben:

[25]Im X Window System gibt es Kommandos mit Hunderten von Optionen. Das wird dann anders gehandhabt (X-Resources).

```
for i in `ls`
do
mv $i `echo $i | tr '[A-Z]' '[a-z]'`
done
```

Das heißt: für jede einzelne Datei aus der von `ls(1)` erzeugten Liste ersetze den Datei-Namen (`$i` durch den gleichen Namen, jedoch unter Umwandlung von Groß- in Kleinbuchstaben. Beachte: `ls` und die Pipe sind von Backquotes eingerahmt, die eckigen Klammern vorsichtshalber durch Apostrophe gequotet. Das braucht man gelegentlich, wenn man Dateien aus der DOS-Welt erbt.

Das Kommando `mvdir(1M)` verschiebt ein Verzeichnis an eine andere Stelle in selben Datei-System und ist dem Verwalter vorbehalten, da bei unvorsichtigem Gebrauch geschlossene Wege innerhalb des Datei-Baums entstehen. Auf manchen Systemen lässt sich das einfache `mv` auch auf Verzeichnisse anwenden.

Zum **Löschen** (to delete, to erase, effacer) von Dateien bzw. Verzeichnissen dienen `rm(1)` und `rmdir(1)`. Ein leeres Verzeichnis wird mit `rmdir(1)` gelöscht, ein volles samt allen Unterverzeichnissen mit `rm -r`, Vorsicht bei der Verwendung von Jokerzeichen! UNIX fragt nicht, sondern handelt – beinhart und gnadenlos. Gefährlich sind vor allem die Kommandos `rm *` und `rm -r directoryname`, die viele Dateien auf einen Schlag löschen. Das Löschen einer Datei mittels `rm(1)` besteht aus folgenden Schritten:

- Im übergeordneten Verzeichnis wird der Name der Datei samt zugehöriger Inode-Nummer gestrichen. Das erfordert Schreibrecht am Verzeichnis, aber keine Rechte an der Datei selbst. Ausnahme: Verzeichnisse, bei denen das Sticky Bit gesetzt ist, oft bei `tmp` der Fall.
- In der Inode der Datei wird der Linkzähler um eins herabgesetzt.
- Ist der Linkzähler auf null angekommen, wird die Inode gelöscht, die Datei existiert nicht mehr. Der Speicherplatz wird als frei markiert und steht für neue Daten zur Verfügung. Die Datei ist **logisch** gelöscht.
- Die Bits auf dem ehemaligen Speicherplatz bleiben **physikalisch** bestehen, bis der Platz erneut belegt wird. Wann das geschieht, entzieht sich dem Einfluss und der Kenntnis des Benutzers, auch des Superusers. Bei hohen Anforderungen an die Sicherheit muss man daher eine zu löschende Datei erst mit sinnlosen Zeichenmustern überschreiben, die Puffer leeren und dann löschen.

Eine mit `rm(1)` gelöschte Datei kann nicht wiederhergestellt werden, außer mit viel Glück und Aufwand. Der als frei markierte Bereich auf dem Massenspeicher wird im nächsten Augenblick von anderen Benutzern, einem Dämon oder vom System erneut belegt. Wer dazu neigt, die Reihenfolge von Denken und Handeln zu verkehren, sollte sich ein Alias für `rm` einrichten, das vor dem Löschen zurückfragt:

```
alias -x del='rm -i'
```

oder das Löschen durch ein Verschieben in ein besonderes Verzeichnis ersetzen, welches am Ende der Sitzung oder nach einer bestimmten Frist (cron(1) und find(1)) geleert wird:

```
# Shellscript saferm zum verzoegerten Loeschen 05.12.96
# Verzeichnis /saferm 333 root root erforderlich

case $1 in
        -*) option=$1; shift;;
         *) ;;
esac

/bin/cp $* /saferm
/bin/rm $option $*
```

Quelle 2.3 : Shellskript saferm zum verzögerten Löschen von Dateien

Hat man versehentlich wichtige Daten gelöscht, besteht noch ein Hauch von Hoffnung. So schnell wie möglich ist die betroffene Partition vor schreibenden Zugriffen zu schützen (unmounten oder wenigstens readonly mounten). Es gibt Werkzeuge zum Untersuchen von Platten auf niedriger Ebene wie den File System Debugger fsdb(1M) oder den *Coroner's Toolkit* von:

```
http://www.fish.com/tct/
```

Die Reparaturversuche sind mühsam, ein Erfolg ist nicht garantiert.

Zum Leeren einer Datei, ohne sie zu löschen, verwendet man am einfachsten folgende Zeile:

```
> filename
```

Die Datei hat anschließend die Größe 0 Bytes. Eine andere Möglichkeit ist das Kopieren von /dev/null in die Datei.

Nun zu einem Dauerbrenner in der entsprechenden Gruppe der Netnews. Wie werde ich eine Datei mit einem absonderlichen Namen los? In UNIX-Datei-Namen können – wenn es mit rechten Dingen zugeht – alle Zeichen außer dem Schrägstrich und dem ASCII-Zeichen Nr. 0 vorkommen. Der Schrägstrich trennt Verzeichnisnamen voneinander, die ASCII-0 beendet einen Datei-Namen, einen String. Escape-Folgen, die den Bildschirm löschen oder die Tastatur blockieren, sind erlaubte, wenn auch unzweckmäßige Namen. Aber auch die beiden genannten Zeichen fängt man sich gelegentlich über das Netz ein. Erzeugen Sie ein paar absonderlich benamte Dateien, am besten in einem für Experimente vorgesehenen Verzeichnis:

```
touch -abc
touch '   '
touch 'x  y'
touch '/'
```

und schauen Sie sich Ihr Verzeichnis mit:

```
ls -aliq
```

an. Wenn Sie vorsichtig sind, kopieren oder transportieren Sie alle vernünftigen Dateien in ein anderes Verzeichnis, ehe Sie dem Übel zu Leibe rücken. Die Datei -abc, deren Name mit einem Bindestrich wie bei einer Option beginnt, wird man mit einem der folgenden Kommandos los (ausprobieren):

```
rm ./-abc
rm - -abc
rm -- -abc
```

Enthalten die Namen Zeichen, die für die Shell eine besondere Bedeutung haben (Metazeichen), hilft Einrahmen des Namens in Apostrophe (Quoten mit Single Quotes), siehe oben. Zwei weitere, meist gangbare Wege sind:

```
rm -i *
find . -inum 12345 -ok rm '{}' \;
```

Das erste Kommando löscht alle Dateien im Arbeitsverzeichnis, fragt aber zuvor bei jeder einzelnen Datei um Erlaubnis. Das zweite Kommando ermittelt im Arbeitsverzeichnis die Datei mit der Inode-Nummer 12345, fragt um Erlaubnis und führt gegebenenfalls das abschließende Kommando rm(1) aus. Die geschweiften Klammern, der Backslash und das Semikolon werden von find(1) verlangt. Wollen Sie die widerspenstige Datei nur umbennen, sieht das Kommando so aus:

```
find . -inum 12345 -ok mv '{}' anstaendiger_name \;
```

Datei-Namen mit einem Schrägstrich oder ASCII-Null kommt man so jedoch nicht bei. In diesem Fall kopiert man sämtliche gesunden Dateien in ein anderes Verzeichnis, löscht mittels clri(1M) die Inode des schwarzen Schafes, führt einen File System Check durch und holt sich die Daten aus lost+found zurück. Man kann auch – sofern man kann – mit einem File System Debugger den Namen im Verzeichnis editieren. Weiteres siehe in der FAQ-Liste der Newsgruppe comp.unix.questions. Zur Zeit besteht sie aus acht Teilen und wird von TED TIMAR gepflegt. Unbedingt lesenswert, auch wenn man keine Probleme hat.

Der Verwalter kann eine Inode mit dem Kommando clri(1M) löschen, etwaige Verzeichniseinträge dazu bleiben jedoch erhalten und müssen mit rm(1) oder fsck(1M) beseitigt werden. Das Kommando ist eigentlich dazu gedacht, Inodes zu löschen, die in keinem Verzeichnis mehr aufgeführt sind.

Zum Auffinden von Dateien dienen locate, which, whereis und find. locate sucht in einem vorbereiteten Index nach Dateien aller Art, which sucht nach ausführbaren Dateien (Kommandos), whereis nach Kommandos, deren Quelldateien und man-Seiten. type und whence geben ähnliche Informationen:

```
which ls
whereis ls
type ls
whence ls
```

Das Werkzeug find(1) ist vielseitig und hat daher eine umfangreiche, auf unterschiedlichen Systemen nicht völlig einheitliche Syntax:

```
find . -name 'vorwort.*' -print
find . -name '*.tex' | xargs grep -i hallo
find $HOME -size +1000 -print
find / -atime +32 -print
find /tmp -type f -mtime +8 -exec rm -f '{}' \;
find /tmp -type f -mtime +8 -print | xargs rm -f
```

Das Kommando der ersten Zeile sucht im Arbeits-Verzeichnis und seinen Unterverzeichnissen (das heißt rekursiv) nach Dateien mit dem Namen vorwort.* und gibt die Namen auf stdout aus. Der gesuchte Name vorwort.* steht in Hochkommas (Apostrophe, Single Quotes), damit das Jokerzeichen nicht von der Sitzungsshell, sondern von find(1) ausgewertet wird. In der nächsten Zeile schicken wir die Ausgabe durch eine Pipe zu dem Kommando xargs(1). Dieses fügt die Ausgabe von find(1) an die Argumentliste von grep(1) an und führt grep(1) aus. xargs(1) ist also ein Weg unter mehreren, die Argumentliste eines Kommandos aufzubauen. In der dritten Zeile wird im Home-Verzeichnis und seinen Unterverzeichnissen nach Dateien gesucht, die größer als 1000 Blöcke (zu 512 Bytes) sind. Der vierte Aufruf sucht im ganzen Datei-System (von root abwärts) nach Dateien, auf die seit mehr als 32 Tagen nicht mehr zugegriffen wurde (access time, Zeitstempel). Der Normalbenutzer erhält bei diesem Kommando einige Meldungen, dass ihm der Zugriff auf Verzeichnisse verwehrt sei, aber der Verwalter benutzt es gern, um Ladenhüter aufzuspüren. Die Kommandos der vierten und fünften Zeile erzielen die gleiche Wirkung auf verschiedenen Wegen. Beide suchen im Verzeichnis /tmp nach gewöhnlichen Dateien (und nicht nach Verzeichnissen, Pipes usw.), die seit mehr als 8 Tagen nicht modifiziert worden sind. Das Kommando der vierten Zeile führt für jeden gefundenen Datei-Namen einen rm-Prozess aus. Die fünfte Zeile erzeugt einen einzigen rm-Prozess mit einer Argumentliste. Dieser Weg ist effektiver, kann aber bei eigenartigen Datei-Namen wegen der fehlenden Quotung Probleme bereiten.

Ein Kommando wie DOS tree zur Anzeige des Dateibaumes gibt es in UNIX leider nicht. Deshalb hier ein Shellskript für diesen Zweck, das wir irgendwo abgeschrieben haben:

```
dir=${1:-$HOME}
(cd $dir; pwd)
find $dir -type d -print |
sort -f |
sed -e "s,^$dir„" -e "/^$/d" -e \
"s,[^/]*/\([^/]*\)$,\--->\1," -e "s,[^/]*/, |    ,g"
```

Quelle 2.4 : Shellskript tree zur Anzeige der Datei-Hierarchie

Die Zwischenräume und Tüttelchen sind wichtig; fragen sie bitte jetzt nicht nach ihrer Bedeutung. Schreiben Sie das Shellskript in eine Datei namens tree und rufen Sie zum Testen tree /usr auf. Ohne die Angabe eines Verzeichnisses zeigt `tree` das Home-Verzeichnis. Unter MINIX dient das Kommando `traverse(1)` demselben Zweck.

Der Verwalter (nur er, wegen der Zugriffsrechte) verschafft sich mit:

```
/etc/quot -f myfilesystem
```

eine Übersicht darüber, wieviele Kilobytes von wievielen Dateien eines jeden Besitzers im Datei-System `myfilesystem` belegt werden. Das Datei-System kann das `root`-Verzeichnis, ein gemountetes Verzeichnis oder ein Unterverzeichnis sein. Das Kommando geht nicht über die Grenze eines Datei-Systems hinweg.

2.4.14 Begriffe Dateien

Folgende Begriffe sollten klarer geworden sein:

- Datei-System
- Inode
- Link, harter und weicher
- Verzeichnis, Home-Verzeichnis
- Zugriffsrechte

Folgende Kommandos sollten beherrscht werden:

- `cat`, `more`
- `cd`
- `chmod`, `chgrp`, `chown`
- `compress`, `gzip`
- `find`
- `ln`
- `ls` mit einigen Optionen
- `mkdir`, `rmdir`
- `mv`, `cp`, `rm`
- `tar`
- `touch`

2.4.15 Memo Dateien

- Unter UNIX gibt es gewöhnliche Dateien, Verzeichnisse und Geräte-Dateien.
- Alle Dateien sind in einem einzigen Verzeichnis- und Dateibaum untergebracht, an dessen Spitze (Wurzel) das root-Verzeichnis steht.

- Datei-Namen dürfen bis zu 255 Zeichen lang sein und alle ASCII-Zeichen außer dem Schrägstrich und der ASCII-Nr. 0 (nicht druckbares Zeichen) enthalten.
- Jede Datei oder jedes Verzeichnis gehört einem Besitzer und einer Gruppe.
- Die Zugriffsrechte bilden eine Matrix von Besitzer - Gruppe - Rest der Welt und Lesen - Schreiben - Ausführen/Durchsuchen.
- Jede Datei oder jedes Verzeichnis besitzt eine Inode-Nummer. In der Inode stehen die Informationen über die Datei, in den Verzeichnissen die Zuordnung Inode-Nummer - Name.
- Ein harter Link ist ein weiterer Name zu einer Inode-Nummer. Ein weicher Link ist eine Datei mit einem Verweis auf eine andere Datei oder Verzeichnis.
- Ein Datei-System kann in einen Mounting Point (leeres Verzeichnis) eines anderen Datei-Systems eingehängt (gemountet) werden, auch über ein Netz (NFS).
- Entfernbare Datenträger enthalten entweder ein Datei-System oder ein Archiv.
- Ein Archivierprogramm wie `tar(1)` packt mehrere Dateien oder Verzeichnisse ins eine einzige Datei (Archiv).
- Ein Packprogramm wie `gzip(1)` verdichtet eine Datei ohne Informationsverlust (reversibel).

2.4.16 Übung Dateien

Melden Sie sich unter Ihrem Benutzernamen an. Ihr Passwort wissen Sie hoffentlich noch. Geben Sie folgende Kommandos ein:

`id`	(Ihre persönlichen Daten)
`who`	(Wer ist zur Zeit eingeloggt?)
`tty`	(Wie heißt mein Terminal?)
`pwd`	(Wie heißt mein Arbeits-Verzeichnis?)
`ls`	(Arbeits-Verzeichnis auflisten)
`ls -l` oder `ll`	
`ls -li`	
`ls /`	(root-Verzeichnis auflisten)
`ls /bin`	(bin-Verzeichnis)
`ls /usr`	(usr-Verzeichnis)
`ls /dev`	(dev-Verzeichnis, Gerätefiles)
`cat lsfile`	(lsfile lesen)
`mail`	(Falls Ihnen Mail angezeigt wird, kommen Sie mit RETURN weiter.)

`mail root`	(Nun können Sie dem Verwalter einen Brief schreiben. Ende: RETURN, `control-d`)
`mkdir privat`	(Verzeichnis erzeugen)
`cd privat`	(dorthin wechseln)
`cp /mnt/student/beispiel beispiel`	
	(Die Datei `/mnt/student/beispiel` ist eine kurze, allgemein lesbare Textdatei. Fragen Sie Ihren Verwalter.)
`cat beispiel`	(Datei anzeigen)
`head beispiel`	(Datei-Anfang anzeigen)
`more beispiel`	(Datei bildschirmweise anzeigen)
`less beispiel`	(Datei bildschirmweise anzeigen, Linux)
`pg beispiel`	(Datei bildschirmweise anzeigen, HP-UX)
`od -c beispiel`	(Datei als ASCII-Text dumpen)
`file beispiel`	(Datei-Typ ermitteln)
`file /bin/cat`	
`whereis cat`	(Datei suchen)
`ln beispiel exempel`	(Linken)
`cp beispiel uebung`	(Datei kopieren)
`ls -i`	
`mv uebung schnarchsack`	
	(Datei umbenennen)
`ls`	
`more schnarchsack`	
`rm schnarchsack`	(Datei löschen) (Auf die Frage `mode?` antworten Sie `y`)
`vi text1`	(Editor aufrufen)
`a`	
Schreiben Sie einen kurzen Text. Drücken Sie die ESCAPE-Taste.	
`:wq`	(Editor verlassen)
`pg text1`	
`lp text1`	(Fragen Sie Ihren Verwalter nach dem üblichen Druckkommando)
Abmelden mit `exit`	

2.4.17 Fragen Dateien

- Was ist eine Datei?
- Was ist ein Verzeichnis?
- Die wichtigsten drei Datei-Arten unter UNIX?

- Wie sieht die Datei-Hierarchie unter UNIX aus?
- Was bedeutet *mounten*?
- Was ist ein Netz-Datei-System (NFS)?
- Was enthalten die obersten Verzeichnisse der Hierarchie?
- Was bedeutet die Ausgabe des Kommandos `ls -l`?
- Was ist ein Home-Verzeichnis, ein Arbeits-Verzeichnis?
- Was ist ein (absoluter) Pfad?
- Welche Einschränkungen gelten für Datei-Namen unter UNIX?
- Welche Zugriffsrechte gibt es unter UNIX?
- Welche Zeitstempel gibt es?
- Was ist eine Inode?
- Was ist ein harter Link? Ein weicher Link? Warum braucht man beide?
- Was bedeuten die Datei-Pointer `stdin`, `stdout` und `stderr`?
- Wie greift man auf entfernbare Datenträger zu?
- Was ist ein Archiv?
- Was heißt *Packen* einer Datei?
- Wie kann ich nach Dateien suchen?
- Wie werde ich Dateien mit absonderlichen Namen los?

2.5 Shells

2.5.1 Gesprächspartner im Dialog

2.5.1.1 Kommandointerpreter

Wenn man einen **Dialog** mit dem Computer führt, muß im Computer ein Programm laufen, die rohe Hardware antwortet nicht. Der Gesprächspartner ist ein **Kommandointerpreter**, also ein Programm, das unsere Eingaben als Kommandos oder Befehle auffaßt und mit Hilfe des Betriebssystems und der Hardware ausführt. Man findet auch den Namen *Bediener* für ein solches Programm, das zwischen Benutzer und Betriebssystem vermittelt. Dieses erste Dialogprogramm einer Sitzung wird aufgrund der Eintragung in der Datei `/etc/passwd(4)` gestartet; es ist der Elternprozess aller weiteren Prozesse der Sitzung und fast immer eine Shell, die **Sitzungsshell**, bei uns `/bin/ksh(1)` auf HP-Maschinen, `bash(1)` unter Linux.

Ein solcher Kommandointerpreter gehört zwar zu jedem dialogfähigen Betriebssystem, ist aber im strengen Sinn nicht dessen Bestandteil (Abbildung 2.1 auf Seite 35). Er ist ein Programm, das für das Betriebssystem auf gleicher Stufe steht wie vom Anwender aufgerufene Programme. Er ist ersetzbar, es dürfen auch mehrere Kommandointerpreter gleichzeitig verfügbar sein (aber nicht mehrere Betriebssysteme).

Die übliche Form eines **UNIX-** oder **Shell-Kommandos** sieht so aus:

```
command -option1 -option2 argument1 argument2 ....
```

Man findet auch Langformen der Optionen, die aussagekräftiger sind und mit zwei Minuszeichen eingeleitet werden:

```
command -\/-lange_option1 -\/-lange_option2 argument1 ,,,,
```

Manche Optionen dürfen wahlweise in beiden Formen eingegeben werden oder auch gemischt. Auskunft erteilt die jeweilige man-Seite. Kommandos werden auch Systembefehle genannt, im Gegensatz zu den Befehlen oder Anweisungen in einem Programm. Wegen der Verwechslungsgefahr mit Systemaufrufen (system call) ziehen wir das Wort *Kommando* vor. **Optionen** modifizieren die Wirkung des Kommandos. Es gibt Kommandos ohne Optionen wie `pwd(1)` und Kommandos mit unüberschaubar vielen wie `ls(1)`. Mehrere gleichzeitig gewählte Optionen dürfen meist zu einem einzigen Optionswort zusammengefaßt werden. Es gibt auch Kommandos, die keinen Bindestrich oder zwei davon vor einer Option verlangen. Einige Optionen brauchen ein Argument, das unmittelbar oder durch Leerzeichen getrennt anschließt. Unter **Argumenten** werden Dateinamen oder Strings verstanden, soweit das Sinn ergibt. Die Reihenfolge von Optionen und Argumenten ist bei manchen Kommandos beliebig, aber da die UNIX-Kommandos auf Programmierer mit unterschiedlichen Vorstellungen zurückgehen, hilft im Zweifelsfall nur der Blick ins Referenz-Handbuch oder auf die man-Seite.

Kommandoeingabe per Menü ist unüblich, aber machbar, siehe Programm 2.11 *Shellskript für ein Menü* auf Seite 105. Die Verwendung einer Maus setzt erweiterte curses-Funktionen voraus, siehe Abschnitt 2.6.1.3 *Fenster (Windows), curses-Bibliothek* auf Seite 121, oder das X Window System.

Unter PC-DOS heißt der Standard-Kommandointerpreter `command.com`. Auf UNIX-Anlagen sind es die **Shells**. Im Anfang war die **Bourne-Shell** `sh(1)` oder `bsh(1)`, geschrieben von STEPHEN R. BOURNE. Als Programmiersprache ist sie ziemlich mächtig, als Kommando-Interpreter läßt sie Wünsche offen. Dennoch ist sie die einzige Shell, die auf jedem UNIX-System vorhanden ist.

Aus Berkeley kam bald die **C-Shell** `csh(1)`, geschrieben von BILL JOY, die als Kommando-Interpreter mehr leistete, als Programmiersprache infolge ihrer Annäherung an C ein Umgewöhnen erforderte. Sie enthielt auch anfangs mehr Fehler als erträglich. So entwickelte sich der unbefriedigende Zustand, daß viele Benutzer als Interpreter die C-Shell, zum Abarbeiten von Shellskripts aber die Bourne-Shell wählten (was die doppelte Aufgabe der Shell verdeutlicht). Alle neueren Shells lassen sich auf diese beiden zurückführen. Eine Weiterentwicklung der C-Shell (mehr Möglichkeiten, weniger Fehler) ist die **TENEX-C-Shell** `tcsh(1)`. Wer bei der C-Syntax bleiben möchte, sollte sich diese Shell ansehen.

Die **Korn-Shell** `ksh(1)` von DAVID G. KORN verbindet die Schreibweise der Bourne-Shell mit der Funktionalität der C-Shell. Einige weitere Funktionen, die sich inzwischen als zweckmäßig erwiesen hatten, kamen hinzu. Der Umstieg von Bourne nach Korn ist einfach, manche Benutzer merken es nicht einmal. Die Korn-Shell ist proprietär, sie wird nur gegen Bares abgegeben. Die **Windowing-Korn-Shell** `wksh(1)` ist eine grafische Version der Korn-Shell, die vom X Window System Gebrauch macht; in ihren Shellskripts werden auch X-Window-Funktionen aufgerufen.

Das GNU-Projekt stellt die **Bourne-again-Shell** `bash(1)` frei zur Verfügung, die in vielem der Korn-Shell ähnelt. Linux verwendet diese Shell. Die **Z-Shell** `zsh(1)` kann mehr als alle bisherigen Shells zusammen. Wir haben sie auf unserer Anlage eingerichtet, benutzen sie aber nicht, da uns bislang die Korn-Shell oder die `bash(1)` reicht und wir den Aufwand der Umstellung scheuen. Dann gibt es noch eine **rc-Shell**, die klein und schnell sein soll. Hinter uns steht kein Shell-Test-Institut, wir enthalten uns daher einer Bewertung. Im übrigen gibt es zu dieser Frage eine monatliche Mitteilung in der Newsgruppe `comp.unix.shell`. Wem das nicht reicht, dem steht es frei, sich eine eigene Shell zu schreiben. Im folgenden halten wir uns an die Korn-Shell `ksh(1)`.

Einige der Kommandos, die Sie der Shell übergeben, führt sie persönlich aus. Sie werden **interne** oder **eingebaute Kommandos** genannt. Die Kommandos `cd(1)` und `pwd(1)` gehören dazu, unter PC-DOS beispielsweise `dir`. Das `dir` entsprechende UNIX-Kommando `ls(1)` hingegen ist ein externes Kommando, ein eigenes Programm, das wie viele andere Kommandos vom Interpreter aufgerufen wird und irgendwo in der Datei-Hierarchie zu Hause ist (`/bin/ls` oder `/usr/bin/ls`). Welche Kommandos intern und welche extern sind, ist eine Frage der Zweckmäßigkeit. Die internen Kommandos finden Sie unter `sh(1)` beziehungsweise `ksh(1)`, Abschnitt Special Commands. Die Reihe der externen Kommandos können Sie durch eigene Programme beliebig erweitern. Falls Sie für die eigenen Kommandos Namen wie `test(1)` oder `pc(1)` verwenden, die durch UNIX schon belegt sind, gibt es Ärger.

Die Namen von UNIX-Kommandos unterliegen nur den allgemeinen Regeln für Dateinamen, eine besondere Kennung à la `.exe` oder `.bat` ist nicht üblich. Eine Eingabe wie `karlsruhe` veranlaßt die Shell zu folgenden Tätigkeiten:

- Zuerst prüft die Shell, ob das Wort ein Aliasname ist (Zweitname, wird bald erklärt). Falls ja, wird es ersetzt.
- Ist das – unter Umständen ersetzte – Wort ein internes Kommando, wird es ausgeführt.
- Falls nicht, wird ein externes Kommando – eine Datei also – in einem der in der PATH-Variablen (wird auch bald erklärt) genannten Verzeichnisse gesucht. Bleibt die Suche erfolglos, erscheint eine Fehlermeldung: `not found`.
- Dann werden die Zugriffsrechte untersucht. Falls diese das Lesen und Ausführen gestatten, geht es weiter. Andernfalls: `cannot execute`.

- Die Datei sei gefunden und ein Shellskript (wird auch bald erklärt) oder ein ausführbares (kompiliertes) Programm. Dann läßt die Shell es in einem Kindprozess ausführen. Das Verhalten bei Syntaxfehlern (falsche Option, fehlendes Argument) ist Sache des Shellskripts oder Programmes, hängt also davon ab, was sich der Programmierer gedacht hat. Ein guter Programmierer läßt den Benutzer nicht ganz im Dunkeln tappen.
- Die Datei sei gefunden, sei aber eine Textdatei wie ein Brief oder eine Programmquelle. Dann bedauert die Shell, damit nichts anfangen zu können, d. h. sie sieht den Text als ein Shellskript mit furchtbar vielen Fehlern an. Das gleiche gilt für Gerätefiles oder Verzeichnisse.

Die Shell vermutet also hinter dem ersten Wort einer Kommandozeile immer ein Kommando. Den Unterschied zwischen einem Shellskript und einem übersetzten Programm merkt sie schnell. `karlsruhe` war eine leere Datei mit den Zugriffsrechten 777. Was hätten Sie als Shell damit gemacht?

In Dateinamen ermöglicht die Shell den Gebrauch von **Jokerzeichen**, auch Wildcards genannt. Diese Zeichen haben *nichts* mit regulären Ausdrücken zu tun, sie sind eine Besonderheit der Shell. Die Auswertung der Joker heißt **Globbing**. Ein Fragezeichen bedeutet genau ein beliebiges Zeichen. Ein Dateiname wie

```
ab?c
```

trifft auf Dateien wie

```
ab1c  abXc  abcc  ab_c
```

zu. Ein Stern bedeutet eine beliebige Anzahl beliebiger Zeichen. Das Kommando

```
ls abc*z
```

listet alle Dateien des augenblicklichen Arbeits-Verzeichnisses auf, deren Name mit `abc` beginnt und mit `z` endet, beispielsweise:

```
abcz  abc1z  abc123z  abc.z  abc.fiz   abc_xyz
```

Der Stern allein bedeutet *alle Dateien* des Arbeitsverzeichnisses. Eine Zeichenmenge in eckigen Klammern wird durch genau ein Zeichen aus der Menge ersetzt. Der Name

```
ab[xyz]
```

trifft also zu auf

```
abx  aby  abz
```

In der Regel setzt die Shell die Jokerzeichen um, es ist aber auch programmierbar, daß das aufgerufene Kommando diese Arbeit übernimmt. Dann muß man beim Aufruf des Kommandos die Jokerzeichen quoten (unwirksam machen).

Was bewirken die Kommandos `rm a*` und `rm a *` (achten Sie auf den Space im zweiten Kommando)? Also Vorsicht bei `rm` in Verbindung mit dem Stern! Das Kommando – hier `rm(1)` – bekommt von der Shell eine Liste der gültigen Dateinamen, sieht also die Jokerzeichen gar nicht.

Es gibt weitere Zeichen, die für die Shells eine besondere Bedeutung haben. Schauen Sie im Handbuch unter `sh(1)`, Abschnitt *File Name Generation and Quoting* oder unter `ksh(1)`, Abschnitt *Definitions*, **Metazeichen** nach. Will man den Metazeichen ihre besondere Bedeutung nehmen, muß man sie **quoten**[26]. Es gibt drei Stufen des Quotens, Sperrens, Zitierens, Entwertens oder Maskierens. Ein Backslash quotet das nachfolgende Zeichen mit Ausnahme von Newline (line feed). Ein Backslash-Newline-Paar wird einfach gelöscht und kennzeichnet daher die Fortsetzung einer Kommadozeile. Anführungszeichen (double quotes) quoten alle Metazeichen außer Dollar, back quotes, Backslash und Anführungszeichen. Einfache Anführungszeichen (Hochkomma, Apostroph, single quotes) quoten alle Metazeichen außer dem Apostroph oder Hochkomma (sonst käme man nie wieder aus der Quotung heraus). Ein einzelnes Hochkomma wird wie eingangs gesagt durch einen Backslash gequotet.

Probieren Sie folgende Eingaben aus (`echo` oder für die Korn-Shell `print`):

```
echo TERM
echo $TERM
echo \$TERM
echo "$TERM"
echo '$TERM'
```

Wenn man jede Interpration einer Zeichenfolge durch die Shell verhindern will, setzt man sie meist der Einfachheit halber in Single Quotes, auch wenn es vielleicht nicht nötig wäre.

Schließlich gibt es noch die **back quotes** (accent grave). Für die Shell bedeuten sie *Ersetze das Kommando in den back quotes durch sein Ergebnis*. Sie erkennen die Wirkung an den Kommandos:

```
print Mein Verzeichnis ist pwd.
print Mein Verzeichnis ist `pwd`.
```

Im Druck kommt leider der Unterschied zwischen dem Apostroph und dem Accent grave meist nicht deutlich heraus; für die Shell liegen Welten dazwischen. Geben Sie die beiden Kommandos:

```
pwd
`pwd`
```

[26]englisch *quoting* im Sinne von anführen, zitieren wird in den Netnews gebraucht. Ferner gibt es einen `quota(1)`-Mechanismus zur Begrenzung der Belegung des Massenspeichers. Hat nichts mit dem Quoten von Metazeichen zu tun.

ein, so ist die Antwort im ersten Fall erwartungsgemäß der Name des aktuellen Verzeichnisses, im zweiten Fall eine Fehlermeldung, da der Shell der Pfad des aktuellen Verzeichnisses als Kommando vorgesetzt wird. Die Korn-Shell kennt eine zweite Form der Substitution:

```
lp $(ls)
```

die etwas flexibler in der Handhabung ist und sich übersichtlicher schachteln läßt. Obiges Kommando übergibt die Ausgabe von `ls` als Argument an das Kommando `lp`.

Die C-Shell und die Korn-Shell haben einen **History**-Mechanismus, der die zuletzt eingetippten Kommandos in einer Datei `.sh_history` (bei der Korn-Shell, lesbar) speichert. Mit dem internen Kommando `fc` greift man in der Korn-Shell darauf zurück. Die Kommandos lassen sich editieren und erneut ausführen. Tippt man nur `fc` ein, erscheint das jüngste Kommando als Text in dem Editor, der mittels der Umgebungsvariablen FCEDIT festgelegt wurde, meist im `vi(1)`. Man editiert das Kommando und verläßt den Editor auf die übliche Weise, den `vi(1)` also mit `:wq`. Das editierte Kommando wird erneut ausgeführt und in die History-Datei geschrieben. Das Kommando `fc -l -20` zeigt die 20 jüngsten Kommandos an, das Kommando `fc -e -` wiederholt das jüngste Kommando unverändert. Weiteres im Handbuch unter `ksh(1)`, Special Commands.

Der Ablauf einer Sitzung läßt sich festhalten, indem man zu Beginn das Kommando `skript(1)` gibt. Alle Bildschirmausgaben werden gleichzeitig in eine Datei `typeskript` geschrieben, das man später lesen oder drucken kann. Die Wirkung von `skript(1)` wird durch das shellinterne Kommando `exit` beendet. Wir verwenden `skript(1)` bei Literaturrecherchen im Netz, wenn man nicht sicher sein kann, daß alles bis zum glücklichen Ende nach Wunsch verläuft.

Mittels des shellinternen Kommandos `alias` (sprich ejlias) – das aus der C-Shell stammt – lassen sich für bestehende Kommandos neue Namen einführen. Diese haben Gültigkeit für die jeweilige Shell und je nach Option für ihre Abkömmlinge. Der Aliasname wird von der Shell buchstäblich durch die rechte Seite der Zuweisung ersetzt; dem Aliasnamen mitgegebene Optionen oder Argumente werden an den Ersatz angehängt. Man überlege sich den Unterschied zu einem gelinkten Zweitnamen, der im Datei-System verankert ist. Ein weiterer Unterschied besteht darin, daß interne Shell-Kommandos zwar mit einem Aliasnamen versehen, aber nicht gelinkt werden können, da sie nicht in einer eigenen Datei niedergelegt sind. Gibt man in der Sitzungsshell folgende Kommandos:

```
alias -x dir=ls
alias -x who='who | sort'
alias -x r='fc -e -'
```

so steht das Kommando `dir` mit der Bedeutung und Syntax von `ls(1)` zur Verfügung, und zwar zusätzlich. Ein Aufruf des Kommandos `who` führt zum Aufruf der Pipe, das echte `who(1)` ist nur noch über seinen absoluten Pfad

/bin/who erreichbar. Dieses who-Alias hat einen Haken. Ruft der nichtsahnende Benutzer who mit einer Option auf, so wird die Zeichenfolge who durch das Alias ersetzt, die Option mithin an sort angehängt, das meist nichts damit anfangen kann und eine Fehlermeldung ausgibt. Der Aufruf von r wiederholt das jüngste Kommando unverändert, entspricht also der F3-Taste auf PCs unter PC-DOS. Die Option -x veranlaßt den Export des Alias in alle Kindprozesse; sie scheint jedoch nicht überall verfügbar zu sein. Die Quotes sind notwendig, sobald das Kommando Trennzeichen (Space) enthält. Das Kommando alias ohne Argumente zeigt die augenblicklichen Aliases an. Mittels unalias wird ein Alias aufgehoben. Aliases lassen sich nur unter bestimmten Bedingungen schachteln.

Einige Shells bieten Shellfunktionen als Alternative zu Aliasnamen an. In der Bourne- und der Kornshell kann man eine Funktion dir() definieren:

```
dir () { pwd; ls -l $*; }
```

(die Zwischenräume um die geschweiften Klammern sind wichtig) die wie ein Shellkommando aufgerufen wird. Einen Weg zum Exportieren haben wir nicht gefunden. Mittels unset dir wird die Funktion gelöscht.

Die durch die Anmeldung erzeugte erste Shell – die **Sitzungsshell** – ist gegen einige Eingabefehler besonders geschützt. Sie läßt sich nicht durch das Signal Nr. 15 (SIGTERM) beenden, auch nicht durch die Eingabe von EOF (End of File, üblicherweise control-d, festgelegt durch stty(1) in $HOME/.profile), sofern dies durch das Kommando set -o ignoreeof eingestellt ist.

2.5.1.2 Umgebung

Die Shells machen noch mehr. Sie stellen für jede Sitzung eine **Umgebung** (environment, environnement) bereit. Darin sind eine Reihe von Variablen oder Parametern enthalten, die der Benutzer bzw. seine Programme immer wieder brauchen, beispielsweise die Namen des Home-Verzeichnisses und der Mailbox, der Terminaltyp, der Prompt, der Suchpfad für Kommandos, die Zeitzone. Mit dem internen Kommando set holen Sie Ihre Umgebung auf den Bildschirm. Sie können Ihre Umgebung verändern und aus Programmen oder Shellskripts heraus abfragen.

Die **Shellvariablen** gliedern sich in zwei Gruppen:

- benannte Variable oder Parameter, auch als Schlüsselwortparameter bezeichnet,
- Positionsvariable oder -parameter.

Benannte Variable haben einen eindeutigen Namen und erhalten ihren Inhalt oder Wert durch eine Zuweisung:

```
TERM=vt100
```

Die **Positionsvariablen** werden von der Shell beim Aufruf eines Kommandos automatisch mit bestimmten Positionen aus der Kommandozeile gefüllt;

sie ermöglichen den gezielten Zugriff auf Teile der Kommandozeile. Näheres dazu im Abschnitt 2.5.2 *Shellskripts* auf Seite 100. In der Umgebung gibt es nur benannte Variable (programmiertechnisch ist die Umgebung ein Array von Strings).

Benannte Variable gelten zunächst nur in der Shell, in der sie erzeugt wurden. Sie sind lokal. Erst mit Hilfe einer `export`-Anweisung werden sie global und gelten dann für die Shell und ihre Abkömmlinge:

```
MEINNAME="Wulf Alex"; export MEINNAME
export SEINNAME="Bjoern Alex"
```

Hier erzeugen wir die zunächst lokale benannte Variable `MEINNAME` und exportieren sie mittels einer zweiten Anweisung. Manche Shells erlauben die Kombination beider Anweisungen wie in der zweiten Zeile. Die Namen der Variablen werden üblicherweise groß geschrieben. Um das Gleichheitszeichen herum dürfen keine Zwischenräume (spaces) stehen. Die Werte in obigem Beispiel müssen von Anführungszeichen eingerahmt werden, da der Zwischenraum (space) für die Shell ein Trennzeichen ist. Ein leeres Paar von Anführungszeichen stellt den leeren String dar; die Variable ist definiert, hat aber keinen verwertbaren Inhalt. Bleibt die rechte Seite der Zuweisung völlig leer, wird die Variable gelöscht. Das nackte `export`-Kommando zeigt die momentanen globalen benannten Variablen an.

Einige benannte Parameter werden von der Sitzungsshell beim Start erzeugt und auf alle Kindprozesse vererbt. Sie gelten global für die ganze Sitzung bis zu ihrem Ende. Für diese Parameter besteht eine implizite oder explizite `export`-Anweisung; sie werden als **Umgebungs-Variable** bezeichnet. Eine Umgebung, wie sie `set` auf den Bildschirm bringt, sieht etwa so aus:

```
CDPATH=:..:/mnt/alex
EDITOR=/usr/bin/vi
EXINIT=set exrc
FCEDIT=/usr/bin/vi
HOME=/mnt/alex
IFS=

LOGNAME=wualex1
MAIL=/usr/mail/wualex1
MAILCHECK=600
OLDPWD=/mnt/alex
PATH=/bin:/usr/bin:/usr/local/bin::
PPID=1
PS1=A
PS2=>
PS3=#?
PWD=/mnt/alex/unix
RANDOM=2474
SECONDS=11756
SHELL=/bin/ksh
```

```
TERM=ansi
TMOUT=0
TN=console
TTY=/dev/console
TZ=MSZ-2
_=unix.tex
```

Das bedeutet im einzelnen:

- CDPATH legt einen Suchpfad für das Kommando `cd(1)` fest. Die Namen von Verzeichnissen, die sich im Arbeits-Verzeichnis, im übergeordneten oder im Home-Verzeichnis `/mnt/alex` befinden, können mit ihrem Grundnamen (relativ) angegeben werden.
- EDITOR nennt den Editor, der standardmäßig zur Änderung von Kommandozeilen aufgerufen wird.
- EXINIT veranlaßt den Editor `vi(1)`, beim Aufruf das zugehörige Konfigurations-Kommando auszuführen.
- FCEDIT gibt den Editor an, mit dem Kommandos bearbeitet werden, die über den History-Mechanismus zurückgeholt worden sind (Kommando `fc`).
- HOME nennt das Home-Verzeichnis.
- IFS ist das interne Feld-Trennzeichen, das die Bestandteile von Kommandos trennt, in der Regel space, tab und newline.
- LOGNAME (auch USER) ist der beim Einloggen benutzte Name.
- MAIL ist die Mailbox.
- MAILCHECK gibt in Sekunden an, wie häufig die Shell die Mailbox auf Zugänge abfragt.
- OLDPWD nennt das vorherige Arbeits-Verzeichnis.
- PATH ist die wichtigste Umgebungsvariable. Sie gibt den Suchpfad für Kommandos an. Die Reihenfolge spielt eine Rolle. Der zweite Doppelpunkt am Ende bezeichnet das jeweilige Arbeits-Verzeichnis.
- PPID ist die Parent Process-ID der Shell, hier also der `init`-Prozess.
- PS1 ist der erste Prompt, in der Regel das Dollarzeichen, hier individuell abgewandelt. PS2 und PS3 entsprechend.
- PWD nennt das augenblickliche Arbeits-Verzeichnis.
- RANDOM ist eine Zufallszahl zur beliebigen Verwendung.
- SECONDS ist die Anzahl der Sekunden seit dem Aufruf der Shell.
- SHELL nennt die Shell.
- TERM nennt den Terminaltyp, wie er in der `terminfo(4)` steht. Wird vom `vi(1)` und den `curses(3)`-Funktionen benötigt.

- TMOUT gibt die Anzahl der Sekunden an, nach der die Shell sich beendet, falls kein Zeichen eingegeben wird. Der hier gesetzte Wert 0 bedeutet kein Timeout. Üblich: 1000.
- TN ist das letzte Glied aus TTY, eine lokale Erfindung.
- TTY ist die Terminalbezeichnung aus dem Verzeichnis `/dev`, wie sie das Kommando `tty(1)` liefert.
- TZ ist die Zeitzone, hier mitteleuropäische Sommerzeit, zwei Stunden östlich Greenwich.
- _ (underscore) enthält das letzte Argument des letzten asynchronen Kommandos.

Unter PC-DOS gibt es eine ähnliche Einrichtung, die ebenfalls mit dem Kommando `set` auf dem Bildschirm erscheint.

Auch zum Ändern einer Variablen geben Sie ein Kommando der beschriebenen Art ein (keine Spaces um das Gleichheitszeichen):

```
LOGNAME=root
```

Danach hat die bereits vorher vorhandene Variable LOGNAME den Wert `root`, die Rechte der `root` haben Sie aber noch lange nicht.

In der Korn-Shell kann man dem Prompt etwas Arbeit zumuten (back quotes):

```
PS1='${pwd##*/}>  '
```

Er zeigt dann den Grundnamen des augenblicklichen Arbeitsverzeichnisses an, was viele Benutzer vom PC her gewohnt sind.

Die Umgebungsvariablen werden in einer Reihe von Shellskripts gesetzt. Bei mehrfachem Setzen hat das zuletzt aufgerufene Skript Vorrang. In einfachen UNIX-Systemen werden zunächst in einem für alle Benutzer gültigen Skript `/etc/profile` die wichtigsten Umgebungsvariablen festgelegt. Dann folgen individuelle Werte in `$HOME/.profile`, die aber größtenteils für alle Benutzer gleich sind, so daß es sinnvoll ist, allen Benutzern ein `.profile` zum Kopieren anzubieten oder besser gleich bei der Einrichtung des Home-Verzeichnisses dorthin zu kopieren (und dem Benutzer die Schreibrechte zu verweigern).

Auf Systemen mit X11 und möglicherweise einer darauf aufgesetzten Arbeitsumgebung wie das Common Desktop Environment (CDE) oder Hewlett-Packards Visual User Environment (VUE) wird die Geschichte unübersichtlich. X11 und VUE laufen bereits vor der Anmeldung eines Benutzers, so daß die oben genannten `profile`-Skripts nicht von `/usr/bin/login` aufgerufen werden. Stattdessen kommen Variable aus X11, CDE und VUE zum Zuge. Auf unseren Anlagen ist das Skript `.vueprofile` allerdings so konfiguriert, daß es `$HOME/.profile` aufruft. Dann findet sich noch ein `$HOME/.dtprofile`, das beim Öffnen eines Terminalfensters abgearbeitet wird. Der System-Manager sollte einmal mittels `find(1)` nach allen Skripts forschen, deren Namen die Zeichenfolge `profil` enthält.

Ein C-Programm zur Anzeige der Umgebung ähnlich dem Kommando `set` sieht so aus:

```
/* umgebung.c, Programm zur Anzeige der Umgebung */

#include <stdio.h>

int main(argc, argv, envp)
int argc;
char *argv[], *envp[];

{
int i;

for (i = 0; envp[i] != NULL; i++)
    printf("%s\n", envp[i]);

return 0;
}
```

Quelle 2.5 : C-Programm zur Anzeige der Umgebung

Die Umgebung ist ein Array of Strings namens `envp`, dessen Inhalt genau das ist, was `set` auf den Bildschirm bringt. In der `for`-Schleife werden die Elemente des Arrays sprich Zeilen ausgegeben, bis das Element `NULL` erreicht ist. Statt die Zeilen auszugeben, kann man sie auch anders verwerten.

2.5.1.3 Umlenkung

Beim Aufruf eines Kommandos oder Programmes lassen sich Ein- und Ausgabe durch die Umlenkungszeichen $<$ und $>$ in Verbindung mit einem Dateinamen in eine andere Richtung umlenken. Beispielsweise liest das Kommando `cat(1)` von `stdin` und schreibt nach `stdout`. Lenkt man Ein- und Ausgabe um:

```
cat < input > output
```

so liest `cat(1)` das hhile `input` und schreibt es in die Datei `output`. Das Einlesen von `stdin` oder der Datei `input` wird beendet durch das Zeichen EOF (End Of File) oder `control-d`. Etwaige Fehlermeldungen erscheinen nach wie vor auf dem Bildschirm, `stderr` ist nicht umgeleitet. Doppelte Pfeile zur Umlenkung der Ausgabe veranlassen das Anhängen der Ausgabe an einen etwa bestehenden Inhalt der Datei, während der einfache Pfeil die Datei von Beginn an beschreibt:

```
cat < input >> output
```

Existiert die Datei noch nicht, wird es in beiden Fällen erzeugt.

Die Pfeile lassen sich auch zur Verbindung von Datei-Deskriptoren verwenden. Beispielsweise verbindet

```
command 2>&1
```

den Datei-Deskriptor 2 (in der Regel `stderr`) des Kommandos `command` mit dem Datei-Deskriptor 1 (in der Regel `stdout`). Die Fehlermeldungen von `command` landen in derselben Datei wie die eigentliche Ausgabe. Lenkt man noch `stdout` um, so spielt die Reihenfolge der Umlenkungen eine Rolle. Die Eingabe

```
command 1>output 2>&1
```

lenkt zunächst `stdout` (Datei-Deskriptor 1) in die Datei `output`. Anschließend wird `stderr` (Datei-Deskriptor 2) in die Datei umgelenkt, die mit dem Datei-Deskriptor 1 verbunden ist, also nach `output`. Vertauscht man die Reihenfolge der beiden Umlenkungen, so wird zunächst `stderr` nach `stdout` (Bildschirm) umgelenkt (was wenig Sinn macht, weil `stderr` ohnehin der Bildschirm ist) und anschließend `stdout` in die Datei `output`. In der Datei `output` findet sich nur die eigentliche Ausgabe. Sind Quelle und Ziel einer Umlenkung identisch:

```
command >filename <filename
```

so hat das unabhängig von der Reihenfolge in der Kommandozeile die unerwünschte Wirkung, dass die Datei geleert wird.

Die Umlenkungen werden von der Shell geleistet. Das Kommando erhält von der Shell die bereits umgelenkten Datei-Deskriptoren. Das hat den Vorteil, daß man sich beim Schreiben eigener Kommandos nicht um den Umlenkungsmechanismus zu kümmern braucht.

2.5.2 Shellskripts

Wenn man eine Folge von Kommandos häufiger braucht, schreibt man sie in eine Datei und übergibt dem Kommandointerpreter den Namen dieser Datei. Unter PC-DOS heißt eine solche Datei Stapeldatei oder Batchfile, unter UNIX **Shellskript** und bei manchen Verfassern Kommandoprozedur, Makro oder Makrobefehl. Ein **Skript** ist ganz allgemein ein nicht zu umfangreiches Programm in Form eines lesbaren Textfiles, das ohne vorangehende, von der Ausführung getrennte Übersetzung vom System ausgeführt wird. Man sagt, ein Skript werde interpretiert, nicht erst übersetzt und anschließend ausgeführt wie C/C++-Programme meistens. Außer Shellskripts gibt es eine Vielzahl weiterer Skripts, beispielsweise in Perl oder awk.

Es ist nicht selbstverständlich, aber zweckmäßig, für die Shellskripts dieselbe Kommandosprache zu verwenden wie im Dialog. Der Teil der Shell, der Shellskripts abarbeitet, wird auch als Abwickler bezeichnet. Es gibt weitere Skriptsprachen – vor allem Perl (nicht Pearl, das ist eine andere Geschichte) – anstelle der Shellsprache. Shellskripts dürfen geschachtelt werden (ohne `call` wie in PC-DOS). Externe UNIX-Kommandos sind teils unlesbare kompilierte Programme, teils lesbare Shellskripts.

Es gibt zwei Wege, ein Shellskript auszuführen. Falls es nur lesbar, aber nicht ausführbar ist, übergibt man es als Argument einer Subshell:

```
sh shellskript
```

Ist es dagegen les- und ausführbar, reicht der Aufruf mit dem Namen allein:

```
shellskript
```

Bei der ersten Möglichkeit kann man eine andere als die augenblickliche Sitzungsshell aufrufen, also beispielsweise Bourne statt Korn. Es soll auch leichte Unterschiede in der Vererbung der Umgebung geben, die Literatur – so weit wie wir sie kennen – hält sich mit klaren Aussagen zurück. Experimentell konnten wir nur einen Unterschied hinsichtlich der Umgebungsvariablen EDITOR feststellen.

Der Witz an den Shellskripts ist, daß sie weit mehr als nur Programmaufrufe enthalten dürfen. Die Shells verstehen eine Sprache, die an BASIC heranreicht; sie sind programmierbar. Es gibt Variable, Schleifen, Bedingungen, Ganzzahlarithmetik, Zuweisungen, Funktionen, nur keine Gleitkommarechnung[27]. Die Syntax gleicht einer Mischung von BASIC und C. Man muß das Referenz-Handbuch oder die man-Seite zur Shell sorgfältig lesen, gerade wegen der Ähnlichkeiten. Die Shells sind ziemlich pingelig, was die Schreibweise (Syntax) anbetrifft: manchmal müssen Leerzeichen stehen, ein anderes Mal dürfen keine stehen. Oft hilft ein bißchen Experimentieren weiter. **Kommentar** wird mit einem Doppelkreuz eingeleitet, das bis zum Zeilenende wirkt. Hier ein einfaches Shellskript zum Ausdrucken von man-Seiten:

```
# Shellscript prman zum Drucken von man-Seiten
man $1 | col -b | /usr/local/bin/lf2cl | lp -dlp9
```

Quelle 2.6 : Shellskript zum Drucken von man-Seiten

Zuerst wird das Kommando `man(1)` mit dem ersten Argument der Kommandozeile (Positionsparameter) aufgerufen, so wie man es von Hand eingeben würde. Das Filter `col -b` wirft alle Backspaces hinaus, das selbstgeschriebene Filter `lf2cl` stellt der man-Seite einige Steuerbefehle für den Drucker voran und ersetzt jedes Line-Feed-Zeichen durch das Paar Carriage-Return, Line-Feed, wie es auf die PC-DOS-Welt eingestellte Drucker erwarten. Schließlich geht der Text zu einem Drucker. Das selbstgeschriebene Filter ist ein einfaches C-Programm:

```
/* Programm lf2cl zum Umwandeln von Line-Feed nach
   Carriage-Return, Line-Feed. Dazu Voranstellen von
   PCL-Escape-Folgen zur Druckersteuerung vor den Text.
   Programm soll als Filter in Pipe eingefuegt werden. */

#define DINA4 "Esc&l26A"   /* Escape-Sequenzen */
#define LINES "Esc&l66F"   /* 66 Zeilen/Seite */
#define LRAND "Esc&a2L"    /* 2 Zeichen einruecken */

#include <stdio.h>

int main()
{
```

[27]Die Korn-Shell beherrscht seit 1993 auch Gleitkommaarithmetik. Im Interesse der Portabilität der Shellskripts sollte man jedoch möglichst wenig Gebrauch davon machen.

```
        int c;

        printf(DINA4);
        printf(LINES);
        printf(LRAND);

        while ((c = getchar()) != EOF)
                switch (c) {
                case  10:
                        putchar(13);
                        putchar(10);
                        break;
                default:
                        putchar(c);
                }
return 0;
}
```

Quelle 2.7 : C-Programm zum Ersetzen eines Line-Feed-Zeichens durch das Paar Carriage-Return, Line-Feed; ferner Voranstellen einiger Drucker-Steuerberfehle

Shellskript und C-Programm sind nicht die Hohe Schule der Programmierung, aber hilfreich. Das folgende Beispiel zeigt, wie man eine längere Pipe in ein Shellskript verpackt:

```
# Shellscript frequenz, Frequenzwoerterliste
cat $* |
tr "[A-Z]" "[a-z]" |
tr -c "[a-z]" "[\012*]" |
sort |
uniq -c |
sort -nr
```

Quelle 2.8 : Shellskript Frequenzwörterliste

Dieses Shellskript – in einer Datei namens `frequenz` – nimmt die Namen von einem oder mehreren Textfiles als Argument (Positionsparameter) entgegen, liest die Dateien mittels `cat`, ersetzt alle Großbuchstaben durch Kleinbuchstaben, ersetzt weiterhin jedes Zeichen, das kein Kleinbuchstabe ist, durch ein Linefeed (das heißt schreibt jedes Wort in eine eigene Zeile), sortiert das Ganze alphabetisch, wirft mit Hilfe von `uniq` mehrfache Eintragungen hinaus, zählt dabei die Eintragungen und sortiert schließlich die Zeilen nach der Anzahl der Eintragungen, die größte Zahl zuvorderst. Die Anführungszeichen verhindern, dass die Shell auf dumme Gedanken kommt. Ein solches Skript entwickelt und testet man schrittweise. Der Aufruf des Skripts erfolgt mit `frequenz filenames`. Es ist zugleich ein schönes Beispiel dafür, wie man durch eine Kombination einfacher Werkzeuge eine komplexe Aufgabe löst. Das Zurückführen der verschiedenen Formen eines Wortes auf die Grundform (Infinitiv, Nominativ) muß von Hand geleistet werden, aber einen großen und stumpfsinnigen Teil der Arbeit beim Aufstellen einer Frequenzwörterliste erledigt unser pfiffiges Werkzeug.

Bereinigt man unser Vorwort (ältere Fassung, nicht nachzählen) von allen LaTeX-Konstrukten und bearbeitet es mit `frequenz`, so erhält man eine Wörterliste, deren Beginn so aussieht:

```
  16 der
  16 und
   9 das
   9 die
   8 wir
   7 mit
   7 unix
   6 fuer
   6 in
   6 man
```

Solche Frequenzwörterlisten verwendet man bei Stiluntersuchungen, zum Anlegen von Stichwortverzeichnissen und beim Lernen von Fremdsprachen. Auch zum Nachweis von Plagiaten lassen sie sich einsetzen, da eine häufige Übereinstimmung bei selten vorkommenden Wörtern zwischen zwei Texten den Verdacht auf einen gemeinsamen Verfasser nährt. Für diesen Zweck werden auch Wortgruppen (Paare, Tripel oder Trigramme) untersucht.

Auf Variable greift man in einem Shellskript zurück, indem man ein Dollarzeichen vor ihren Namen setzt. Das Shellskript

```
print TERM
print $TERM
print TERM = $TERM
```

schreibt erst die Zeichenfolge `TERM` auf den Bildschirm und in der nächsten Zeile den Inhalt der Variablen `TERM`, also beispielsweise `hp2393`. Die dritte Zeile kombiniert beide Ausgaben. Weiterhin kennen Shellskripts noch **benannte Parameter** – auch Schlüsselwort-Parameter geheißen – und **Positionsparameter**. Benannte Parameter erhalten ihren Wert durch eine Zuweisung

```
x=3
P1=lpjet
```

während die Positionsparameter von der Shell erzeugt werden. Ihre Namen und Bedeutungen sind:

- `$0` ist das erste Glied der Kommandozeile, also das Kommando selbst ohne Optionen oder Argumente,
- `$1` ist das zweite Glied der Kommandozeile, also eine Option oder ein Argument,
- `$2` ist das dritte Glied der Kommandozeile usw.
- `$#` ist die Anzahl der Positionsparameter,

- `$*` ist die gesamte Kommandozeile ohne das erste Glied `$0`, also die Folge aller Optionen und Argumente,
- `$?` ist der Rückgabewert des jüngsten Kommandos.

Die Bezifferung der Positionsparameter geht bis 9, die Anzahl der Glieder der Kommandozeile ist nahezu unbegrenzt. Die Glieder jenseits der Nummer 9 werden in einem Sumpf verwahrt, aus dem sie mit einem `shift`-Kommando herausgeholt werden können. Hier ein Shellskript, das zeigt, wie man auf Umgebungsvariable und Positionsparameter zugreift:

```
# Shellscript posparm zur Anzeige von Umgebungsvariablen
# und Positionsparametern, 30.08.91

print Start $0
x=4711
print $*
print $#
print $1
print $2
print ${9:-nichts}
print $x
print $TERM
print Ende $0
```

Quelle 2.9 : Shellskript zur Anzeige von Positionsparametern

Nun ein umfangreicheres Beispiel. Das Shellskript `userlist` wertet die Dateien `/etc/passwd` und `/etc/group` aus und erzeugt zwei Benutzerlisten, die man sich ansehen oder ausdrucken kann:

```
# Shellscript userlist, 30. Okt. 86

# Dieses Shellskript erzeugt eine formatierte Liste der
# User und schreibt sie ins File userlist. Voraussetzung
# ist, dass die Namen der User aus mindestens einem Buch-
# staben und einer Ziffer bestehen. Usernamen wie root,
# bin, who, gast werden also nicht in die Liste auf-
# genommen. Die Liste ist sortiert nach der UID. Weiterhin
# erzeugt das Skript eine formatierte Liste aller Gruppen
# und ihrer Mitglieder und schreibt sie ins File grouplist.

# cat liest /etc/passwd
# cut schneidet die gewuenschten Felder aus
# grep sortiert die gewuenschten Namen aus
# sort sortiert nach der User-ID
# sed ersetzt die Doppelpunkte durch control-i (tabs)
# expand ersetzt die tabs durch spaces

print Start /etc/userlist

print "Userliste `date '+%d. %F %y'`\n" > userlist

cat /etc/passwd | cut -f1,3,5 -d: |
grep '[A-z][A-z]*[0-9]' | sort +1.0 -2 -t: |
```

```
sed -e "s/[:]/  /g" | expand -12 » userlist

print "\n`cat userlist | grep '[A-z][A-z]*[0-9]' |
cut -c13-15 | uniq |
wc -l` User. Userliste beendet" » userlist

# cat liest /etc/group
# cut schneidet die gewuenschten Felder aus
# sort sortiert numerisch nach der Group-ID
# sed ersetzt : oder # durch control I (tabs)
# expand ersetzt tabs durch spaces

print "Gruppenliste `date '+%d. %F %y'`\n" > grouplist

cat /etc/group | cut -f1,3,4 -d: |
sort -n +1.0 -2 -t: | sed -e "s/:/  /g" |
sed -e "s/#/    /g" | expand -12 » grouplist

print "\nGruppenliste beendet" » grouplist

print Ende userlist
```

Quelle 2.10 : Shellskript zur Erzeugung einer Benutzerliste

Das folgende Shellskript schreibt ein Menü auf den Bildschirm und wertet die Antwort aus, wobei man statt der Ausgabe mittels echo oder print irgendetwas Sinnvolles tun sollte:

```
# Shellscript menu zum Demonstrieren von Menues, 30.08.91

clear
print "\n\n\n\n\n\n"
print "\tMenu"
print "\t====\n\n\n"
print "\tAuswahl 1\n"
print "\tAuswahl 2\n"
print "\tAuswahl 3\n\n\n"
print "\tBitte Ziffer eingeben: \c"; read z
print "\n\n\n"
case $z in
    1) print "Sie haben 1 gewaehlt.\n\n";;
    2) print "Sie haben 2 gewaehlt.\n\n";;
    3) print "Sie haben 3 gewaehlt.\n\n";;
    *) print "Ziffer unbekannt.\n\n";;
esac
```

Quelle 2.11 : Shellskript für ein Menü

Im obigen Beispiel wird die **Auswahl** case - esac verwendet, die der switch-Anweisung in C entspricht. Es gibt weiterhin die **Bedingung** oder **Verzweigung** mit if - then - else - fi, die das folgende Beispiel zeigt. Gleichzeitig wird Arithmetik mit ganzen Zahlen vorgeführt:

```
#!/bin/ksh
```

```
# Shellscript primscript zur Berechnung von Primzahlen

typeset -i ende=100     # groesste Zahl, max. 3600
typeset -i z=5          # aktuelle Zahl
typeset -i i=1          # Index von p
typeset -i p[500]       # Array der Primzahlen, max. 511
typeset -i n=2          # Anzahl der Primzahlen

p[0]=2; p[1]=3          # die ersten Primzahlen

while [ z -le ende ]    # die [] muessen von Leerzeichen
                        # umgeben sein (Alias fuer test)
do
        if [ z%p[i] -eq 0 ]      # z teilbar
        then
                z=z+2
                i=1
        else                     # z nicht teilbar
                if [ p[i]*p[i] -le z ]
                then
                        i=i+1
                else             # Primzahl gefunden
                        p[n]=z; n=n+1
                        z=z+2
                        i=1
                fi
        fi
done

i=0                     # Ausgabe des Arrays
while [ i -lt n ]
do
        print ${p[i]}
        i=i+1
done

print Anzahl: $n
```

Quelle 2.12 : Shellskript zur Berechnung von Primzahlen

Die erste Zeile beginnt mit einem Kommentarzeichen # gefolgt von einem Ausrufezeichen. An dieser und nur an dieser Stelle lässt sich auf die gezeigte Weise ein bestimmter Kommandointerpreter auswählen. Eine geschachtelte Verzweigung wie in obigem Shellskript darf auch kürzer mit `if - then - elif - then - else - fi` geschrieben werden. Man gewinnt jedoch nicht viel damit.

Neben `if` kennt die Shell zwei weitere Bedingungsoperatoren. Die Zeilen:

```
ls && ps
false && ps
```

sind zu verstehen als *Führe das erste Kommando aus. Bei Erfolg des ersten Kommandos führe anschließend das zweite Kommando aus.* Da `ls` eigentlich immer erfolgreich ist, liefert die erste Zeile die Ausgabe von `ls`, gefolgt von der Ausgabe von ps. Das Kommando false hat immer Misserfolg (Rückgabewert 1), also wird in der zweiten Zeile ps nie ausgeführt. Die Zeilen:

```
false || ps
true || ps
```

sind zu lesen als *Führe das erste Kommando aus. Bei Misserfolg des ersten Kommandos führe anschließend das zweite Kommando aus*. In der ersten Zeile wird ps ausgeführt, da false Misserfolg hat. In der zweiten Zeile hat true Erfolg, also wird ps nicht beachtet. Die Zeilen dürfen verlängert werden:

```
false || true && ps
```

Was ereignet sich hier? In realen Skripts stehen statt true oder false Kommandos, die nutzbringende Taten verrichten, aber für die Beispiele ist es so am einfachsten.

Die for-**Schleife** hat in Shellskripts eine andere Bedeutung als in C. Im folgenden Shellskript ist sie so aufzufassen: für die Argumente in dem Positionsparameter $* (der Name user ist beliebig) führe der Reihe nach die Kommandos zwischen do und done aus.

```
# Shellscript filecount zum Zaehlen der Files eines Users

for user in $*
do
print $user `find /mnt -user $user -print | wc -l`
done
```

Quelle 2.13 : Shellskript zum Zählen der Dateien eines Benutzers

Es gibt weiterhin die while-**Schleife** mit while - do - done, die der gleichnamigen Schleife in anderen Programmiersprachen entspricht. Auf while folgt eine Liste von Kommandos, deren Ergebnis entweder true oder false ist (also nicht ein logischer Ausdruck wie in den Programmiersprachen). true(1) ist hier kein logischer oder boolescher Wert, sondern ein externes UNIX-Kommando, das eine Null (= true) zurückliefert (entsprechend auch false(1)):

```
# Shellscript mit Funktion zum Fragen, 21.05.1992
# nach Bolsky + Korn, S. 183, 191

# Funktion frage

function frage
{
typeset -l antwort          # Typ Kleinbuchstaben
while true
        do      read "antwort?$1" || return 1
                case $antwort in
                j|ja|y|yes|oui) return 0;;
                n|nein|no|non)  return 1;;
                *) print 'Mit j oder n antworten';;
```

```
                esac
        done
}

# Anwendung der Funktion frage

while frage 'Weitermachen? '
do
        date    # oder etwas Sinnvolleres
done
```

Quelle 2.14 : Shellskript mit einer Funktion zum Fragen

Eine Schleife wird abgebrochen, wenn

- die Rücksprung- oder Eintrittsbedingung nicht mehr erfüllt ist oder
- im Rumpf der Schleife das shellinterne Kommando `exit`, `return`, `break` oder `continue` erreicht wird.

Die Kommandos zeigen unterschiedliche Wirkungen. `exit` gibt die Kontrolle an das aufrufende Programm (Sitzungsshell) zurück. Außerhalb einer Funktion hat `return` die gleiche Wirkung. `break` beendet die Schleife, das Shellskript wird nach der Schleife fortgesetzt wie bei einer Verletzung der Bedingung. `continue` hingegen führt zu einem Rücksprung an den Schleifenanfang. Für die gleichnamigen C-Anweisungen gilt dasselbe.

Shellskripts lassen sich durch **Funktionen** strukturieren, die sogar rekursiv aufgerufen werden dürfen, wie das folgende Beispiel zeigt:

```
# Shellscript hanoiscript (Tuerme von Hanoi), 25.05.1992
# Aufruf hanoi n mit n = Anzahl der Scheiben

# nach Bolsky + Korn S. 84, veraendert
# max. 16 Scheiben, wegen Zeitbedarf

# Funktion, rekursiv (selbstaufrufend)

function fhanoi
{
        typeset -i x=$1-1
        ((x>0)) && fhanoi $x $2 $4 $3
        print "\tvon Turm $2 nach Turm $3"
        ((x>0)) && fhanoi $x $4 $3 $2
}

# Hauptscript

case $1 in
[1-9] | [1][0-6])
        print "\nTuerme von Hanoi (Shellscript)"
        print "Start Turm 1, Ziel Turm 2, $1 Scheiben\n"
        print "Bewege die oberste Scheibe"
        fhanoi $1 1 2 3;;
*) print "Argument zwischen 1 und 16 erforderlich"
   exit;;
```

```
esac
```

Quelle 2.15 : Shellskript Türme von Hanoi, rekursiver Funktionsaufruf

Die Türme von Hanoi sind ein Spiel und ein beliebtes Programmbeispiel, bei dem ein Stapel unterschiedlich großer Scheiben von einem Turm auf einen zweiten Turm gebracht werden soll, ein dritter Turm als Zwischenlager dient, mit einem Zug immer nur eine Scheibe bewegt werden und niemals eine größere Scheibe über einer kleineren liegen darf. Das Spiel wurde 1883 von dem französischen Mathematiker FRANÇOIS EDUOUARD ANATOLE LUCAS erdacht. Im obigen Shellskript ist die Anzahl der Scheiben auf 16 begrenzt, weil mit steigender Scheibenzahl die Zeiten lang werden (Anzahl der Züge minimal $2^n - 1$).

Das Hauptskript ruft die Funktion `fhanoi` mit vier Argumenten auf. Das erste Argument ist die Anzahl der Scheiben, die weiteren Argumente sind Start-, Ziel- und Zwischenturm. Die Funktion `fhanoi` setzt die Integervariable x auf den um 1 verminderten Wert der Anzahl, im Beispiel also zunächst auf 2. Diese Variable begrenzt die Rekursionstiefe. Ist der Wert des ersten Argumentes im Aufruf bei 1 angekommen, ruft sich die Funktion nicht mehr auf, sondern gibt nur noch aus. Die Zeile:

```
((x>0)) && fhanoi $x $2 $4 $3
```

ist in der Korn-Shell so zu verstehen:

- berechne den Wert des booleschen Ausdrucks `x > 0`,
- falls TRUE herauskommt, rufe die Funktion `fhanoi` mit den jeweiligen Argumenten auf, wobei `$2` das zweite Argument ist usw.

Schreiben wir uns die Folge der Funktionsaufrufe untereinander, erhalten wir:

```
fhanoi 3 1 2 3
      fhanoi 2 1 3 2
            fhanoi 1 1 2 3 -> print 1 2
      print 1 3
            fhanoi 1 2 3 1 -> print 2 3
print 1 2
      fhanoi 2 3 2 1
            fhanoi 1 3 1 2 -> print 3 1
      print 3 2
            fhanoi 1 1 2 3 -> print 1 2
```

Die Ausgabe des Skripts für $n = 3$ sieht folgendermaßen aus:

```
Tuerme von Hanoi (Shellskript)
Start Turm 1, Ziel Turm 2, 3 Scheiben

Bewege die oberste Scheibe
von Turm 1 nach Turm 2
```

```
von Turm 1 nach Turm 3
von Turm 2 nach Turm 3
von Turm 1 nach Turm 2
von Turm 3 nach Turm 1
von Turm 3 nach Turm 2
von Turm 1 nach Turm 2
```

Für $n = 1$ ist die Lösung trivial, für $n = 2$ offensichtlich, für $n = 3$ überschaubar, sofern die Sterne günstig und die richtigen Getränke in Reichweite stehen. Bei größeren Werten muß man systematisch vorgehen. Ein entscheidender Moment ist erreicht, wenn nur noch die unterste (größte) Scheibe im Start liegt und sich alle übrigen Scheiben im Zwischenlager befinden, geordnet natürlich. Dann bewegen wir die größte Scheibe ins Ziel. Der Rest ist nur noch, den Stapel vom Zwischenlager ins Ziel zu bewegen, eine Aufgabe, die wir bereits beim Transport der $n-1$ Scheiben vom Start ins Zwischenlager bewältigt haben. Damit haben wir die Aufgabe von n auf $n-1$ Scheiben reduziert. Das Rezept wiederholen wir, bis wir bei $n = 2$ angelangt sind. Wir ersetzen also eine vom Umfang her nicht zu lösende Aufgabe durch eine gleichartige mit geringerem Umfang so lange, bis die Aufgabe einfach genug geworden ist. Das Problem liegt darin, sich alle angefangenen, aber noch nicht zu Ende gebrachten Teilaufgaben zu merken, aber dafür gibt es Computer. Mit der Entdeckung eines Algorithmus, der mit Sicherheit und in kürzestmöglicher Zeit zum Ziel führt, ist der Charakter des Spiels verloren gegangen, es ist nur noch ein Konzentrations- und Gedächtnistest. Beim Schach liegen die Verhältnisse anders.

Dieses Programmle haben wir ausführlich erklärt, weil Rekursionen für manchen Leser ungewohnt sind. Versuchen Sie, die Aufgabe ohne Rekursion zu lösen (nicht alle Programmiersprachen kennen die Rekursion) und suchen Sie mal im WWW nach *Towers of Hanoi* und *recurs* und ihren deutschen Übersetzungen.

Eine Shellfunktion wird von der aktuellen Shell ausgeführt und teilt daher Variable und die weitere Umgebung mit dieser. Der Aufruf eines Shellskripts anstelle einer Funktion führt zur Erzeugung einer Subshell mit eigenem Leben. Im Gegensatz zu einem Punkt-Skript kann eine Shellfunktion Positionsparameter speichern und lokale Variable verwenden. Die drei Wege, aus einem Shellskript ein Programmmodul aufzurufen, unterscheiden sich deutlich in ihren Arbeitsbedingungen. Vermutlich ist die Shellfunktion am einfachsten zu überschauen.

Beim Anmelden werden automatisch zwei Shellskripts ausgeführt, die Sie sich als Beispiele ansehen sollten: `/etc/profile` wird für jeden Benutzer ausgeführt, das Skript `.profile` im Home-Verzeichnis für die meisten.

```
# /etc/profile $Revision: 64.2, modifiziert 02.10.90

# Default system-wide profile (/bin/ksh initialization)
# This should be kept to the minimum every user needs.
```

```
  trap "" 1 2 3            # ignore HUP, INT, QUIT

  PATH=/rbin:/usr/rbin:    # default path
  CDPATH=:..:$HOME
  TZ=MEZ-1

  TTY=`/bin/tty`           # TERM ermitteln
  TN=`/bin/basename $TTY`
  TERM=`/usr/bin/fgrep $TN /etc/ttytype | /usr/bin/cut -f1`

  if [ -z "$TERM" ]        # if term is not set,
  then
    TERM=vt100             # default terminal type
  fi

  TMOUT=500
  LINES=24                 # fuer tn3270
  PS1="mvmhp "             # Prompt
  GNUTERM=hp2623A          # fuer gnuplot
  HOSTALIASES=/etc/hostaliases

  export PATH CDPATH TZ TERM TMOUT LINES PS1
  export GNUTERM HOSTALIASES

# initialisiere Terminal gemaess TERMINFO-Beschreibung

  /usr/bin/tset -s

# set erase to ^H , kill to ^X , intr to ^C, eof to ^D

  /bin/stty erase "^H" kill "^X" intr "^C" eof "^D"

# Set up shell environment

  trap clear 0

# Background-Jobs immer mit nice und andere Optionen

  set -o bgnice -o ignoreeof

# Schirm putzen und Begruessung

  /usr/rbin/clear
  print " * Willkommen .... * "

  if [ $TN = "tty2p4" ]        # Modem
  then
    print
    /usr/local/bin/speed
  fi

  if [ $LOGNAME != root -a $LOGNAME != adm ]
  then

    print
```

```
    if [ -f /etc/motd ]
    then
      /bin/cat /etc/motd  # message of the day.
    fi

    if [ -f /usr/bin/news ]
    then /usr/bin/news      # display news.
    fi

    print "\nHeute ist        `/rbin/zeit`"

    if [ -r $HOME/.logdat -a -w $HOME/.logdat ]
    then
      print "Letzte Anmeldung \c"; /bin/cat $HOME/.logdat
    fi
    /bin/zeit > $HOME/.logdat

    print "\nIhr Home-Directory  $HOME  belegt \c"
    DU=`/bin/du -s $HOME | /usr/bin/cut -f1`
    print "`/bin/expr $DU / 2` Kilobyte.\n"
    unset DU

    /bin/sleep 4

    /usr/bin/elm -azK
    print

  fi

  cd
  umask 077

  /bin/mesg y 2>/dev/null

  /usr/rbin/clear

  if [ $LOGNAME != gast ]
  then
    print y | /bin/ln /mnt/.profile $HOME/.profile 2>/dev/null
    /bin/ln /mnt/.exrc    $HOME/.exrc 2>/dev/null
  fi

  trap 1 2 3      # leave defaults in environment
```

Quelle 2.16 : Shellskript /etc/profile

Das Shellskript `.profile` in den Home-Verzeichnissen dient persönlichen Anpassungen. Auf unserem System wird es allerdings vom System-Manager verwaltet, da es einige wichtige Informationen enthält, die der Benutzer nicht ändern soll. Die Datei `.logdat` speichert den Zeitpunkt der Anmeldung, sodass man bei einer erneuten Anmeldung feststellen kann, wann die vorherige Anmeldung stattgefunden hat, eine Sicherheitsmaßnahme.

```
# .profile zum Linken/Kopieren in die HOME-Directories
```

```
# ausser gast und dergleichen. 1993-02-16

EDITOR=vi
FCEDIT=vi
TMOUT=1000
PATH=/bin:/usr/bin:/usr/local/bin:$HOME/bin::

# PS1="mvmhp> "              # Prompt
# PS1='${PWD#$HOME/}> '
PS1='${PWD##*/}> '

export FCEDIT PATH PS1

alias h='fc -l'

if [ -f .autox ]
then
    . .autox
fi
```

Quelle 2.17 : Shellskript /etc/.profile

In dem obigen Beispiel `/etc/.profile` wird ein weiteres Skript namens `.autox` mit einem vorangestellten und durch einen Zwischenraum (Space) abgetrennten Punkt aufgerufen. Dieser Punkt ist ein Shell-Kommando und hat nichts mit dem Punkt von `.autox` oder `.profile` zu tun. Als Argument übernimmt der Punktbefehl den Namen eines Shellskripts. Er bewirkt, daß das Shellskript nicht von einer Subshell ausgeführt wird, sondern von der Shell, die den Punktbefehl entgegennimmt. Damit ist es möglich, in dem Shellskript beispielsweise Variable mit Wirkung für die derzeitige Shell zu setzen, was in einer Subshell wegen der Unmöglichkeit der Vererbung von Kinderprozessen rückwärts auf den Elternprozess nicht geht. Ein mit dem Punkt-Befehl aufgerufenes Shellskript wird als **Punktskript** bezeichnet, obwohl der Aufruf das Entscheidende ist, nicht das Skript.

Für den Prompt stehen in `.profile` drei Möglichkeiten zur Wahl. Die erste setzt den Prompt auf einen festen String, den Netznamen der Maschine. Die zweite verwendet den Namen des aktuellen Verzeichnisses, verkürzt um den Namen des Home-Verzeichnisses. Die dritte, nicht auskommentierte zeigt den Namen des Arbeits-Verzeichnisses ohne die übergeordneten Verzeichnisse an.

Das waren einige Shellskripts, die vor Augen führen sollten, was die Shell leistet. Der Umfang der Shellsprache ist damit noch lange nicht erschöpft. Die Möglichkeiten von Shellskripts voll auszunutzen erfordert eine längere Übung. Die Betonung liegt auf voll, einfache Shellskripts schreibt man schon nach wenigen Minuten Üben.

Wir haben uns vorstehend mit der Korn-Shell `ksh(1)` befaßt, die man heute als die Standardshell ansehen kann (Protest von Seiten der `csh(1)`-Anhänger). Verwenden Sie die Shell, die auf Ihrer Anlage üblich ist, im Zweifelsfall die Bourne-Shell `sh(1)` oder die Bourne-again-Shell `bash(1)`, und wechseln Sie auf eine leistungsfähigere Shell, wenn Sie an die Grenzen Ihrer

Shell stoßen. Die Bourne-Shell kennengelernt zu haben, ist auf keinen Fall verkehrt.

2.5.3 Noch eine Skriptsprache: Perl

Perl[28] ist eine Alternative zur Shell als Skriptsprache (nicht als interaktiver Kommandointerpreter) und vereint Züge von `sh(1)`, `awk(1)`, `sed(1)` und der Programmiersprache C. Sie wurde von LARRY WALL entwickelt und ist optimiert für Textverarbeitung und Systemverwaltung. Perl-Interpreter sind im Netz frei unter der GNU General Public License verfügbar. Einzelheiten sind einem Buch oder der man-Seite (eher schon ein man-Booklet) zu entnehmen, hier wollen wir uns nur an zwei kleinen Beispielen eine Vorstellung von Perl verschaffen. Dazu verwenden wir das in Perl umgeschriebene Shellskript zur Berechnung von Primzahlen.

```
#!/usr/local/bin/perl
# perl-Script zur Berechnung von Primzahlen

$ende = 10000;      # groesste Zahl
$z = 5;             # aktuelle Zahl
$i = 1;             # Index von p
@p = (2, 3);        # Array der Primzahlen
$n = 2;             # Anzahl der Primzahlen

while ($z <= $ende) {
        if ($z % @p[$i] == 0) {       # z teilbar
                $z = $z + 2;
                $i = 1;
        }
        else {                        # z nicht teilbar
                if (@p[$i] * @p[$i] <= $z) {
                        $i++;
                }
                else {
                        @p[$n] = $z;
                        $n++;
                        $z = $z + 2;
                        $i = 1;
                }
        }
}

# Ausgabe des Arrays

$i = 0;
while ($i < $n) {
        print(@p[$i++], "\n");
}

print("Anzahl: ", $n, "\n");
```

[28]Nicht zu verwechseln mit Pearl = Process and Experiment Automation Real-Time Language.

Quelle 2.18 : Perlskript zur Berechnung von Primzahlen

Man erkennt, daß die Struktur des Skripts gleich geblieben ist. Die Unterschiede rühren von syntaktischen Feinheiten her:

- Die erste Zeile *muß* wie angegeben den Perl-Interpreter verlangen. Sie wird *Shebang*-Zeile genannt, zusammengezogen aus *sharp* und *bang*.
- Die Namen von Variablen beginnen mit Dollar, Buchstabe.
- Die Namen von Arrays beginnen mit dem at-Zeichen (Klammeraffe).
- Die Kontrollanweisungen erinnern an C, allerdings *muß* der Anweisungsteil in geschweiften Klammern stehen, selbst wenn er leer ist.
- Zur Ausgabe auf `stdout` wird eine Funktion `print()` verwendet.

Der Perl-Interpreter unterliegt nicht den engen Grenzen des Zahlenbereiches und der Arraygröße der Shell. Die Stellenzahl der größten ganzen Zahl ist maschinenabhängig und entspricht ungefähr der Anzahl der gültigen Stellen einer Gleitkommazahl. Zum Perl-Paket gehören auch Konverter für `awk(1)`- und `sed(1)`-Skripts, allerdings bringt das Konvertieren von Hand elegantere Ergebnisse hervor.

Im zweiten Beispiel soll aus dem Katalog einer Institutsbibliothek die Anzahl der Bücher ermittelt werden. Zu jedem Schriftwerk gehört eine Zeile im Katalog, jede Zeile enthält ein Feld zur Art des Werkes: ”BUC” heißt Buch, ”DIP” Diplomarbeit, ”ZEI” Zeitschrift. Das Perlskript verwendet ein assoziatives Array, dessen Elemente als Index nicht Ganzzahlen, sondern beliebige Strings gebrauchen. Über die Anordnung der Elemente im Array braucht man sich keine Gedanken zu machen. Das Perlskript:

```
#!/usr/local/bin/perl
# perl-Script zum Zaehlen in Buecherliste

# Verwendung eines assoziativen Arrays

%anzahl = ("BUC", 0, "ZEI", 0, "DIP", 0);

# Leseschleife

while ($input = <STDIN>) {
        while ($input =~ /BUC|ZEI|DIP/g) {
                $anzahl{$&} += 1;
        }
}

# Ausgabe

foreach $item (keys(%anzahl)) {
        print("$item: $anzahl{$item}\n");
}
```

Quelle 2.19 : Perlskript zur Ermittlung der Anzahl der Bücher usw. in einem Katalog

In der ersten ausführbaren Zeile wird ein assoziatives Array namens `%anzahl` mit drei Elementen definiert und initialisiert. Die äußere `while`-Schleife liest Zeilen von `stdin`, per Umlenkung mit dem Katalog verbunden. Die innere `while`-Schleife zählt das jeweilige Element des Arrays um 1 hoch, jedesmal wenn in der aktuellen Zeile ein Substring ”BUC” oder ”ZEI” oder ”DIP” gefunden wird. Die Perl-Variable `$&` enthält den gefundenen Substring und wird deshalb als Index ausgenutzt. Die `foreach`-Schleife zur Ausgabe gleicht der gleichnamigen Schleife der C-Shell oder der `for`-Schleife der Bourne-Shell.

Was man mit Shell- oder Perlskripts macht, läßt sich auch mit Programmen – vorzugsweise in C/C++ – erreichen. Was ist besser? Ein Skript ist schnell geschrieben oder geändert, braucht nicht kompiliert zu werden (weil es interpretiert wird), läuft aber langsamer als ein Programm. Ein Skript eignet sich daher für kleine bis mittlere Aufgaben zur Textverarbeitung oder Systemverwaltung, wobei Perl mehr kann als eine Shell. Für umfangreiche Rechnungen (Datenstrukturen, Algorithmen) oder falls die Laufzeit entscheidet, ist ein kompiliertes Programm besser. Oft schreibt man auch zunächst ein Skript, probiert es eine Zeitlang aus und ersetzt es dann durch ein Programm. Gelegentlich spielt die Portierbarkeit auf andere Betriebssysteme eine Rolle. Ein UNIX-Shellskript läuft nur auf Systemen, auf denen eine UNIX-Shell verfügbar ist, Perl setzt den Perl-Interpreter voraus, ein C-Programm läuft auf jedem System, für das ein C-Compiler zur Verfügung steht.

JOHN K. OUSTERHOUT, der Vater der Skriptsprache Tcl, führt in einer Veröffentlichung von 1998 neun Kriterien zur Entscheidung zwischen Sprachen wie C/C++ oder FORTRAN einerseits und Skriptsprachen wie Tcl oder Perl andererseits an:

- Für Skriptsprachen:
  - Die Anwendung verbindet vorgefertigte Komponenten miteinander,
  - die Anwendung manipuliert eine Vielfalt von Dingen,
  - die Anwendung beinhaltet eine grafische Benutzer-Oberfläche,
  - die Anwendung arbeitet viel mit Strings,
  - die Funktionalität der Anwendung entwickelt sich rasch weiter,
  - die Anwendung soll erweiterbar sein,
- für Systemprogrammiersprachen (wie er sie nennt):
  - die Anwendung benötigt komplexe Algorithmen oder Datenstrukturen,
  - die Anwendung verarbeitet große Datenmengen, Geschwindigkeit spielt eine Rolle,
  - die Funktionalität der Anwendung ist sauber definiert und ändert sich nur allmählich.

Zum Glück schließen sich die beiden Sprachtypen nicht gegenseitig aus, sondern ergänzen sich. Wer beide beherrscht, dem steht eine mächtige Programmierumgebung für alle Zwecke zur Verfügung. Und schließlich hat man auch seine Gewohnheiten.

2.5.4 Begriffe Shells

Folgende Begriffe sollten klarer geworden sein:

- Kommando-Interpreter, Shell
- Metazeichen, Jokerzeichen, quoten
- Shell-Skript, Perl-Skript
- stdin, stdout, stderr
- Umgebung
- Umlenkung

Folgende Kommandos sollten beherrscht werden:

- `alias`
- `set`
- `echo`, `print`

2.5.5 Memo Shells

- Die Shell – ein umfangreiches Programm – ist der Gesprächspartner (interaktiver Kommandointerpreter) in einer Sitzung. Es gibt mehrere Shells zur Auswahl, die sich in Einzelheiten unterscheiden.
- Die Shell faßt jede Eingabe als Kommando (internes Kommando oder externes Kommando = Shellskript oder Programm) auf.
- Die Shell stellt für die Sitzung eine Umgebung bereit, die eine Reihe von Werten (Strings) enthält, die von Shellskripts und anderen Programmen benutzt werden.
- Die Shell ist zweitens ein Interpreter für Shellskripts, eine Art von Programmen, die nicht kompiliert werden. Shellskripts können alles außer Gleitkomma-Arithmetik.
- Perl ist eine Skriptsprache alternativ zur Shell als Sprache, nicht als interaktiver Kommandointerpreter. Sie setzt den Perl-Interpreter voraus.

2.5.6 Übung Shells

Melden Sie sich – wie inzwischen gewohnt – unter Ihrem Benutzernamen an. Die folgende Sitzung läuft mit der der Korn-Shell. Die Shells sind umfangreiche Programme mit vielen Möglichkeiten, wir kratzen hier nur ein bißchen an der Oberfläche.

```
set                     (Umgebung anzeigen)
PS1="zz "               (Prompt aendern)
NEU=Unsinn              (neue Variable setzen)
set
pwd                     (Arbeits-Verzeichnis?)
print Mein Arbeits-Verzeichnis ist pwd
                        (Satz auf Bildschirm schreiben)
print Mein Arbeits-Verzeichnis ist `pwd`
                        (Kommando-Substitution)
print Mein Home-Verzeichnis ist $HOME
                        (Shell-Variable aus Environment)
more /etc/profile       (Shellskript anschauen)
more .profile
```

Schreiben Sie mit dem Editor `vi(1)` in Ihr Home-Verzeichnis eine Datei namens `.autox` mit folgendem Inhalt:

```
PS1="KA "
trap "print Auf Wiedersehen!" 0
/usr/bin/clear
print
/usr/bin/banner "   UNIX"
```

und schreiben Sie in Ihre Datei `.profile` folgende Zeilen:

```
if [ -f .autox ]
then
. .autox
fi              (Die Spaces und Punkte sind wichtig. Die Zeilen
                rufen die Datei .autox auf, falls sie existiert.
```

Wenn das funktioniert, richten Sie in `.autox` einige Aliases nach dem Muster von Abschnitt 2.5.1.1 *Kommandointerpreter* auf Seite 89 ein. Was passiert, wenn in `.autox` das Kommando `exit` vorkommt?

Schreiben Sie ein Shellskript namens `showparm` nach dem Muster aus dem vorigen Abschnitt und variieren es. Rufen Sie `showparm` mit verschiedenen Argumenten auf, z. B. `showparm eins zwei drei`.

2.5.7 Fragen Shells

- Welche beiden Aufgaben hat eine Shell?
- Welche beiden Shellfamilien gibt es unter UNIX?
- Wie sieht eine Kommdozeile aus?
- Was macht die Shell mit einer Eingabe?
- Was sind Jokerzeichen in Dateinamen?
- Was sind Metazeichen? Was heißt *quoten*?

- Woraus besteht die Umgebung einer Sitzung?
- Was ist eine Umlenkung?
- Was ist ein Skript?
- Was sind Positionsparameter?
- Was kann man mit Shellskripts *nicht* machen?
- Erklären Sie einige einfache Shellskripts.
- Welche Vor- und Nachteile haben Shellskripts im Vergleich mit kompilierten Programmen?
- Was ist ein Perlskript? Welche Software setzt es voraus?

2.6 Benutzeroberflächen

2.6.1 Lokale Benutzeroberflächen

2.6.1.1 Kommandozeile

Unter einer **Benutzer-Oberfläche** (user interface) versteht man nicht die Haut, aus der man nicht heraus kann, sondern die Art, wie sich ein Terminal (Bildschirm, Tastatur, Maus) dem Benutzer darstellt, wie es ausschaut (look) und wie es auf Eingaben reagiert (feel). Lokal bedeutet nicht-netzfähig, beschränkt auf einen Computer – im Gegensatz zum X Window System.

Im einfachsten Fall tippt man seine Kommandos zeilenweise ein, sie werden auf dem alphanumerischen Bildschirm ge-echot und nach dem Drücken der Return-Taste ausgeführt. Die Ausgabe des Systems erfolgt ebenfalls auf den Bildschirm, Zeile für Zeile nacheinander.

Diese Art der Ein- und Ausgabe heißt **Kommandozeile**. Sie stellt die geringsten Anforderungen an Hard- und Software und ist mit Einschränkungen sogar auf druckenden Terminals (ohne Bildschirm) möglich. Vom Benutzer verlangt sie die Kenntnis der einzugebenden Kommandos und das zielsichere Landen auf den richtigen Tasten. Die Programme bieten einfache Hilfen an, die üblicherweise durch die Tasten h (wie help), ? oder die Funktionstaste F1 aufgerufen werden. Die Kommandozeile ist auch heute noch nicht überflüssig, sondern immer noch der einzige Weg, auf dem man jedes Kommando mit jeder Option erreicht.

Bei UNIX-Kommandos ist es eine gute Gepflogenheit, daß sie – fehlerhaft aufgerufen – einen Hinweis zum richtigen Gebrauch (Usage) geben. Probieren Sie folgende fehlerhafte Eingaben aus, auch mit anderen Kommandos:

```
who -x
who -?
who --help
```

Die leicht gekürzte Antwort sieht so aus:

```
who: illegal option -- x

Usage: who [-rbtpludAasHTqRm] [am i] [utmp_like_file]

r run level
b boot time
t time changes
p processes other than getty or users
l login processes
u useful information
```

Schreibt man selbst Werkzeuge, sollte man wenigstens diese Hilfe einbauen. Eine zusätzliche man-Seite wäre die Krone.

2.6.1.2 Menüs

Ein erster Schritt in Richtung Benutzerfreundlichkeit ist die Verwendung von **Menüs**. Die erlaubten Eingaben werden in Form einer Liste – einem Menü – angeboten, der Benutzer wählt durch Eintippen eines Zeichens oder durch entsprechende Positionierung des Cursors die gewünschte Eingabe aus. Der Cursor wird mittels der Cursortasten oder einer Maus positioniert.

Menüs haben zwei Vorteile. Der Benutzer sieht, was erlaubt ist, und macht bei der Eingabe kaum syntaktische Fehler. Nachteilig ist die beschränkte Größe der Menüs. Man kann nicht mehrere hundert UNIX-Kommandos mit jeweils mehreren Optionen in ein Menü packen. Ein Ausweg – in Grenzen – sind Menü-Hierarchien, die auf höchstens drei Ebenen begrenzt werden sollten, um übersichtlich zu bleiben. Einfache Menüs ohne Grafik und Mausunterstützung stellen ebenfalls nur geringe Anforderungen an Hard- und Software. Ein typisches Einsatzgebiet von Menüs sind Büroanwendungen. Menüs lassen sich nicht als Filter in einer Pipe verwenden, weil `stdin` innerhalb einer Pipe nicht mehr mit der Tastatur, sondern mit `stdout` des vorhergehenden Gliedes verbunden ist.

Für den ungeübten Benutzer sind Menüs eine große Hilfe, für den geübten ein Hindernis. Deshalb sollte man zusätzlich zum Menü immer die unmittelbare Kommandozeilen-Eingabe zulassen. Zu den am häufigsten ausgewählten Punkten müssen kurze Wege führen. Man kann Defaults vorgeben, die nur durch Betätigen der RETURN-Taste ohne weitere Zeichen aktiviert werden. Solche abgekürzten Wege werden auch **Shortcuts** genannt.

Wir haben beispielsweise für die Druckerausgabe ein Menu namens `p` geschrieben, das dem Benutzer unsere Möglichkeiten anbietet und aus seinen Angaben das `lp(1)`-Kommando mit den entsprechenden Optionen zusammenbaut. Der Benutzer braucht diese gar nicht zu kennen. In ähnlicher Weise verbergen wir den Dialog mit unserer Datenbank hinter Menus, die SQL-Skripte aufrufen. Das Eingangsmenü für unsere Datenbank sieht so aus:

```
Oracle-Hauptmenue (1997-03-21 A)
=================
Bibliothek          1
```

```
Buchaltung          2
Personen            3
Projekte            4

Bitte Ziffer eingeben:
```

Nach Eingabe einer gültigen Ziffer gelangt man ins erste Untermenü usf. Hinter dem Menu steckt ein Shellskript mit einer `case`-Anweisung, das letzten Endes die entsprechenden Shell- und SQL-Skripte aufruft. Der Benutzer braucht weder von der Shell noch von SQL etwas zu verstehen. Er bekommt seine Daten nach Wunsch entweder auf den Bildschirm oder einen Drucker.

2.6.1.3 Zeichen-Fenster, curses

Bildschirme lassen sich in mehrere Ausschnitte aufteilen, die **Fenster** oder **Windows** genannt werden. In der oberen Bildschirmhälfte beispielsweise könnte man bei einem Benutzerdialog mittels `write` den eigenen Text darstellen, in der unteren die Antworten des Gesprächspartners. Das UNIX-Kommando `write(1)` arbeitet leider nicht so. Ein anderer Anwendungsfall ist das Korrigieren (Debuggen) von Programmen. In der oberen Bildschirmhälfte steht der Quellcode, in der unteren die zugehörige Fehlermeldung.

Für den C-Programmierer stellt die `curses(3)`-Bibliothek Funktionen zum Einrichten und Verwalten von monochromen, alphanumerischen Fenstern ohne Mausunterstützung zur Verfügung. Die `curses(3)` sind halt schon etwas älter. Ein Beispiel findet sich im Skriptum *Programmieren in C/C++*. Darüber hinaus gibt es weitere, kommerzielle Fenster- und Menübibliotheken, vor allem im PC-Bereich. An die Hardware werden keine besonderen Anforderungen gestellt, ein alphanumerischer Bildschirm mit der Möglichkeit der Cursorpositionierung reicht aus.

Wer seinen Bildschirm mit Farbe und Maus gestalten will, greift zum X Window System (X11) und seinen Bibliotheken. Das kann man auch lernen, aber nicht in einer Viertelstunde.

2.6.1.4 Grafische Fenster

Im Xerox Palo Alto Research Center ist die Verwendung von Menüs und Fenstern weiterentwickelt worden zu einer **grafischen Benutzeroberfläche** (graphical user interface, GUI), die die Arbeitsweise des Benutzers wesentlich bestimmt. Diese grafische Fenstertechnik ist von Programmen wie SMALLTALK und Microsoft Windows sowie von Computerherstellern wie Apple übernommen und verbreitet worden. Wer't mag, dei mag't, un wer't nich mag, dei mag't jo woll nich mägen.

Ein klassisches UNIX-Terminal gestattet die Eröffnung genau einer Sitzung, deren Kontroll-Terminal es dann wird. Damit sind manche Benutzer noch nicht ausgelastet. Sie stellen sich ein zweites und drittes Terminal auf den Tisch und eröffnen auf diesen ebenfalls je eine Sitzung. Unter UNIX können mehrere Sitzungen unter einem Benutzernamen gleichzeitig laufen. Dieses Vorgehen wird begrenzt durch die Tischfläche und die Anzahl der Terminalanschlüsse. Also teilt man ein Terminal in mehrere **virtuelle Terminals** auf, die Fenster oder Windows genannt werden, und eröffnet in jedem Window eine Sitzung. Auf dem Bildschirm gehört jedes Fenster zu einer Sitzung, Tastatur und Maus dagegen können nicht aufgeteilt werden und sind dem jeweils aktiven Fenster zugeordnet. Die Fenster lassen sich vergrößern, verkleinern und verschieben. Sie dürfen sich überlappen, wobei nur das vorderste Fenster vollständig zu sehen ist. Wer viel mit Fenstern arbeitet, sollte den Bildschirm nicht zu klein wählen, 17 Zoll Bildschirmdiagonale ist die untere Grenze. Ein Schreibtisch hat eine Diagonale von 80 Zoll.

Was ein richtiger Power-User ist, der hat so viele Fenster gleichzeitig in Betrieb, daß er für den Durchblick ein Werkzeug wie das **Common Desktop Environment** (CDE) braucht, in der Linux-Welt eine Arbeitsumgebung wie KDE oder GNOME. Diese setzen auf dem X Window System auf und teilen die Fenster in vier oder mehr Gruppen (Arbeitsflächen, Desktops) ein, von denen jeweils eine auf dem Schirm ist. Zwischen den Gruppen wird per Mausklick umgeschaltet. Die Gruppen können beispielsweise

- Allgemeines
- Verwaltung
- Programmieren
- Internet
- Server A
- Server B

heißen und stellen virtuelle Schreibtische für die jeweiligen Arbeitsgebiete dar. Man kann sich sehr an das Arbeiten mit solchen Umgebungen gewöhnen und beispielsweise – ohne es zu merken – dasselbe Textfile gleichzeitig in mehreren Fenstern oder Arbeitsflächen editieren. Ein gewisser Aufwand an Hard- und Software (vor allem Arbeitsspeicher) steckt dahinter, aber sechs Schreibtische sind ja auch was. Den anklickbaren Papierkorb gibt es gratis dazu.

2.6.1.5 Multimediale Oberflächen

Der Mensch hat nicht nur Augen und Finger, sondern auch noch Ohren, eine Nase, ein Zunge und eine Stimme. Es liegt also nahe, zum Gedankenaustausch mit dem Computer nicht nur den optischen und mechanischen Übertragungsweg zu nutzen, sondern auch den akustischen und zumindest in Richtung vom Computer zum Benutzer auch dessen Geruchssinn[29]. Letzteres wird seit altersher bei der ersten Inbetriebnahme elektronischer Geräte aller Art gemacht (smoke test), weniger während des ordnungsgemäßen Betriebes. Es gibt bereits Aroma-Diffusoren mit USB-Anschluss, die beim Spielen mit

[29] Nachricht in Markt & Technik vom 31. März 1994: IBM entwickelt künstliche Nase. Nachricht in der c't 21/1998: In der Ohio State University erkennt eine elektronische Nase Käsesorten am Geruch. Vielleicht fordert Sie ihr Computer demnächst auf, den Kaffee etwas stärker anzusetzen.

Autorennen-Simulatoren den Duft von Benzin und Gummi verbreiten könnten (c't Nr. 21/2004, S. 70). Der akustische Weg wird in beiden Richtungen vor allem in solchen Fällen genutzt, in denen Augen oder Finger anderweitig beschäftigt sind (Fotolabor, Operationssaal) oder fehlen. In den nächsten Jahren wird die Akustik an Bedeutung gewinnen. Über die Nutzung des Geschmackssinnes wird noch nachgedacht (wie soll das Terminal aussehen bzw. ausschmecken?).

Im Ernst: unter einer multimedialen Oberfläche versteht man bewegte Grafiken plus Ton, digitales Kino mit Dialog sozusagen. Der Computer gibt nicht nur eine dürre Fehlermeldung auf den Bildschirm aus, sondern läßt dazu *That ain't right* mit FATS WALLER am Piano ertönen. Lesen Sie Ihre Email, singt im Hintergrund ELLA FITZGERALD *Email special*. Umgekehrt beantworten Sie die Frage des `vi(1)`, ob er ohne Zurückschreiben aussteigen soll, nicht knapp und bündig mit einem Ausrufezeichen, sondern singen wie EDITH PIAF *Je ne regrette rien*. Der Blue Screen von Microsoft Windows läßt sich leichter ertragen, wenn dazu LOUIS ARMSTRONG sein *Blueberry Hill* tutet und gurgelt. Die eintönige Arbeit am Terminal entwickelt sich so zu einem anspruchsvollen kulturellen Happening. Die Zukunft liegt bei multisensorischen Schnittstellen, die mit Menschen auf zahlreichen kognitiven und physiologischen Ebenen zusammenarbeiten (Originalton aus einem Prospekt).

2.6.1.6 Software für Behinderte

Das Thema *Behinderte und Computer* hat mehrere Seiten. An Behinderungen kommen in Betracht:

- Behinderungen des Sehvermögens
- Behinderungen des Hörvermögens
- Behinderungen der körperlichen Beweglichkeit
- Beeinträchtigungen der Konzentrationsfähigkeit oder des Gedächtnisses

Das **Sehvermögen** kann in vielerlei Hinsicht beeinträchtigt sein: mangelnde Sehschärfe, die nicht in jedem Fall durch Hilfsmittel (Brille) korrigiert werden kann, Farbsehschwächen – insbesondere die bei Männern verbreitete Rot-Grün-Schwäche – Probleme mit Lichtkontrasten oder schnell bewegten Bildern bis hin zu völliger Blindheit. Jeder, der lange genug lebt, stellt ein Nachlassen seines Sehvermögens fest. Für das **Hörvermögen** gilt im Prinzip dasselbe, nur spielt das Hören für die Arbeit am Computer keine so bedeutende Rolle.

Bei Einschränkungen der **Beweglichkeit** ist zu unterscheiden zwischen solchen, die die ganze Person betreffen (Rollstuhlfahrer), und Behinderungen einzelner Gliedmaßen, vor allem der Arme und Hände. Man versuche einmal, Tastatur und Maus oder Rollkugel mit dicken Fausthandschuhen zu betätigen.

Die Benutzung eines Computers durch einen Behinderten erfordert eine Anpassung der Hardware, insbesondere des Terminals, und Rücksicht seitens der Software einschließlich der im Netz angebotenen Informationen (WWW und andere Dienste). Andererseits kann ein Computer als Hilfsmittel bei der Bewältigung alltäglicher Probleme dienen, zum Beispiel beim Telefonieren. Ein bekannter behinderter Benutzer ist der Physiker STEPHEN HAWKING, der sich mit seiner Umwelt per Computer verständigt.

Im Netz finden sich einige Server, die Software und Informationen für Behinderte sammeln:

- `ftp://ftp.th-darmstadt.de/pub/machines/ms-dos/SimTel/msdos/`
- `ftp://ftp.tu-ilmenau.de/pub/msdos/CDROM1/msdos/handicap/`
- `http://seidata.com/~marriage/rblind.html`

sowie die Newsgruppen `misc.handicap` und `de.soc.handicap`, allerdings mit mehr Fragen als Antworten. In der Universität Karlsruhe bemüht sich das *Studienzentrum für Sehgeschädigte*, diesen das Studium der Informatik zu erleichtern: `http://szswww.ira.uka.de/`.

Das Internet und darin besonders das World Wide Web spielen heute eine gewichtige Rolle im Berufs- und Privatleben. Bei der Gestaltung von Fenstern, Webseiten und ähnlichen Informationen kann man mit wenig zusätzlichem Aufwand Behinderten das Leben erleichtern, ohne auf die neuesten Errungenschaften von Grafik und HTML verzichten zu müssen. Auf Englisch lautet das Stichwort *Accessible Design*, übersetzt mit *Zugänglichkeit* oder *Barrierefreiheit*. Hier nur ein paar Hinweise:

- Ein Blinder nimmt nur den Text wahr, und zwar zeilenweise. Grafiken und Farbe existieren für ihn nicht.
- Sehschwache erkennen kleine Schrift fester Größe oder manche Farbunterschiede nicht.
- Für jedes Bild (IMG) ist eine Textalternative (ALT) anzugeben, die bei nur schmückenden Bildern der leere String sein kann. Das Gleiche gilt für Audio-Dateien, die auch nicht für jedermann hörbar sind. Ebenso ist jeder Tabelle (TABLE) eine Zusammenfassung als Text (SUMMARY) beizufügen.
- Nur seit längerem bekannte Standard-Tags verwenden. Die Vorlese-Programme (Screen Reader) kommen mit Nicht-Standard-Tags und den neuesten Errungenschaften von HTML noch nicht klar.
- Vermeiden Sie Rahmen (frames), wenn sie nicht erforderlich sind, oder bieten Sie alternativ eine rahmenlose Variante Ihrer Seiten an.
- Die Vorlese-Programme sind meist auf eine bestimmte Sprache eingestellt, zum Beispiel Deutsch. Das Mischen von Sprachen irritiert den Sehbehinderten erheblich und sollte soweit möglich unterbleiben.

Zugängliche Webseiten sind auch für Nicht-Behinderte unproblematisch.

2.6.2 X Window System (X11)

2.6.2.1 Zweck

Das unter UNIX verbreitete **X Window System** (*nicht*: Windows) ist ein

- grafisches,
- hardware- und betriebssystem-unabhängiges,
- netzfähiges (verteiltes)

Fenstersystem, das am Massachusetts Institute of Technology (MIT) im Projekt Athena entwickelt wurde und frei verfügbar ist. Im Jahr 1988 wurde die Version 11 Release 2 veröffentlicht. Heute wird es vom X Consortium betreut. Weitere korrekte Bezeichnungen sind X Version 11, **X11** und X, gegenwärtig (2005) als sechstes Release X11R6. Eine freie Portierung auf Intel-Prozessoren heißt XFree86.

Netzfähig bedeutet, daß die Berechnungen (die Client-Prozesse) auf einer Maschine im Netz laufen können, während die Terminal-Ein- und -Ausgabe (der Server-Prozess) über eine andere Maschine im Netz erfolgen (**Client-Server-Modell**). Die gesamte Anwendung ist auf zwei Maschinen verteilt. Ein **Client** ist ein Prozess, der irgendwelche Dienste verlangt, ein **Server** ein Prozess, der Dienste leistet. Die Trennung einer Aufgabe in einen Client- und einen Server-Teil erhöht die Flexibilität und ermöglicht das Arbeiten über Netz. Man muß sich darüber klar sein, daß die Daten ohne zusätzliche Maßnahmen unverschlüsselt über das Netz gehen und abgehört werden können. X11 enthält nur minimale Sicherheitsvorkehrungen. Der Preis für die Flexibilität ist ein hoher Bedarf an Speicherkapazität und Prozessorzeit.

Die Leistungsfähigkeit von X11 in Verbindung mit Internet-Protokollen (NSF) zeigt sich am Beispiel des Manuskriptes zu diesem Buch. Ich arbeite auf einer HP-Workstation neben meinem Schreibtisch. Es versteht sich, daß der zugehörige Bildschirm zur Oberklasse zählt. In meinem Alter braucht man das. Die Dateien liegen auf einem Dateiserver mit reichlich Plattenkapazität unter Linux ein Stockwerk tiefer. Das Übersetzen der LaTeX-Dateien bis hin zu PostScript erfolgt auf einem weiteren Linux-PC mit viel Prozessorleistung und Arbeitsspeicher. Das Ergebnis schaue ich mir mit `xdvi(1)` an, das auf dem letztgenannten PC als X-Client läuft und meine HP-Workstation als X-Server nutzt. Ein ahnungsloser Zuschauer könnte meinen, alles liefe lokal auf der Workstation. Könnte es auch, aber der beschriebene Weg nutzt die Ressourcen unseres Netzes besser.

X11 stellt die Funktionen bereit, um grafische Benutzeroberflächen zu gestalten, legt aber die Art der Oberfläche nur in Grundzügen fest. Es ist ein Fundament, um Oberflächen darauf aufzubauen. Die Einzelheiten der Oberfläche sind Sache besonderer Funktionsbibliotheken wie **Motif** bzw. Sache bestimmter Programme, der Window-Manager, die nicht immer Bestandteil von X11 sind und teilweise auch Geld kosten. Der in X11 enthaltene Window-Manager ist der Tab Window Manager `twm(1)`, Hewlett-Packard fügt seinen Systemen den Motif Window Manager `mwm(1X)` und den VUE Window

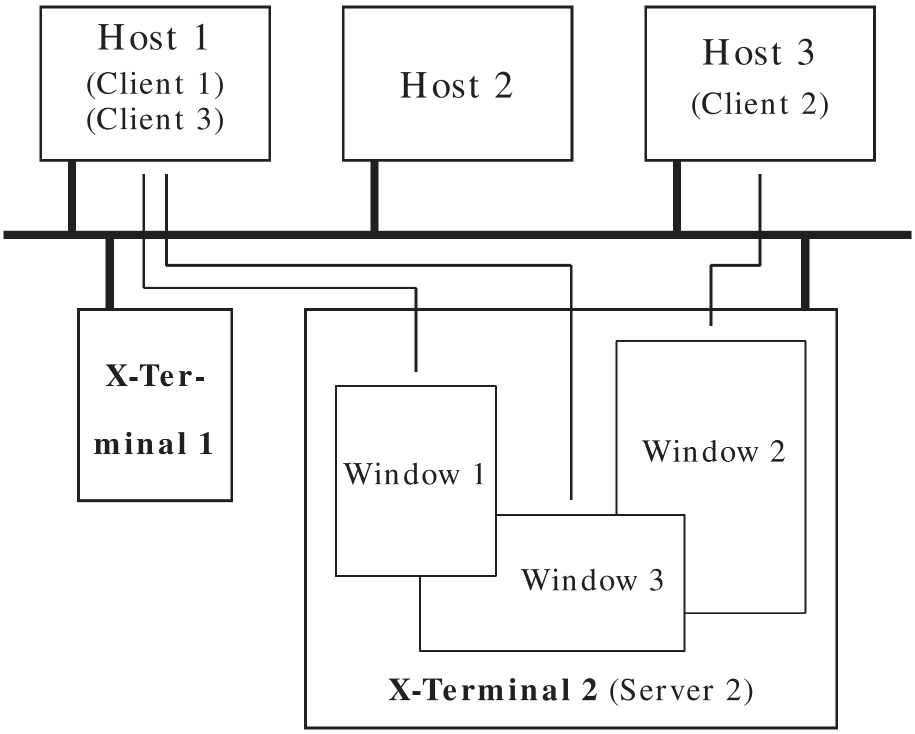

Abb. 2.7: X-Window-Computer und -Terminals, durch einen Ethernet-Bus verbunden

Manager `vuewm(1)` bei, unter Linux findet sich der Win95 Window Manager `fvwm95(1)`, dessen Fenster an Microsoft Windows 95 erinnern. Das K Desktop Environment (KDE) bringt den `kwin(1)` mit, GNOME Metacity.

Die **X-Clients** verwenden X11-Funktionen zur Ein- und Ausgabe, die in umfangreichen Funktionsbibliotheken wie `Xlib` und `Xtools` verfügbar sind. Es bleibt immer noch einiges an Programmierarbeit übrig, aber schließlich arbeitet man unter X11 mit Farben, Fenstern, Mäusen und Symbolen, was es früher zu Zeiten der einfarbigen Kommandozeile nicht gab. Inzwischen machen schon viele Anwendungsprogramme von den Möglichkeiten von X11 Gebrauch.

Der **X-Server** läuft als einziges Programm auf einem kleinen, spezialisierten Computer, dem X-Terminal, oder als eines unter vielen auf einem UNIX-Computer. Ein X-Server kann gleichzeitig mit mehreren X-Clients verkehren.

Mit X11 kann man auf dreierlei Weise zu tun bekommen:

- als Benutzer eines fertig eingerichteten X11-Systems (das gilt für wachsende Kreise von Linux/UNIX-Benutzern),
- als Systemverwalter, der X11 auf mehreren Netzknoten einrichtet,
- als Programmierer, der Programme schreibt, die unmittelbar im Programmcode von X11 Gebrauch machen, also nicht wie gewöhnliche Linux/UNIX-Programme in einer Terminal-Emulation (`xterm(1)`, `hpterm(1X)`) laufen.

Der Benutzer muss vor allem zwei Kommandos kennen. Auf der Maschine, vor der er sitzt (wo seine Sitzung läuft, der X-Server), gibt er mit

```
xhost abcd
```

der fernen Maschine namens `abcd` (wo seine Anwendung läuft, der X-Client) die Erlaubnis zum Zugriff. Das Kommando ohne Argument zeigt die augenblicklichen Einstellungen an. Die Antwort auf `xhost(1)` sollte beginnen mit `Access control enabled`, andernfalls wäre es angebracht, mit seinem Systemverwalter über die Sicherheit von X11 zu diskutieren, Hinweise gibt es im Netz. Auf der fernen Maschine `abcd` setzt man mit

```
export DISPLAY=efgh:0.0
```

die Umgebungsvariable `DISPLAY` auf den Namen `efgh` und die Fensternummer `0.0` des X-Servers. Erst dann kann ein Client-Programm, eine Anwendung über das Netz den X-Server als Terminal nutzen. Die Fensternummer besteht aus Displaynummer und Screennummer und hat nur auf Maschinen mit mehreren Terminals auch Werte größer null.

Abgesehen davon, daß die Daten unverschlüsselt über das Netz gehen und mitgelesen werden können, bestehen weitere Sicherheitslücken bei X11, die nur teilweise durch das Kommando `xhost(1)` geschlossen werden. Einen Schritt weiter geht die Xauthority mit dem Kommando `xauth(1)`, die zwischen Client und Server eine Art von Schlüssel (MIT Magic Cookie) austauscht. Mittels des Kommandos:

```
xauth list
```

kann man sich die Schlüssel in der Datei `$HOME/.Xauthority` ansehen, mittels der Pipe:

```
xauth extract - `hostname`:0.0 |
rexec clienthost xauth merge -
```

wird ein Schlüssel zur fernen Maschine `clienthost` geschickt, möglicherweise über einen noch ungeschützten Kanal. Statt `rexec(1)` kann es auch `rsh(1)` oder `remsh(1)` heißen. Bei Erfolg tauscht der Server nur noch Daten mit dem betreffenden Client aus. Das Kommando `xhost(1)` erübrigt sich dann, die DISPLAY-Variable ist beim Client nach wie vor zu setzen. Die Secure Shell `ssh(1)` mit dem Kommando `slogin(1)` erledigt sämtliche Schritte zum Aufbau einer sicheren Verbindung von Client und Server automatisch und ist damit der einfachste Weg. Sie gehört allerdings nicht zur Standardausrüstung von UNIX.

Die Ausgabe eines Bildschirms oder Fensters in eine Datei oder auf einen Drucker wird **Screen Dump** oder Bildschirmabzug genannt. Man braucht solche Dumps gelegentlich für Vorträge oder Veröffentlichungen, siehe Abbildung 2.8 auf Seite 128. Unter X11 schreibt eine Pipe aus den beiden Kommandos `xwd(1)` und `xpr(1)` einen Dump im PostScript-Format in eine Datei `xdump.ps`:

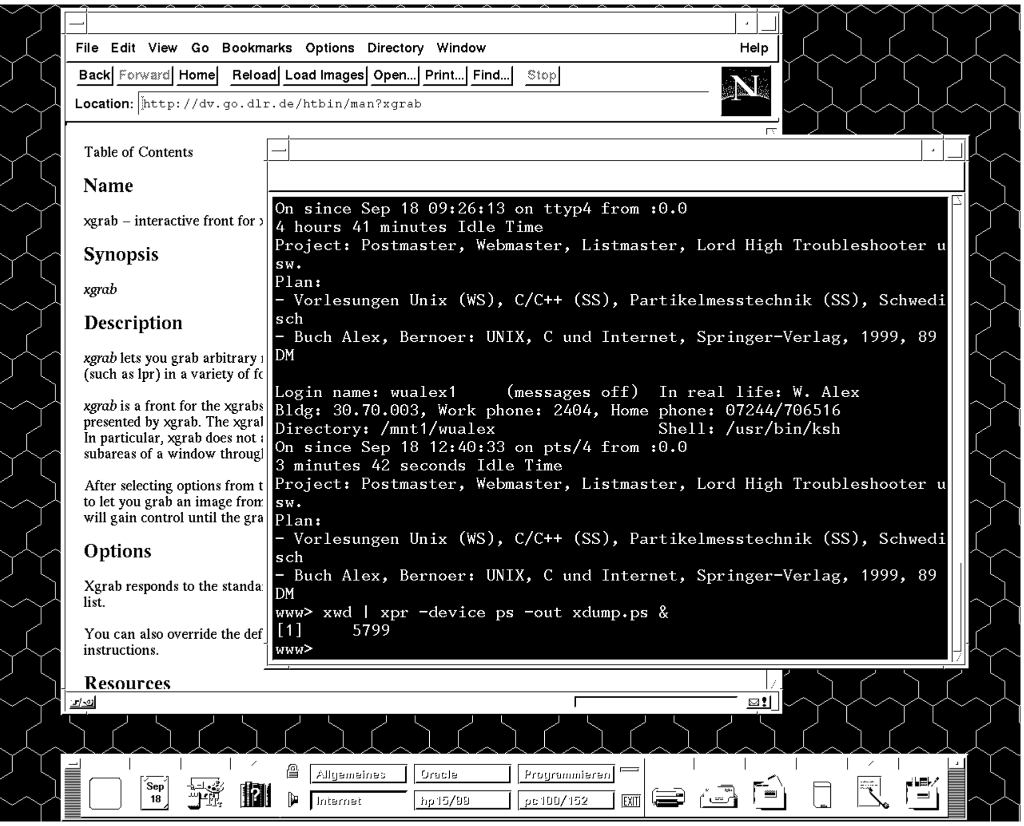

Abb. 2.8: Screen Dump eines X11-Bildschirms mittels `xwd` und `xpr`

```
xwd | xpr -device ps -out xdump.ps &
```

Nach dem Aufruf der Pipe im Hintergrund muß man noch das zu dumpende Fenster oder den Hintergrund anklicken, siehe die man-Seiten zu den beiden Kommandos. Das Kommando `xgrab(1)` leistet das Gleiche, ist jedoch nicht überall verfügbar. Auch Grafik-Pakete wie `gimp(1)` oder `xv(1)` enthalten Werkzeuge zum Dumpen des Bildschirms, ebenso das K-Desktop-Environment unter dem Menüpunkt `ksnapshot`.

Für den Programmierer stehen umfangreiche X11-Bibliotheken zur Verfügung, das heißt X11-Funktionen zur Verwendung in eigenen Programmen, so daß diese mit einem X-Server zusammenarbeiten. Die Xlib ist davon die unterste, auf der weitere aufbauen.

Wer tiefer in X11 eindringen möchte, beginnt am besten mit `man X`, geht dann zu `http://www.camb.opengroup.org/tech/desktop/x/` ins WWW und landet schließlich bei den ebenso zahl- wie umfangreichen Bänden des Verlages O'Reilly.

2.6.2.2 OSF/Motif

OSF/Motif von der Open Software Foundation ist ein Satz von Regeln zur Gestaltung einer grafischen Benutzeroberfläche für X11, eine Bibliothek mit Funktionen gemäß diesen Regeln sowie eine Sammlung daraus abgeleiteter Programme. Die Open Software Foundation OSF ist ein Zusammenschluß mehrerer Hersteller und Institute, die Software für UNIX-Anlagen herstellen. Motif ist heute in der kommerziellen UNIX-Welt die am weitesten verbreitete grafische Benutzeroberfläche und für viele Systeme verfügbar, leider nicht kostenlos. Aber es gibt freie Nachbauten. Das Common Desktop Environment (CDE), eine integrierte Benutzeroberfläche oder Arbeitsumgebung, baut auf Motif auf. Unter Linux stellen das K Desktop Environment (KDE) und das GNU Network Object Model Environment (GNOME) eine Alternative zu CDE dar.

Programme, die Motif benutzen, stellen sich dem Benutzer in einheitlicher Weise dar. Ihre Benutzung braucht man nur einmal zu lernen. Motif benötigt eine **Maus** oder ein anderes Zeigegerät (pointing device). Die Maustasten haben drei Funktionen:

- select (linker Knopf),
- menu (mittlerer Knopf bzw. bei einer Maus mit zwei Tasten beide gleichzeitig),
- custom (rechter Knopf).

Durch Verschieben der Maus auf einer Unterlage bewegt man eine Marke (Pointer, Cursor) auf dem Bildschirm. Die Marke nimmt je nach Umgebung verschiedene Formen an: Kreuz, Pfeil, Sanduhr, Motorradfahrer usw. **Zeigen** (to point) heißt, die Marke auf ein Bildschirmobjekt zu bewegen. Unter **Klicken** (to click) versteht man das kurze Betätigen einer Taste der ruhenden Maus. Zwei kurze Klicks unmittelbar nacheinander heißen Doppel-Klick (double-click). **Ziehen** (to drag) bedeutet Bewegen der Maus mit gedrückter

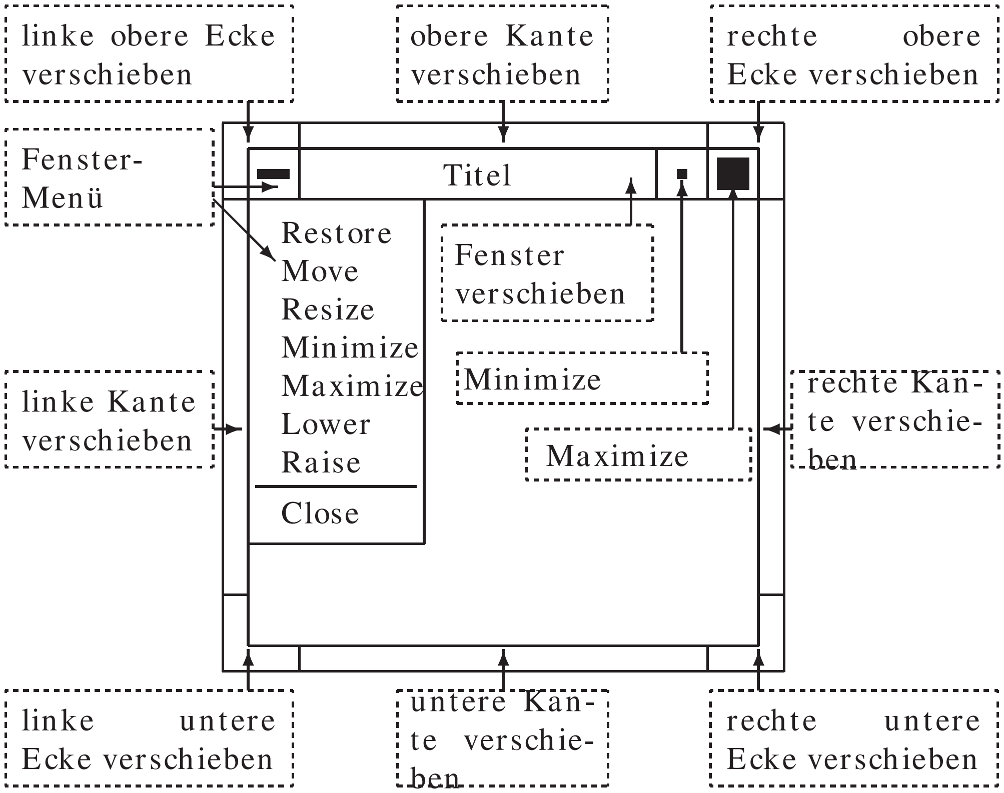

Abb. 2.9: OSF/Motif-Fenster

Taste. In einigen Systemen lassen sich die Mauseingaben durch Tastatureingaben ersetzen, aber das ist nur ein Behelf.

Falls das UNIX-System entsprechend konfiguriert ist, startet nach der Anmeldung automatisch **X11** und darin wiederum der **Motif Window Manager** `mwm(1)`. Unter UNIX sind das Prozesse. Der Motif Window Manager erzeugt standardmäßig zunächst einen Terminal-Emulator samt zugehörigem Fenster auf dem Bildschirm. Dieses Fenster kann man seinen Wünschen anpassen. Es besteht aus einer **Kopfleiste** (title bar), dem **Rahmen** (frame) und der Fenster- oder Arbeitsfläche. Die Kopfleiste enthält links ein kleines Feld mit einem Minuszeichen (menu button). Rechts finden wir ein Feld mit einem winzigen Quadrat (minimize button) und ein Feld mit einem größeren Quadrat (maximize button) (Abbildung 2.9 auf Seite 130).

Ehe ein Fenster bzw. der mit ihm verbundene Prozeß Eingaben annimmt, muß es durch Anklicken eines beliebigen Teils mit der Select-Maustaste **aktiviert** oder selektiert werden. Dabei ändert sich die Rahmenfarbe. Gibt man nun auf der Tastatur Zeichen ein, erscheinen sie im Fenster und gelangen zum Computer. Man sagt auch, das Fenster habe den **Fokus**, oder genauer, die Eingabe von der Tastatur sei auf das Fenster fokussiert. Es ist immer nur ein Fenster aktiv. Ein Fenster wird deaktiviert, wenn ein anderes Fenster aktiviert wird oder der Mauscursor das aktive Fenster verläßt.

Ein Fenster wird auf dem Bildschirm verschoben, indem man seine Kopfleiste mit der Select-Maustaste in die neue Position zieht. Nach Loslassen der Taste verharrt das Fenster an der neuen Stelle. Die Größe eines Fenster wird durch Ziehen einer Rahmenseite verändert. Zieht man eine Ecke, ändern sich die beiden angrenzenden Seiten gleichzeitig.

Gelegentlich möchte man ein Fenster vorübergehend beiseite legen, ohne es jedoch ganz zu löschen, weil mit ihm noch ein laufender Prozeß verbunden ist. In diesem Fall klickt man mit der Select-Maustaste den **Minimize-Button** an, und das Fenster verwandelt sich in ein Sinnbild, Symbol oder **Icon**. Das ist ein Rechteck von Briefmarkengröße am unteren Bildschirmrand. Der zugehörige Prozeß läuft weiter, nimmt aber keine Eingaben von der Tastatur mehr an. Icons lassen sich auf dem Bildschirm verschieben. Um aus dem Icon wieder ein Fenster zu machen, klickt man es doppelt mit der Select-Maustaste an.

Durch Anklicken des **Maximize-Buttons** bringt man ein Fenster auf volle Bildschirmgröße, so daß kein weiteres Fenster mehr zu sehen ist. Das empfiehlt sich für längere Arbeiten in einem Fenster. Auf die vorherige Fenstergröße zurück kommt man durch nochmaliges Anklicken des Maximize-Buttons.

Jetzt fehlt noch der **Menü-Button**. Klickt man ihn an, erscheint unterhalb der Kopfleiste ein Menü (Pull-down-Menü) mit einigen Funktionen zur Fenstergestaltung. Eine zur Zeit nicht verfügbare oder sinnlose Funktion erscheint grau.

2.6.3 Begriffe Oberflächen, X Window System

Folgende Begriffe sollten klarer geworden sein:

- CDE, KDE, GNOME
- Fenster
- Desktop
- Umgebung
- Kommandozeile
- Menü
- OSF Motif
- Widget
- X Window System (X11), X-Server, X-Client

Folgende Kommandos sollten beherrscht werden:

- startx
- xterm
- xclock &

2.6.4 Memo Oberflächen, X Window System

- Die Oberfläche mit den geringsten Ansprüchen an das System ist die Kommandozeile.
- Der erste Schritt in Richtung Benutzerfreundlichkeit sind Menus. Man kann allerdings nicht alle Kommandos in Menus verpacken.
- Der nächste Schritt sind zeichenorientierte Fenster, wie sie mit Hilfe der `curses(3)`-Funktionen geschrieben werden können. Dann kommen grafische Fenster.
- Das X Window System (X11) ist ein netzfähiges, hardwareunabhängiges, grafisches Fenstersystem. Die Netzfähigkeit unterscheidet es von anderen grafischen Fenstersystemen.
- Der X-Server sorgt für die Ein- und Ausgabe auf einem Terminal.
- X-Clients sind die Anwendungsprogramme.
- X-Server und X-Clients können auf verschiedenen Computern im Netz laufen, aber auch auf demselben.
- Das Aussehen und Verhalten (look and feel) wird von dem X Window Manager bestimmt. Es gibt verschiedene X Window Manager.
- Die Motif-Oberfläche (Motif-Widget-Bibliothek, Motif Window Manager) hat sich in der UNIX-Welt durchgesetzt.
- Auf der Motif-Oberfläche baut das Common Desktop Environment (CDE) auf, das zusätzliche Arbeitshilfen (Desktops) bietet.
- Unter Linux stellen das K Desktop Environment (KDE) und das GNU Network Object Model Environment (GNOME) eine Alternative zu CDE dar.
- Für Programmierer stehen umfangreiche X- und Motif-Bibliotheken zur Verfügung.

2.6.5 Übung Oberflächen, X Window System

Die folgende Übung setzt in ihrem letzten Teil voraus, daß Sie an einem X-Window-fähigen Terminal arbeiten, ein an einen seriellen Multiplexer angeschlossenes Terminal reicht nicht.

Melden Sie sich unter Ihrem Benutzernamen an. Der Verlauf der Sitzung hängt davon ab, welche Möglichkeiten Ihr UNIX-System bietet. Wir beginnen mit der Kommandozeilen-Eingabe:

```
who ?
help who
who -x
who -a
primes 0 100
factor 5040
```

Programme mit Menüs sind nicht standardmäßig in UNIX vorhanden. Wir geben daher das Shellskript `menu` ein und verändern es. Insbesondere ersetzen wir die Ausgabe der gewählten Ziffer durch den Aufruf eines Kommandos.

Um mit Fenstern und der `curses(3)`-Bibliothek arbeiten zu können, müssen wir in C programmieren. Hierzu lässt sich ein Beispiel aus dem Skriptum *Programmieren in C/C++* heranziehen.

Das Arbeiten mit der Benutzeroberfläche OSF/Motif setzt voraus, daß diese eingerichtet ist. Auf vernetzten UNIX-Workstations ist das oft der Fall. In der Regel startet Motif mit einer Terminalemulation, beispielsweise `xterm` oder `hpterm`. Geben Sie in diesem Fenster zunächst einige harmlose UNIX-Kommandos ein.

Verschieben Sie das Fenster, indem Sie mit der Maus den Cursor in die Titelleiste bringen und dann bei gedrückter linker Maustaste das Fenster bewegen (ziehen).

Verändern Sie die Größe des Fensters, indem Sie mit der Maus den Cursor auf einen Rand bringen und dann bei gedrückter linker Maustaste den Rand bewegen (ziehen).

Reduzieren Sie das Fenster, indem Sie mit der Maus den Cursor auf den Minimize-Button (rechts oben) bringen und dann die linke Maustaste drücken. Verschieben Sie das Icon. Stellen Sie das Fenster wieder her, indem Sie den Cursor auf das Icon bringen und zweimal die linke Maustaste drücken.

Bringen Sie das Fenster auf die maximale Größe, indem Sie den Cursor auf den Menü-Button (links oben) bringen und dann mit gedrückter linker Maustaste `maximize` wählen. Stellen Sie die ursprüngliche Größe durch erneute Anwahl von `maximize` (Menü oder Button) wieder her.

Erzeugen Sie ein zweites Fenster, indem Sie den Cursor aus dem ersten Fenster herausbewegen und mit dem linken Mausknopf eine Terminalemulation wählen. Bewegen Sie den Cursor abwechselnd in das erste und zweite Fenster, klicken Sie links und achten Sie auf die Farbe der Rahmen.

Tippen Sie im aktiven Fenster

```
xterm -bg red -fg green -fn cr.12x20 &
```

ein. Nach Erscheinen eines Rahmens nochmals RETURN drücken. Die Optionen bedeuten background, foreground und font. Warum muß das et-Zeichen eingegeben werden? Tippen Sie

```
xclock &
```

ein und veschieben Sie die Uhr in eine Ecke.

Schließen Sie ein Fenster, indem Sie den Cursor auf den Menü-Button bringen und mit gedrückter linker Maustaste `close` wählen. Verlassen Sie Motif mit der Kombination `control-shift-reset` (System-Manager fragen) und beenden Sie Ihre Sitzung mittels `exit` aus der Kommandozeile, wie gewohnt.

2.6.6 Fragen Oberflächen, X Window System

- Was ist eine Benutzeroberfläche?
- Was ist eine Kommandozeile?
- Was ist ein Menue?
- Was ist ein Fenster?
- Was ist ein Desktop?
- Was bedeutet *Multimedia* im Zusammenhang mit Benutzeroberflächen?
- Was ist das X Window System? Was unterscheidet es von anderen grafischen Fenstersystemen?
- Wo läuft der X-Server, wo der X-Client?
- Was ist OSF/Motif? Alternativen?
- Was braucht man, wenn man Programme schreibt, die unter dem X Window System laufen sollen?

2.7 Writer's Workbench

Unter der *Werkbank des Schreibers* (Writer's Workbench) werden Werkzeuge zur Textverarbeitung zusammengefasst. Linux/UNIX bietet eine Vielfalt davon. Man darf jedoch nicht vergessen, dass Linux/UNIX keine Büroumgebung, sondern ein Betriebssystem ist.

2.7.1 Zeichensätze und Fonts (oder die Umlaut-Frage)

2.7.1.1 Zeichensätze

Wenn es um Texte geht, muss man sich leider zuerst mit dem Problem der Zeichensätze (character set, data code, code de caractère) herumschlagen. Das hat nichts mit UNIX zu tun, sondern tritt unter allen Systemen auf.

Der Rechner kennt nur Bits. Die Bedeutung erhalten die Bits durch die Programme. Ob eine Bitfolge in der Menschenwelt eine Zahl, ein Zeichen oder einen Schnörkel darstellt, entscheidet die Software. Um mit Texten zu arbeiten, muss daher ein Zeichensatz vereinbart werden. Dieser besteht aus einer zunächst ungeordneten Menge von Zeichen (character), auch Répertoire oder **Zeichenvorrat** genannt, die nur besagt, welche Zeichen bekannt sind. Zu dem Zeichenvorrat gehören bei europäischen Sprachen:

- Kleine und große Buchstaben, auch mit Akzenten usw.,
- Ziffern,
- Satzzeichen,
- Symbole: aus der Mathematik, Euro-Symbol, Klammeraffe,
- Zwischenraum (Space), Tabulator (sogenannte Whitespaces),

- Steuerzeichen wie Seitenwechsel (Form Feed), Backspace.

In einem zweiten Schritt wird die Menge geordnet, jedem Zeichen wird eine **Position** (code position, code point) zugewiesen. Naheliegend ist eine mit null beginnende Numerierung. Die geordnete Menge ist der **Zeichensatz**. Bekannt ist die Zeichensatztabelle Nr. 850 aus der DOS-Welt, in der 256 Zeichen den hexadezimalen Zahlen von 00 bis FE zugeordnet werden. Wir wissen aber noch nicht, wie die Zeichen im Rechner und auf dem Bildschirm oder auf Papier dargestellt werden.

Die **Zeichencodierung** (character encoding) legt fest, wie eine Folge von Zeichen in eine Folge von Bytes umzuwandeln ist, und umgekehrt. Im einfachsten Fall wird die vorzeichenlose, ganzzahlige Positionsnummer als Code für ein Zeichen genommen.

Dann stellt sich die Frage, wie solche Zahlen im Rechner dargestellt werden. Der häufigste Fall ist eine Darstellung als duale Zahl mit sieben, acht oder mehr Bits. Es geht auch komplizierter, vor allem wenn internationale oder nicht-lateinische Zeichensätze zu codieren sind. Wir kommen so zur **Codetafel**. In der linken Spalte stehen die Zeichen, in der rechten die Bits (oder umgekehrt). Mit diesem Schritt ist die rechnerinterne Darstellung der Zeichen festgelegt, aber noch nicht deren Aussehen auf Schirm oder Papier, die Glyphen.

Zu Zeiten, als Bits noch knapp und teuer waren, haben die Yankees[30] eine Codetafel (Tabelle) aufgestellt, in der die ihnen bekannten Buchstaben, Ziffern und Satzzeichen zuzüglich einiger Steueranweisungen wie Zeilen- und Seitenvorschub mit sieben Bits dargestellt werden. Das war Sparsamkeit am falschen Platz. Mit sieben Bits Breite unterscheide ich $2^7 = 128$ Zeichen, numeriert von 0 bis 127. Diese Codetafel ist unter dem Namen *American Standard Code for Information Interchange* **ASCII** weit verbreitet. Genau heißt sie 7-bit-US-ASCII. Jeder Rechner kennt sie.

Die ersten 32 Zeichen der ASCII-Tafel dienen der Steuerung der Ausgabegeräte, es sind unsichtbare Zeichen. Ein Beispiel für Steuerzeichen ist das ASCII-Zeichen Nr. 12, Form Feed, das einen Drucker zum Einziehen eines Blattes Papier veranlasst. Auf der Tastatur werden sie entweder in Form ihrer Nummer oder mit gleichzeitig gedrückter control-Taste erzeugt. Die Ziffern 0 bis 9 tragen die Nummern 48 bis 57, die Großbuchstaben die Nummern 65 bis 90. Die Kleinbuchstaben haben um 32 höhere Nummern als die zugehörigen Großbuchstaben. Der Rest sind Satzzeichen. Im Anhang ist die ASCII-Tafel samt einigen weiteren Tafeln wiedergegeben.

Textausgabegeräte wie Bildschirme oder Drucker erhalten vom Rechner die ASCII-Nummer eines Zeichens und setzen diese mithilfe einer fest eingebauten Software in das entsprechende Zeichen um. So wird beispielsweise die ASCII-Nr. 100 in den Buchstaben d umgesetzt. Die Ausgabe der Zahl 100 erfordert das Abschicken der ASCII-Nr. 49, 48, 48.

[30]Yankee im weiteren, außerhalb der USA gebräuchlichen Sinne als Einwohner der USA. Die Yankees im weiteren Sinne verstehen unter Yankee nur die Bewohner des Nordostens der USA.

Die US-ASCII-Tafel enthält nicht die deutschen **Umlaute** und andere europäische Absonderlichkeiten. Es gibt einen Ausweg aus dieser Klemme, leider sogar mehrere. Bleibt man bei den sieben Bits, muss man einige nicht unbedingt benötigte US-ASCII-Zeichen durch nationale Sonderzeichen ersetzen. Für deutsche Zeichen ist eine Ersetzung gemäß Anhang B.2 *German ASCII* auf Seite 298 üblich. Für Frankreich oder Schweden lautet die Ersetzung anders. Diese Ersatztafel liegt nicht im Rechner, sondern im Ausgabegerät, das die Umsetzung der ASCII-Nummern in Zeichen vornimmt. Deshalb kann ein entsprechend ausgestatteter Bildschirm oder Drucker dieselbe Textdatei einmal mit amerikanischen ASCII-Zeichen ausgeben, ein andermal mit deutschen ASCII-Zeichen. Werden bei Ein- und Ausgabe unterschiedliche Zeichensätze verwendet, gibt es Zeichensalat. Andersherum gesagt: Wenn ich einen Text ausgebe, muss ich die Codetafel der Eingabe kennen.

Spendiert man ein Bit mehr, so lassen sich $2^8 = 256$ Zeichen darstellen. Das ist der bessere Weg. Hewlett-Packard hat die nationalen Sonderzeichen den Nummern 128 bis 255 zugeordnet und so den Zeichensatz **ROMAN8** geschaffen, dessen untere Hälfte mit dem US-ASCII-Zeichensatz identisch ist. Das hat den Vorzug, dass reine US-ASCII-Texte genau so verarbeitet werden wie ROMAN8-Texte. Leider hat sich diese Codetafel nicht allgemein durchgesetzt.

Die Firma IBM hat schon früh bei größeren Anlagen den *Extended Binary Coded Decimal Interchange Code* **EBCDIC** mit acht Bits verwendet, der aber nirgends mit ASCII übereinstimmt. Hätte sich diese Codetafel statt ASCII durchgesetzt, wäre uns Europäern einige Mühe erspart geblieben.

Die internationale Normen-Organisation ISO hat mehrere 8-bit-Zeichensätze festgelegt, von denen einer unter dem Namen **Latin-1** nach ISO 8859-1 Verbreitung gewonnen hat, vor allem in weltweiten Netzdiensten. Seine untere Hälfte ist wieder mit US-ASCII identisch, die obere enthält die Sonderzeichen west- und mitteleuropäischer Sprachen. Polnische und tschechische Sonderzeichen sind in **Latin-2** nach ISO 8859-2 enthalten, siehe Anhang *Latin-1* und *Latin-2* ab Seite 300. Die Zeichensätze Latin-1 bis 4 sind auch im Standard ECMA-94 beschrieben (ECMA = European Computer Manufacturers Association). Kyrillische Zeichen sind in ISO 8859-5, griechische in ISO 8859-7 festgelegt (nicht als Latin-* bezeichnet; Latin-5 ist Türkisch nach ISO 8859-9).

Die Latin-Zeichensätze enthalten außer dem gewohnten Zwischenraumzeichen (space) ein in Textverarbeitungen oft benötigtes Zeichen für einen Zwischenraum, bei dem kein Zeilenumbruch erfolgen darf (Latin-1 Nr. 160, no-break space). In LaTeX wird hierfür die Tilde verwendet, in HTML die Entity ` `. Dieses Zeichen kommt beispielsweise zwischen Zahl und Maßeinheit oder zwischen den Initialen eines Namens vor.

Bei ihren PCs schließlich wollte IBM außer nationalen Sonderzeichen auch einige Halbgrafikzeichen wie Mondgesichter, Herzchen, Noten und Linien unterbringen und schuf einen weiteren Zeichensatz **IBM-PC**, der in seinem Kern mit ASCII übereinstimmt, ansonsten aber weder mit EBCDIC noch mit ROMAN8.

Auch wenn die Ausgabegeräte 8-bit-Zeichensätze kennen, ist noch nicht sicher, dass man die Sonderzeichen benutzen darf. Die Programme müssen ebenfalls mitspielen. Der hergebrachte `vi(1)`-Editor, die `curses(3)`-Bibliothek für Bildschirmfunktionen und einige Email-Programme verarbeiten nur 7-bit-Zeichen. Erst jüngere Versionen von UNIX mit **Native Language Support** unterstützen 8-bit-Zeichensätze voll. Textverarbeitende Software, die 8-bit-Zeichensätze verträgt, wird als **8-bit-clean** bezeichnet. Bei Textübertragungen zwischen Rechnern (Email) ist Misstrauen angebracht. Die Konsequenz heißt in kritischen Fällen Beschränkung auf 7-bit-US-ASCII, das funktioniert überall.

Was macht man, wenn es zu viele Standards gibt? Man erfindet einen neuen, der eine Obermenge der bisherigen ist. Das *Unicode Consortium* (`www.unicode.org/`) arbeitet seit 1991 daran, einen Zeichensatz (Zeichen, Namen und Positionsnummer) zu definieren, der die Zeichen aller historischen, gegenwärtigen und zukünftigen Schriftsprachen der Erde berücksichtigt. Dieser Zeichensatz namens **Unicode** ist in ISO/IEC 10 646 genormt. Er reicht für mehr als eine Million Zeichen.

Darüber hinaus legt der Unicode die Darstellung (Codierung) der Zeichen im Verkehr der Rechner untereinander fest. Die Darstellung wird als **Unicode Transformation Format** (UTF) bezeichnet; es gibt drei davon:

- UTF-32 (UCS-4) verwendet eine feste Anzahl von 32 Bits (4 Bytes) und codiert damit bis zu $2^32 = 4294967296$ Zeichen (Positionen). Der Bedarf an Speicherplatz ist maximal. Kommt bei einigen UNIXen vor.
- UTF-8 verwendet 1 bis 4 Bytes nach Bedarf. Der Bedarf an Speicherplatz ist minimal, dafür muss etwas mehr gerechnet werden. Im Web verbreitet. Siehe auch RFC 2279.
- UTF-16 (UCS-2) ist ein Kompromiss zwischen UTF-8 und UTF-32, der für die häufigeren Zeichen 2 Bytes (65 536 Positionen) und für die selteneren ein Paar von UTF-16-Codes (4 Bytes) verwendet. Von Java und MS Windows benutzt.

Die ASCII-Zeichen bleiben an den gewohnten Positionen, gleichzeitig kann bei allen UTFs jedoch der vollständige Unicode-Zeichensatz benutzt werden.

UTF-7 nach RFC 1642 ist ein Zugeständnis an die Email-Standards, die verlangen, dass Email nur mit 7 Bits codierte Zeichen enthalten darf. Die US-ASCII-Zeichen bleiben unverändert, Zeichen auf höheren Positionen werden durch ein Plus- und ein Minuszeichen eingerahmt nach UTF-16 und Base64 codiert.

Zur Umsetzung von Zeichen gibt es mehrere UNIX-Werkzeuge wie `tr(1)` und `sed(1)`. Das erstere ist auch ein Nothelfer, wenn man ein Zeichen in einen Text einbauen möchte, das man nicht auf der Tastatur findet. Will man beispielsweise den Namen *Citroën* richtig schreiben und sieht keine Möglichkeit, das e mit dem Trema per Tastatur zu erzeugen, dann schreibt man *CitroXn* und schickt den Text durch:

```
tr '\130' '\315' < text > text.neu
```

Die Zahlen sind die oktalen Positionen, hier im Zeichensatz HP-Roman8. Ein C-Programm für diesen Zweck ist andererseits einfach:

```
/* Programm zum Umwandeln von bestimmten Zeichen eines
   Zeichensatzes in Zeichen eines anderen Zeichensatzes,
   hier ROMAN8 nach LaTeX. Als Filter (Pipe) einfuegen.
   Zeichen werden durch ihre dezimale Nr. dargestellt. */

#include <stdio.h>

int main()
{
    int c;

    while ((c = getchar()) != EOF)
        switch (c) {
            case  189:
                putchar(92);
                putchar(83);
                break;
            case  204:
                putchar(34);
                putchar(97);
                break;
            .
            .
            .
            case 219:
                putchar(34);
                putchar(85);
                break;
            case 222:
                putchar(92);
                putchar(51);
                break;
            default:
                putchar(c);
        }
}
```

Quelle 2.20 : C-Programm zur Zeichenumwandlung

Aus dem GNU-Projekt stammt ein Filter namens `recode(1)`, dass etwa hundert Codetafeln oder Zeichensätze ineinander umrechnet:

```
recode --help
recode -l
recode ascii-bs:EBCDIC-IBM textfile
```

Man beachte jedoch, dass beispielsweise ein HTML-Text, der mit ASCII-Ersatzdarstellungen für die Umlaute (`ä` für a-Umlaut) geschrieben ist, bei Umwandlung nach ASCII unverändert bleibt. Es werden Zeichensätze umgewandelt, mehr nicht. Auch werden LaTeX-Formatanweisungen nicht in HTML-Formatanweisungen übersetzt, dafür gibt es andere Werkzeuge wie `latex2html`. Die Ursprungsdatei wird überschrieben, daher sicherheitshalber mit einer Kopie arbeiten. Nicht jede Umsetzung ist reversibel.

2.7.1.2 Fonts, Orientierung

Der Zeichensatz sagt, welche Zeichen bekannt sind, nicht wie sie aussehen. Ein Font legt das Aussehen der Zeichen, die Glyphen, fest. Ursprünglich war ein Font der Inhalt eines Setzkastens. Verwirrung entsteht dadurch, dass einige einfache Font-Formate die Gestaltinformationen nicht verschiedenen Zeichen, sondern verschiedenen Zahlen (Positionen) zuordnen und so der Font darüber befindet, welches Zeichen für welche Zahl ausgegeben wird. Die einzige saubere Lösung ist die Trennung von Zeichensatz und Font. Die Vielzahl der Fonts hat technische und künstlerische Gründe.

Bei einem anspruchsvollen Font werden nicht nur die einzelnen Zeichen dargestellt, sondern auch bestimmte Zeichenpaare (Ligaturen). JOHANNES GUTENBERG verwendete 190 Typen, und da war noch kein Klammeraffe und kein Euro dabei. Der Buchstabe *f* ist besonders kritisch. Erkennen Sie den Unterschied zwischen *hoffen* und *hoffähig*, *Kaufleute* und *Kaufläche* oder *Mitte* und *mitteilen*? Ligaturen sind ein einfacher Test für die Güte eines Textprogramms. Werden sie beachtet, kann man annehmen, dass sich seine Schöpfer Gedanken gemacht haben. LaTeX-Fonts kennen Ligaturen.

Unter einer Schrift, Schriftfamilie oder **Schriftart** (typeface) wie Times Roman, New Century Schoolbook, Garamond, Bodoni, Helvetica, Futura, Univers, Schwabacher, Courier, OCR (optical character recognition) oder Schreibschriften versteht man einen stilistisch einheitlichen Satz von Fonts in verschiedenen Ausführungen. Die Schriftart muss zum Charakter und Zweck des Schriftstücks und zur Wiedergabetechnik passen. Die Times Roman ist beispielsweise ziemlich kompakt sowie leicht und schnell zu lesen (sie stammt aus der Zeitungswelt), während die New Century Schoolbook 10 % mehr Platz benötigt, dafür aber deutlicher lesbar ist, was für Leseanfänger und Sehschwache eine Rolle spielt. Die klassizistische Bodoni wirkt etwas gehoben und ist die Lieblingsschrift von IBM. In einer Tageszeitung wäre sie fehl am Platze. Serifenlose Schriften wie die Helvetica eignen sich für Plakate, Overhead-Folien, Beschriftungen von Geräten und kurze Texte. Diese Schriften liegen in verschiedenen **Schriftschnitten** (treatment) vor: mager, fett, breit, schmal, kursiv, dazu in verschiedenen Größen oder **Schriftgraden** (point size). Die **Schriftweite**, der Zeichenabstand (pitch), ist entweder fest wie bei einfachen Schreibmaschinen, beispielsweise 10 oder 12 Zeichen pro Zoll, oder von der Zeichenbreite abhängig wie bei den **Proportionalschriften**. Diese sind besser lesbar und sparen Platz, erfordern aber in Tabellen Aufwand. Ein vollständiger Satz von Buchstaben, Ziffern, Satz- und Sonderzeichen einer Schrift, eines Schnittes, eines Grades und gegebenenfalls einer Schriftweite wird heute **Font** genannt. Die in diesem Text verwendeten Fonts heißen Times Roman oder New Century Schoolbook als Standardfont, `Courier`, Sans Serif, *Times Roman Italic*, KAPITÄLCHEN, im Manuskript in 12 pt Größe. Daneben habe ich Sonderausgaben von Teilen des Manuskripts in der New Century Schoolbook in 14 Punkten Größe hergestellt. Noch größere Schrift wäre möglich, aber dann passen nur noch wenige Wörter in die Zeile.

Im wesentlichen gibt es zwei Kategorien von Font-Formaten: Bitmap-

Fonts und Vektor-Fonts. **Bitmap-Fonts** speichern die Gestalt eines Zeichens in einer Punktematrix. Der Vorteil besteht in der einfacheren und schnelleren Verarbeitung. Darüber hinaus existieren auf vielen Systemen mehrere Bitmap-Fonts derselben Schrift, optimiert für die am häufigsten benötigten Schriftgrößen. Nachteilig ist die unbefriedigende Skalierbarkeit (Vergrößerung oder Verkleinerung). Das Problem ist das gleiche wie bei Grafiken.

Bessere Systeme verwenden daher **Vektor-Fonts**, die die Gestalt der Zeichen durch eine mathematische Beschreibung ihrer Umrisse festhalten. Vektor-Fonts lassen sich daher problemlos skalieren. Bei starken Maßstabsänderungen muss jedoch auch die Gestalt etwas verändert werden. Gute Vektor-Fonts speichern deshalb zusätzliche, beim Skalieren zu beachtende Informationen (hints) zu jedem Zeichen.

Die beiden wichtigsten Vektor-Font-Formate sind True Type (TT), hauptsächlich auf Macintoshs und unter Microsoft Windows, und das von Adobe stammende PostScript-Type-1-Format (PS1), das auch vom X Window System (X11) dargestellt werden kann und daher unter UNIX verbreitet ist. True Type kam um 1990 heraus und war die Antwort von Apple und Microsoft auf Adobe. Im Jahr 1996 rauften sich Adobe und Microsoft zusammen und schufen das Open Type Format als eine einheitliche Verpackung von PostScript- und Truetype-Fonts.

Das X Window System (X11) hat zunächst nichts mit der Druckausgabe zu tun. Die Tatsache, dass ein Font unter X11 verfügbar ist, bedeutet noch nicht, dass er auch gedruckt werden kann. Einen gemeinsamen Nenner von X11 und der Druckerwelt stellt das Type-1-Format dar: Fonts dieses Formates können sowohl von X11 auf dem Bildschirm dargestellt als auch als Softfonts (in Dateien gespeicherte Fonts) in PostScript-Drucker geladen werden. Für den Privatanwender, der sich keinen PostScript-Drucker leisten kann, bietet sich der Weg an, den freien PostScript-Interpreter *Ghostscript* als Druckerfilter zu verwenden. Er wandelt PostScript-Daten in verschiedene Druckersteuersprachen (PCL) um.

Da die Papierformate länglich sind, spielt die **Orientierung** (orientation) eine Rolle. Das Hochformat wird englisch mit portrait, das Querformat mit landscape bezeichnet[31]. Ferner trägt der **Zeilenabstand** oder Vorschub (line spacing) wesentlich zur Lesbarkeit bei. Weitere Gesichtspunkte zur Schrift und zur Gestaltung von Schriftstücken findet man in der im Anhang angegebenen Literatur und im Netz, zum Beispiel in dem FAQ der Newsgruppe `comp.fonts`, auf Papier 260 Seiten, zusammengestellt von NORMAN WALSH. Trinken Sie einen auf sein Wohl und denken Sie darüber nach, wieviel freiwillige und unentgeltliche Arbeit in den FAQs steckt.

Die vorstehenden Zeilen waren vielleicht etwas viel zu einem so einfachen Thema wie der Wiedergabe von Texten, aber es gibt nun einmal auf der Welt mehr als die sechsundzwanzig Zeichen des lateinischen Alphabets und mehr als ein Textprogramm. Auf längere Sicht kommt man nicht darum herum,

[31]Woraus man schließt, dass Engländer ein Flachland bewohnende Langschädler sind, während alpine Querköpfe die Bezeichnungen vermutlich andersherum gewählt hätten.

sich die Zusammenhänge klar zu machen. Dann versteht man, warum in der Textverarbeitung so viel schiefgeht, von der künstlerischen Seite ganz abgesehen.

2.7.2 Reguläre Ausdrücke

Reguläre Ausdrücke (regular expression, expression régulière, RE) sind Zeichenmuster, die nach bestimmten Regeln gebildet und ausgewertet werden[32]. Eine Zeichenfolge (String) kann darauf hin untersucht werden, ob sie mit einem gegebenen regulären Ausdruck übereinstimmt oder nicht. Einige Textwerkzeuge wie Editoren, `grep(1)`, `lex(1)`, `awk(1)`, `sed(1)` und `perl(1)` machen von regulären Ausdrücken Gebrauch, leider in nicht völlig übereinstimmender Weise. Die Jokerzeichen in Dateinamen und die Metazeichen der Shells haben *nichts* mit regulären Ausdrücken zu tun. Näheres findet man im Referenz-Handbuch beim Editor `ed(1)` und in dem Buch von ALFRED V. AHO und anderen über `awk(1)`. Hier einige einfache Regeln und Beispiele:

- Ein Zeichen mit Ausnahme der Sonderzeichen (Metazeichen) trifft genau auf sich selbst zu (klingt so selbstverständlich wie $a = a$, muss aber gesagt sein),
- ein Backslash gefolgt von einem Sonderzeichen trifft genau auf das Sonderzeichen zu (der Backslash quotet das Sonderzeichen),
- Punkt, Stern, linke eckige Klammer und Backslash sind Sonderzeichen, sofern sie nicht in einem Paar eckiger Klammern stehen,
- der Circumflex ist ein Sonderzeichen am Beginn eines regulären Ausdrucks oder unmittelbar nach der linken Klammer eines Paares eckiger Klammern,
- das Dollarzeichen ist ein Sonderzeichen am Ende eines regulären Ausdrucks,
- ein Punkt trifft auf ein beliebiges Zeichen außer dem Zeilenwechsel zu,
- eine Zeichenmenge innerhalb eines Paares eckiger Klammern trifft auf ein Zeichen aus dieser Menge zu,
- ist jedoch das erste Zeichen in dieser Menge der Circumflex, so trifft der reguläre Ausdruck auf ein Zeichen zu, das weder der Zeilenwechsel noch ein Zeichen aus dieser Menge ist,
- ein Bindestrich in dieser Menge kennzeichnet einen Zeichenbereich, `[0-9]` bedeutet dasselbe wie `[0123456789]`,
- ein regulärer Ausdruck aus einem Zeichen gefolgt von einem Fragezeichen bedeutet ein null- oder einmaliges Vorkommen dieses Zeichens,
- ein regulärer Ausdruck aus einem Zeichen gefolgt von einem Pluszeichen bedeutet ein ein- oder mehrmaliges Vorkommen dieses Zeichens,

[32] Eine genaue Definition findet sich in Werken zum Übersetzerbau.

- ein regulärer Ausdruck aus einem Zeichen gefolgt von einem Stern bedeutet ein beliebig häufiges Vorkommen dieses Zeichens, nullmaliges Vorkommen eingeschlossen (erinnert an Jokerzeichen in Dateinamen, aber dort kann der Stern auch ohne ein anderes Zeichen davor auftreten),
- ist `r` ein regulärer Ausdruck, dann bedeutet `(r)*` ein beliebig häufiges Vorkommen dieses Ausdrucks, entsprechend auch für Plus- oder Fragezeichen,
- eine Verkettung regulärer Ausdrücke trifft zu auf eine Verkettung von Strings, auf die die einzelnen regulären Ausdrücke zutreffen.

Die Regeln gehen weiter. Am besten übt man erst einmal mit einfachen regulären Ausdrücken. Nehmen Sie irgendeinen Text und lassen Sie `grep(1)` mit verschiedenen regulären Ausdrücken darauf los:

```
grep 'aber' textfile
grep 'ab.a' textfile
grep 'bb.[aeiou]' textfile
grep '^[0-9]+$' textfile
grep '\\[a-z][a-z]*{..*}' textfile
grep 'M[ae][iy]e?r' textfile
```

Die Single Quotes um die Ausdrücke sind eine Vorsichtsmaßnahme, die verhindern soll, dass sich die Shell die Ausdrücke zu Gemüte führt. `grep(1)` gibt die Zeilen aus, in denen sich wenigstens ein String befindet, auf den der reguläre Ausdruck passt. Im ersten Beispiel sind das alle Zeilen, die den String `aber` enthalten wie `aber`, `labern`, `Schabernack`, `aberkennen`, im zweiten trifft unter anderem `abwarten` zu, im dritten `Abbruch`. Die vierte Form ermittelt alle Zeilen, die nur Ziffern enthalten: Am Anfang der Zeile (Circumflex) ein Zeichen aus der Menge 0 bis 9, dann eine beliebige Wiederholung von Ziffern bis zum Ende (Dollar) der Zeile. Man sagt, dass Circumflex oder Dollarzeichen das Muster am Zeilenanfang oder -ende verankern. Das fünfte Beispiel liefert die Zeilen mit LaTeX-Kommandos wie `\index{}`, `\begin{}`, `\end{}` zurück. Der fünfte Ausdruck ist folgendermaßen zu verstehen:

- ein Backslash,
- genau ein Kleinbuchstabe,
- eine beliebige Anzahl von Kleinbuchstaben,
- eine linke geschweifte Klammer,
- genau ein beliebiges Zeichen,
- eine beliebige Anzahl beliebiger Zeichen,
- eine rechte geschweifte Klammer.

In der sechsten Zeile wird nach dem Namen *Meier* mit all seinen Varianten geforscht. Wie lautet ein regulärer Ausdruck, der auf die Namen aller `.exe`-Programme aus der DOS-Welt zutrifft? Ganz einfach:

```
\.exe$
```

Der Punkt muss mittels Backslash seiner besonderen Bedeutung beraubt (gequotet) werden, dann folgen drei harmlose Buchstaben. Das Dollarzeichen besagt, dass die vorgenannte Zeichenfolge am Ende eines Strings (Dateinamen) vorkommen soll. Wollen wir auch noch groß geschriebene Dateinamen erwischen, geht das mit einer oder-Verknüpfung:

```
\.exe$|\.EXE$
```

Wir wollen nun einen regulären Ausdruck zusammenstellen, der auf alle gültigen Internet-Email-Anschriften zutrifft. Dazu schauen wir uns einige Anschriften an:

```
wulf.alex@mvm.uni-karlsruhe.de
wualex1@mvmc64.ciw.uni-karlsruhe.de
ig03@rz.uni-karlsruhe.de
012345678-0001@t-online.de
Dr_Rolf.Muus@DEGUSSA.de
```

Links steht immer ein Benutzername, dessen Form vom jeweiligen Betriebssystem (Eintrag in `/etc/passwd`) bestimmt wird, dann folgen das `@`-Zeichen (Klammeraffe) und ein Maschinen- oder Domänenname, dessen Teile durch Punkte voneinander getrennt sind. Im einzelnen:

- Anfangs ein Zeichen aus der Menge der Ziffern oder kleinen oder großen Buchstaben,
- dann eine beliebige Anzahl einschließlich null von Zeichen aus der Menge der Ziffern, der kleinen oder großen Buchstaben und der Zeichen `_-.`,
- genau ein Klammeraffe als Trennzeichen,
- im Maschinen- oder Domänennamen mindestens eine Ziffer oder ein Buchstabe,
- dann eine beliebige Anzahl von Ziffern, Buchstaben oder Strichen,
- mindestens ein Punkt zur Trennung von Domäne und Top-Level-Domäne,
- nochmals mindestens ein Buchstabe zur Kennzeichnung der Top-Level-Domäne.

Daraus ergibt sich folgender regulärer Ausdruck (einzeilig):

```
^[0-9a-zA-Z][0-9a-zA-Z_-.]*@[0-9a-zA-Z][0-9a-zA-Z_-.]*
                                        \.[a-zA-Z][a-zA-Z]*
```

Das sieht kompliziert aus, ist aber trotzdem der einfachste Weg zur Beschreibung solcher Gebilde. Man denke daran, dass die UNIX-Kommandos leicht unterschiedliche Vorstellungen von regulären Ausdrücken haben. Auf meiner Mühle beispielsweise unterscheiden sich `grep(1)` und `egrep(1)`. Außerdem ist obige Form einer Email-Anschrift nicht gegen die RFCs abgeprüft und daher vermutlich zu eng. Eine Anwendung für den regulären Ausdruck könnte ein Programm sein, das Email-Anschriften verarbeitet und sicherstellen will, dass die ihm übergebenen Strings wenigstens ihrer Form nach gültig sind. Robuste Programme überprüfen Eingaben oder Argumente, ehe sie sich weiter damit abgeben.

2.7.3 Editoren (ed, ex, vi, elvis, vim)

Ein **Editor**[33] ist ein Programm zum Eingeben und Ändern von Textdateien, nach dem Kommando-Interpreter, der Shell, das am häufigsten benutzte Programm. Eine **Textdatei** enthält nur druck- und sichtbare Zeichen einschließlich Zwischenraum (space), Tabulator und Zeilenwechsel (CR und/oder LF), jedoch niemals darüber hinausgehende *versteckte*, unsichtbare Informationen. Alle Editoren stehen vor der Aufgabe, dass sowohl der Text wie auch die Editierkommandos eingegeben und voneinander unterschieden werden müssen. Folgende Lösungen sind denkbar:

- Getrennte Tastaturen für Text und Kommandos. Das kommt so selten vor, dass kein Rechner Anschlüsse für zwei Tastaturen hat (zwei Bildschirme sind möglich).
- Eine Tastatur mit besonderen Tasten für Editorkommandos (insert character, delete character usw.). Das ist gebräuchlich, aber es gibt weit mehr Kommandos (um die 100) als Spezialtasten. Außerdem sind die Spezialtasten von Hersteller zu Hersteller unterschiedlich, und UNIX bemüht sich, hardwareunabhängig zu sein.
- Alle Kommandos beginnen mit einem besonderen Zeichen oder einer besonderen Zeichenkombination, die in normalem Text nicht vorkommt. Diese Lösung verwendet der Editor `emacs(1)`.
- Der Editor befindet sich entweder im Eingabemodus oder im Kommandomodus. Im Eingabemodus werden Eingaben als Text interpretiert und in den Speicher geschrieben. Im Kommandomodus werden Eingaben als Kommandos aufgefasst und ausgeführt. Diesen Weg geht der Editor `vi(1)`.

Aus Editoren kommt man nur schwer wieder hinaus, wenn man nicht das Zauberwort kennt. Unzählige Benutzer wären schon in den Labyrinthen der Editoren verschmachtet, wenn ihnen nicht eine kundige Seele geholfen hätte. Deshalb hier vorab die Zauberworte:

- Falls Ihr Terminal auf nichts mehr reagiert, ist entweder auf der Rückseite ein Stecker locker, oder Sie haben es unwissentlich umkonfiguriert. Dann müssen Sie eine Reset-Taste drücken, bei unseren HP-Terminals die Kombination `control-shift-reset`.
- Aus dem `vi(1)`-Editor kommen Sie immer hinaus, indem Sie nacheinander die fünf Tasten `escape : q ! return` drücken.

[33]Zur deutlichen Unterscheidung von HTML-, Grafik- oder Sound-Editoren auch Text-Editor genannt.

- Den `emacs(1)`-Editor verlässt man mittels Drücken der beiden Tastenkombinationen `control-x control-c` nacheinander.
- Den `joe(1)`-Editor beendet man mit der Tastenkombination `control-k` und dann `x`.
- Den `pico`-Editor bricht man mit der Tastenkombination `control-x` ab.
- Das Anklicken von `exit` im Datei-Menue des `nedit(1)` lässt sich ersetzen durch die Tastenkombination `control-q`.

Achtung: Mit diesen Kommandos wird der Editor verlassen, nicht aber der bearbeitete Text in den Massenspeicher zurückgeschrieben, etwaige Änderungen gehen verloren. Falls das alles nicht wirkt, ist es geboten, um Hilfe zu rufen.

Das einfachste Kommando zur Eingabe von Text ist `cat(1)`. Mittels

```
cat > textfile
```

schreibt man von der Tastatur in die Datei `textfile`. Die Eingabe wird mit dem EOF-Zeichen `control-d` abgeschlossen. Die Fähigkeiten von `cat(1)` sind allerdings so bescheiden, dass es nicht die Bezeichnung Editor verdient.

Einfache Editoren bearbeiten immer nur eine Zeile eines Textes und werden zeilenweise weitergeschaltet. Auf dem Bildschirm sehen Sie zwar dank des Bildschirmspeichers mehrere Zeilen, aber nur in einer – der jeweils aktuellen – können Sie editieren. Diese Editoren stammen aus der Zeit, als man noch Fernschreibmaschinen als Terminals verwendete. Daher beschränken sie den Dialog auf das Allernötigste. **Zeilen-Editoren** wie DOS `edlin` oder UNIX `ed(1)` werden heute nur noch im Notfall benutzt. Der `ed(1)` ist robust und arbeitet auch unter ungünstigen Verhältnissen (während des Bootvorgangs, langsame Telefonleitungen, unbekannte Terminals) einwandfrei. Systemmanager brauchen ihn gelegentlich bei Konfigurationsproblemen, wenn keine Terminalbeschreibung zur Verfügung steht. Im Handbuch findet man bei `ed(1)` die Syntax regulärer Ausdrücke.

Das Kommando `ex(1)` ruft einen erweiterten Zeileneditor auf und dient nicht etwa zum Abmelden. Wird praktisch nicht mehr benutzt. Der nachfolgend beschriebene Editor `vi(1)` ist identisch mit dem `ex(1)` – beherrscht also alle `ex`-Kommandos – nur sein Verhalten gegenüber dem Benutzer ist komfortabler. Da `ex` auf einigen anderen Systemen das Kommando zum Beenden der Sitzung ist und es immer wieder vorkommt, dass Benutzer dieses Kommando mit der letztgenannten Absicht eintippen, habe ich den Editor in `exed` umbenannt und unter `ex` ein hilfreiches Shellskript eingerichtet.

Auf dem `ex(1)` baut der verbreitete UNIX-Bildschirm-Editor `vi(1)` auf[34]. Ein **Bildschirm-Editor** stellt einen ganzen Bildschirm oder mehr des Textes gleichzeitig zur Verfügung, sodass man mit dem Cursor im Text herumfahren kann. Dazu muss der `vi(1)` den Terminaltyp kennen, den

[34] `docs.FreeBSD.org/44doc/usd/12.vi/paper.ps.gz` WILLIAM JOY und MARK HORTON: An Introduction to Display Editing with Vi.

er in der Umgebungs-Variablen TERM findet. Die zugehörige **Terminal-Beschreibung** sucht er im Verzeichnis `/usr/lib/terminfo` oder in der Datei `/etc/termcap`. Falls diese fehlt oder – noch unangenehmer – Fehler enthält, benimmt sich der `vi(1)` eigenartig. Näheres zur Terminalbeschreibung unter `terminfo(4)` sowie im Abschnitt 2.14.5.2 *Terminals* auf Seite 262.

Das Arbeiten mit dem `vi(1)` wird von Leuten, die ihn nicht kennen, für schwierig gehalten. Deshalb vorweg einige Erfahrungen von langjährigen Benutzern des `vi(1)` und anderer Editoren:

- Der `vi(1)` kennt mehr als hundert Editier-Kommandos. Sie brauchen davon etwa ein Dutzend. Die weiteren lernt man nach Bedarf. Zum Schreiben des vorliegenden Buches habe ich zwei Dutzend verwendet.
- Der `vi(1)` lässt sich nahezu beliebig an persönliche Wünsche anpassen. Das kann auf der Ebene des Systems, des Benutzers oder einzelner Verzeichnisse geschehen. Dazu dienen Dateien namens `.exrc` oder ähnlich. Mit den Voreinstellungen kommt man aber schon weit.
- Der `vi(1)` hat Grenzen hinsichtlich der Zeilenlänge und Größe von Dateien. Bei normalen Texten stoßen Sie niemals an diese Grenzen. Zum Editieren extrem großer Dateien (mehr als 100 MByte) greifen Sie besser zu einem Binär-Editor, siehe Abschnitt 2.7.8 *Binär-Editoren* auf Seite 152.
- Schwierige Aufgaben in der Textverarbeitung lassen sich am einfachsten mit einem leistungsfähigen Editor bewältigen.

Das Gesagte gilt genau so für den Editor `emacs(1)`.

Da der `vi(1)` mit den unterschiedlichsten Tastaturen klar kommen muss, setzt er nur eine minimale Anzahl von Tasten voraus, im wesentlichen die Schreibmaschinentasten, Control (Ctrl oder Strg) und Escape (Esc). Was sich sonst noch an Tasten oben und rechts befindet, ist nicht notwendig. Dies führt zu einer Doppelbelegung jeder Taste. Im **Schreibmodus** (input mode) des `vi(1)` veranlasst ein Tastendruck das Schreiben des jeweiligen Zeichens auf den Bildschirm und in den Speicher. Im **Kommandomodus** (command mode) bedeutet ein Tastendruck ein bestimmtes Kommando an den Editor. Beispielsweise löscht das kleine x das Zeichen, auf dem sich gerade der Cursor befindet.

Beim Start ist der `vi` im Kommandomodus, außerdem schaltet die Escape-Taste immer in diesen Modus, auch bei mehrmaligem Drücken. In den Schreibmodus gelangt man mit verschiedenen Kommandos:

- a (append) schreibt anschließend an den Cursor,
- i (insert) schreibt vor den Cursor,
- o (open) öffnet eine neue Zeile unterhalb der aktuellen,
- r (replace) ersetzt das Zeichen auf der Cursorposition.
- R (replace) ersetzt den Text ab Cursorposition.

Die `vi(1)`-Kommandos werden auf dem Bildschirm nicht wiederholt, sondern machen sich nur durch ihre Wirkung bemerkbar. Die mit einem Doppelpunkt beginnenden Kommandos sind eigentlich `ex(1)`-Kommandos[35] und werden in der untersten Bildschirmzeile angezeigt. Weitere `vi(1)`-Kommandos im Anhang. Zum Arbeiten muss man zehn bis zwanzig im Kopf haben.

Wie bekommt man mit dem `vi(1)` das Escape-Zeichen und gegebenenfalls andere Sonderzeichen in Text? Man stellt `control-v` voran. Mit dem Kommando `u` für *undo* macht man das jüngste Kommando, das den Text verändert hat, rückgängig.

Der `vi(1)` kann Zeichenfolgen in einem Text suchen und automatisch ersetzen. Die Zeichenfolgen sind **reguläre Ausdrücke**. Um im Text vorwärts zu suchen, gibt man das Kommando `/ausdruck` ein, um rückwärts zu suchen, `?ausdruck`. Der Cursor springt auf das nächste Vorkommen von `ausdruck`. Mittels `n` wiederholt man die Suche. Wollen wir das Wort *kompilieren* durch *compilieren* ersetzen, rufen wir den `vi` mit dem Namen unserer Textdatei auf und geben folgendes Kommando ein:

```
:1,$ s/kompil/compil/g
```

Im einzelnen heißt das: von Zeile 1 bis Textende ($) substituiere die Zeichenfolge *kompil* durch *compil*, und zwar nicht nur beim ersten Auftreten in der Zeile, sondern global in der gesamten Zeile, das heißt hier also im gesamten Text. Die Zeichenfolgen brauchen nicht gleich lang zu sein. Groß- und Kleinbuchstaben sind wie immer verschiedene Zeichen, deshalb wird man die Ersetzung auch noch für große Anfangsbuchstaben durchführen. Der vorliegende Text ist auf mehrere Dateien verteilt. Soll eine Ersetzung in allen Dateien vorgenommen werden, schreibt man ein Shellskript `korr` und ruft es auf:

```
korr 's/kompil/compil/g' *.tex
```

Die korrigierten Texte findet man in den Dateien `*.tex.k` wieder, die ursprünglichen Texte bleiben vorsichtshalber erhalten.

```
# Shellscript fuer fileuebergreifende Text-Ersetzungen
print Start /usr/local/bin/korr

sedcom="$1"
shift
files="$*"

for file in $files
do
sed -e "$sedcom" $file > "$file".k
done
```

[35]Manche Autoren unterscheiden beim `vi(1)` drei Modi, indem sie beim Kommando-Modus `ex(1)`- und `vi(1)`-Kommandos trennen. `ex(1)`-Kommandos erscheinen in der Fußzeile (last line). Dann gibt es alternativ zu dem normalen Visual Mode auch noch einen Open Mode für dumme oder unbekannte Terminals. Diese Feinheiten ersparen wir uns.

```
print Ende korr
```

Quelle 2.21 : Shellskript zur Textersetzung in mehreren Dateien

Zum Löschen einer Zeichenfolge substituiert man sie durch nichts:

```
:1,$ s/(Slang)//g
```

In der Textdatei wird die Zeichenfolge `(Slang)` ersatzlos gestrichen. Die runden Klammern sind in regulären Ausdrücken keine Metazeichen, anders als in der Shell.

Beim Aufruf des `vi(1)` zusammen mit dem Namen einer existierenden Textdatei:

```
vi textfile
```

legt er eine Kopie (Editierpuffer, editing buffer) der Datei an und arbeitet nur mit der Kopie. Der `vi(1)` kann gleichzeitig mit den Kopien mehrerer Dateien arbeiten, so dass man Textteile leicht zwischen den Dateien hin- und herschieben kann (cut and paste). Erst das abschließende `write`-Kommando – meist in der Form `:wq` für *write* und *quit* – schreibt die Kopie zurück auf den Massenspeicher. Es ist zweckmäßig, auch während des Editierens von Zeit zu Zeit zurückzuschreiben – immer wenn man sich der Änderungen eines Textteiles sicher ist. Hat man Unsinn gemacht, so quittiert man den Editor ohne zurückzuschreiben, und das Original ist nicht verdorben. Man kann auch die eben editierte Kopie in eine Datei mit einem neuen Namen – gegebenenfalls in einem anderen Verzeichnis wie `/tmp` – zurückschreiben und so das Original unverändert erhalten. Will man den `vi(1)` verlassen ohne zurückzuschreiben, warnt er. Greifen zwei Benutzer gleichzeitig schreibend auf dieselbe Textdatei zu, so kann zunächst jeder seine Kopie editieren. Wer als letzter zurückschreibt, gewinnt. Das passiert auch leicht, wenn ein einzelner Benutzer gleichzeitig mit vielen Fenstern arbeitet und die Übersicht verloren hat.

Wird der Editiervorgang gewaltsam unterbrochen (Stromausfall), so steht die mit dem jüngsten `:w`-Kommando zurückgeschriebene Kopie im Massenspeicher. Aber der `vi(1)` tut von sich aus noch mehr. Er legt im jeweiligen Arbeitsverzeichnis eine Punktdatei an – mittels `ls -la` zu sehen – die laufend nachgeführt wird. Sie enthält den Text und binäre Informationen, ist also nur eingeschränkt verständlich. Beim ordnungsgemäßen Beenden des Editors wird es gelöscht, bei Stromausfall bleibt es erhalten. Rufen wir später den `vi(1)` mit dem Namen des während des Stromausfalls editierten Dateien als Argument auf, bemerkt der Editor die Existenz der Punktdatei und bietet an, diese als die jüngste Fassung des Textes weiter zu verarbeiten (recover). So gehen im schlimmsten Fall nur wenige Eingaben infolge der Unterbrechung verloren. Einige Versionen des Editors legen vor jedem Zurückschreiben eine Sicherungskopie des vorhergehenden Textes an. Man kann so nach einer Katastrophe auf drei Fassungen des Textes zurückgreifen:

- die Punkt- oder Recovery-Datei, die aktuellste Fassung,

- die Textdatei auf dem Massenspeicher mit Stand des jüngsten Zurückschreibens,
- die Sicherungskopie mit Stand vor dem jüngsten Zurückschreiben.

Ich bin dem `vi(1)` schon dankbar gewesen. Bei einem Plattencrash hilft dieses Vorgehen allerdings nichts, da müsste man mit Spiegeln (RAID) oder eigenen Skripts und `cron(1)`-Jobs arbeiten. Bei den heutigen Preisen für CD-Rohlinge würde ich beim Schreiben meiner Diplom- oder Doktorarbeit jeden Abend eine CD mit dem Manuskript brennen.

In der Datei `$HOME/.exrc` legt man individuelle **Tastatur-Anpassungen** und Editor-Variable nieder. Mit dem Kommando:

```
:set all
```

sieht man sich die gesetzten Variablen an. Ihre Bedeutung ist dem Handbuch (`man vi`) zu entnehmen. Auch in einem Unterverzeichnis darf man noch einmal eine Datei `.exrc` unterbringen, dies gilt dann für `vi(1)`-Aufrufe aus dem Unterverzeichnis. Beispielsweise setze ich für die Unterverzeichnisse, die meine C-Quellen enthalten, die Tabulatorweite auf 4 statt 8 Stellen, um die Einrückungen nicht zu weit nach rechts wandern zu lassen. Die `.exrc`-Datei für diesen Zweck enthält folgende Zeilen:

```
:set tabstop=4
:map Q :wq
```

Die zweite Zeile bildet das Kommando `Q` (ein Makro) auf das `vi(1)`-Kommando `:wq` ab. Dabei darf der Macroname kein bereits bestehendes `vi(1)`-Kommando sein. Die Ersetzung darf 100 Zeichen lang sein. Auch Funktionstasten lassen sich abbilden. Auf diese Weise kann man sich Umlaute oder häufig gebrauchte Kommandos auf einzelne Tasten legen.

Vom `vi(1)` gibt es zwei Sonderausführungen. Der Aufruf `view(1)` startet den `vi(1)` im Lesemodus; man kann alles machen wie gewohnt, nur nicht zurückschreiben. Das ist ganz nützlich zum Lesen und Suchen in Texten. Die Fassung `vedit(1)` ist für Anfänger gedacht und überflüssig, da man dieselbe Wirkung durch das Setzen einiger Parameter erreicht und die anfänglichen Gewöhnungsprobleme bleiben.

Der `vi(1)` gehört zur Grundausstattung von UNIX und wird nicht von einem besonderen Gremium gepflegt. Aus dem GNU-Projekt stammt der `vi(1)`-ähnliche Editor `elvis(1)`. Er liegt wie alle GNU-Software im Quellcode vor und kann daher auf verschiedene UNIXe und auch DOS übertragen werden. Bei MINIX und Linux gehört er zum Lieferumfang. Im Netz findet sich die `vi(1)`-Erweiterung `vim(1)` (vi improved), auch für `vi(1)`-Liebhaber, die unter DOS arbeiten:

```
www.vim.org/
```

Weitere Informationen am einfachsten per WWW-Suchmaschine.

Das soll genügen. Den `vi(1)` lernt man nicht an einem Tag. Die Arbeitsweise des `vi(1)` ist im Vergleich zu manchen Textsystemen unbequem, aber man muss die Umstände berücksichtigen, unter denen er arbeitet. Von seinen Leistungen her erfüllt er mehr Wünsche, als der Normalbenutzer hat. Man gewöhnt sich an jeden Editor, nur nicht jede Woche an einen anderen.

2.7.4 Universalgenie (emacs)

Neben dem `vi(1)` findet man auf UNIX-Systemen oft den Editor `emacs(1)`, der aus dem GNU-Projekt stammt und daher im Quellcode verfügbar ist. Es gibt auch Portierungen auf andere Systeme einschließlich IBM-PC unter DOS sowie die Varianten `microemacs` und `xemacs`. Der grundsätzliche Unterschied zum `vi(1)` ist, dass der `emacs(1)` nur einen Modus kennt und die Editierkommandos durch besondere Tastenkombinationen mit den control- und alt-Tasten vom Text unterscheidet. Im übrigen ist er mindestens so mächtig (= gewöhnungsbedürftig) wie der `vi(1)`. Chacun à son goût.

2.7.4.1 Einrichtung

Falls der Emacs nicht – wie bei den Linux-Distributionen – fertig eingerichtet vorliegt, muss man sich selbst darum bemühen. Man holt ihn sich per Anonymous FTP oder mittels eines WWW-Brausers von:

- ftp.informatik.rwth-aachen.de/pub/gnu/
- ftp.informatik.tu-muenchen.de/pub/comp/os/unix/gnu/

oder anderen Servern. Die Datei heiße `emacs-20.2.tar.gz`, sei also ein mit `gzip` gepacktes `tar`-Archiv. Man legt es in ein temporäres Verzeichnis, entpackt es und dröselt es in seine Teile auf:

```
gunzip emacs-20.2.tar.gz
tar -xf emacs-20.2.tar
```

Danach hat man neben dem Archiv ein Verzeichnis `emacs-20.2`. Man wechselt hinein und liest die Dateien `README` und `INSTALL`, die Datei `PROBLEMS` heben wir uns für später auf. In der Datei `INSTALL` wird angeraten, sich aus der Datei `./etc/MACHINES` die zutreffende Systembezeichnung herauszusuchen, in unserem Fall `hppa1.1-hp-hpux10`. Ferner soll man sich noch die Datei `leim-20.2.tar.gz` zur Verwendung internationaler Zeichensätze (Latin-1 usw.) besorgen und neben der Emacs-Datei entpacken und aufdröseln; seine Dateien gehen in das Emacs-Verzeichnis. Dann ruft man ein Shellskript auf, das ein Makefile erzeugt:

```
./configure hppa1.1-hp-hpux10
```

Es folgen `make(1)`, das hoffentlich ohne Fehlermeldung durchläuft, und `make install` (als Benutzer `root` wegen der Schreibrechte in `/usr/local/`). Als Fehler kommen in erster Linie fehlende Bibliotheken in Betracht, deren Beschaffung in Arbeit ausarten kann. Mittels `make clean` und `make distclean` lassen sich die nicht mehr benötigten Dateien löschen. Sobald alles funktioniert, sollte man auch das Verzeichnis `emacs-20.2` löschen, man hat ja noch das Archiv. Der fertige Editor – die Datei `/usr/local/bin/emacs` – sollte die Zugriffsrechte 755 haben. Mittels `man emacs` kommt die Referenz auf den Schirm.

2.7.4.2 Benutzung

Der Aufruf `emacs mytext` startet den Editor zur Erzeugung oder Bearbeitung der Textdatei `mytext`. Mittels `control-h` und `t` bekommt man ein Tutorial auf den Schirm, das vierzehn Seiten DIN A4 umfasst. Zum Einarbeiten ist das Tutorial besser als die man-Seiten. Eine *GNU Emacs Reference Card* – sechs Seiten DIN A4 – liegt dem Editor-Archiv bei. Mit `control-h` und `i` gibt es eine Information von elf Seiten Umfang, von der University of Texas zieht man sich eine *GNU Emacs Pocket Reference List* von vierzehn Seiten. Als ultimative Bettlektüre erhält man im guten Buchhandel schließlich ein Buch von 560 Seiten.

Eine Reihe von Programmen wie Compiler, Mailer, Informationsdienste arbeitet mit dem `emacs(1)` zusammen, so dass man diesen nicht zu verlassen braucht, wenn man etwas anderes als Textverarbeitung machen möchte. Unter dem Namen *emacspeak* gibt es eine Sprachausgabe für sehgeschädigte Benutzer. Das geht in Richtung integrierte Umgebungen. Eigentlich ist der `emacs(1)` gar kein Editor, sondern ein LISP-Interpreter mit einer Sammlung von Macros. Es spricht nichts dagegen, diese Sammlung zu erweitern, so dass man schließlich alles mit dem `emacs(1)` macht. Den `vi(1)` emuliert er natürlich auch.

Zu MINIX gehört der `emacs(1)`-ähnliche Editor `elle(1)`, neben dem `vi(1)`-Clone `elvis(1)`. Zu Linux gibt es den originalen `emacs(1)` neben dem `vi(1)`. Der XEmacs (ehemals Lucid Emacs) ist ein `emacs(1)` für das X Window System, der jedoch teilweise andere Wege geht als das Original aus dem GNU-Projekt. Die WWW-Seite:

`emacs.org/`

steckte Anfang 2001 noch in den Anfängen, Informationen also wie beim `vi(1)` am einfachsten mit Hilfe einer Suchmaschine.

2.7.5 Einfachst: pico

Der `pico(1)` ist ein kleiner Editor, der usprünglich zu einem Email-Programm gehörte, aber auch selbständig zu gebrauchen ist. Wer nur einfache, kurze Texte schreibt, kommt mit ihm aus. Ansehen schadet nicht.

2.7.6 Joe's Own Editor (joe)

Der `joe(1)`[36] von JOSEPH. H. ALLEN soll als Beispiel für eine Vielzahl von Editoren stehen, die im Netz herumschwimmen und entweder mehr können oder einfacher zu benutzen sind als die Standard-Editoren. Er bringt eine eigene Verhaltensweise in normaler und beschränkter Fassung mit, kann aber auch WordStar, `pico(1)` oder `emacs(1)` emulieren (nachahmen), je nach Aufruf und Konfiguration. Diese lässt sich in eine Datei `$HOME/.joerc` den

[36]Leider hat sich ein HTML-Editor aus Frankreich denselben Namen zugelegt.

eigenen Wünschen anpassen. Seine Verwendung unterliegt der GNU General Public License, das heißt sie ist kostenfrei.

Der `joe(1)` kennt keine Modi. Nach dem Aufruf legt man gleich mit der Texteingabe los. Editorkommandos werden durch control-Sequenzen gekennzeichnet. Beispielsweise erzeugt die Folge `control-k` und `h` ein Hilfefenster am oberen Bildschirmrand. Nochmalige Eingabe der Sequenz löscht das Fenster. Am Ende verlässt man den Editor mittels `control-c` ohne Zurückschreiben oder mit der Sequenz `control-k` und `x` unter Speichern des Textes. Weitere Kommandos im Hilfefenster oder mit `man joe`. In Linux-Distributionen ist `joe(1)` meist enthalten.

2.7.7 Der Nirwana-Editor (nedit)

Der `nedit(1)` setzt auf X11 auf und verwendet eine grafische Oberfläche im Stil von Motif. Er ist in vielen Linux-Distributionen enthalten und für weitere UNIXe sowie VMS zu haben. Insbesondere lässt er sich an die Eigenheiten vieler Programmiersprachen anpassen; seine Makro-Sprache ähnelt C. Wer viel programmiert, sollte sich ihn ansehen. Informationen findet man unter:

`nedit.org/`

Weitere auf X11 basierende Editoren sind `xedit(1)` und seine Fortentwicklung `axe(1)`.

2.7.8 Binär-Editoren

Text-Editoren fassen ihnen vorgelegte Dateien immer als Text auf. Es gibt aber Aufgaben, bei denen eine Datei Byte für Byte ohne jegliche Interpretation angezeigt und editiert werden muss. Das Werkzeug zu diesem Zweck sind Binär-Editoren. Die meisten zeigen einen Dateiinhalt wahlweise binär, oktal, dezimal, hexadezimal oder soweit möglich als ASCII-Zeichen an. Deshalb ist auch die Bezeichnung Hex-Editor gebräuchlich.

Die beiden Text-Editoren `vim(1)` und `emacs(1)` kennen einen Binär-Modus. Spezielle Binär-Editoren sind jedoch besser an die Aufgabe angepasst. Neben kommerziellen Produkten wie `vedit` oder `ultraedit` finden sich im Netz auch freie Binär-Editoren in Form des Quellcodes oder als Paket einer Linux-Distribution:

- `khexit` aus der KDE-Arbeitsumgebung,
- `shed`, der *Simple Hex Editor*, siehe `shed.sourceforge.ne/`,
- `lfhex`, der *Large File Hex Editor*, siehe `www.freshports.org/editors/lfhex/`,
- `bed`, der *Menu Driven Binary Editor*, siehe `bedlinux.tripod.com/`.

Ihrer Aufgabe gemäß können Binär-Editoren mit extrem großen Dateien umgehen, wo Text-Editoren versagen.

2.7.9 Stream-Editor (sed)

Der **Stream-Editor** sed(1) bearbeitet eine Textdatei zeilenweise nach Regeln, die man ihm als Option oder in einer getrennten Datei (sed-Skript) mitgibt. Er ist im Gegensatz zu den bisher genannten Editoren nicht interaktiv, er führt keinen Dialog. Die letzte Zeile der zu bearbeitenden Textdatei muss leer sein oder anders gesagt, das letzte Zeichen des Textes muss ein newline-Zeichen (Linefeed) sein.

Die einfachste Aufgabe für den sed(1) wäre der Ersatz eines bestimmten Zeichens im Text durch ein anderes (dafür gibt es allerdings ein besseres, weil einfacheres Werkzeug tr(1)). Der sed(1) bewältigt ziemlich komplexe Aufgaben, daher ist seine Syntax umfangreich. Sie baut auf der Syntax des Zeileneditors ed(1) auf. Der Aufruf

```
sed 'Kommandos' filename
```

veranlasst den sed(1), die Datei filename Zeile für Zeile einzulesen und gemäß den Kommandos bearbeitet nach stdout auszugeben. Der Aufruf

```
sed '1d' filename
```

löscht die erste Zeile in der Datei filename und schreibt das Ergebnis nach stdout. Die Quotes um das sed(1)-Kommando verhindern, dass die Shell sich das für den sed(1) bestimmte Kommando ansieht und möglicherweise Metazeichen interpretiert. Hier wären sie nicht nötig und stehen einfach aus Gewohnheit. Jokerzeichen in filename dagegen werden von der Shell zu Recht interpretiert, so dass der sed(1) von der Shell eine Liste gültiger Namen erhält.

Folgender Aufruf ersetzt alle Großbuchstaben durch die entsprechenden Kleinbuchstaben (einzeilig):

```
sed 'y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
            abcdefghijklmnopqrstuvwxyz/' filename
```

Das y-Kommando kennt keine Zeichenbereiche, wie sie bei regulären Ausdrücken oder beim Kommando tr(1) erlaubt sind, man muss die beiden notwendigerweise gleichlangen Zeichenmengen auflisten. Übrigens ist obiges Kommando ein Weg zur ROT13-Verschlüsselung, indem man die zweite Zeichenmenge mit n beginnen lässt. Geht es um den Ersatz eines festen Zeichenmusters oder eines regulären Ausdrucks durch einen festen String, so nimmt man:

```
sed 's/\\[a-z][a-z]*{..*}/LaTeX-K/g' filename
```

Im Kommando steht s für substitute. Dann folgt ein regulärer Ausdruck zur Kennzeichnung dessen, was ersetzt werden soll, hier das bereits erwähnte Muster eines LaTeX-Kommandos. An dritter Stelle ist der Ersatz (replacement) aufgeführt, hier die feste Zeichenfolge LaTeX-K, und schließlich ein Flag, das besagt, den Ersatz global (überall, nicht nur beim ersten Auftreten des regulären Ausdrucks in der Zeile) auszuführen. Das Trennzeichen zwischen den vier Teilen kann jedes beliebige Zeichen sein, es darf nur nicht in den Teilen selbst vorkommen.

Der `sed(1)` ist mächtig und nützlich, aber lernbedürftig. Erleichtert wird die Arbeit dadurch, dass man ein sed-Skript oder auch die Anweisung in der Kommandozeile Schritt für Schritt aufbauen und testen kann. Im Netz findet sich viel Material dazu, lassen Sie eine Suchmaschine nach `unix AND editor AND sed` suchen.

Merke: Der `vi(1)` ist ein interaktiver Editor, der Tastatureingaben erfordert und nicht Bestandteil einer Pipe sein oder im Hintergrund laufen kann. Der `sed(1)` ist ein Filter, das keine Tastatureingaben verlangt, Glied einer Pipe oder eines Shellskripts sein und unbeaufsichtigt laufen kann.

2.7.10 Listenbearbeitung (awk)

Das Werkzeug `awk(1)` ist nach seinen Urhebern Alfred V. Aho, Peter J. Weinberger und Brian W. Kernighan benannt und firmiert als programmierbares **Filter** oder **Listengenerator**. Es lässt sich auch als eine Programmiersprache für einen bestimmten, engen Zweck auffassen. Der `awk(1)` bearbeitet eine Textdatei zeilenweise, wobei er jede Zeile – auch Satz genannt – in Felder zerlegt. Eine typische Aufgabe ist die Bearbeitung von Listen. Hier ist er angenehmer als der `sed(1)`, allerdings auch langsamer. Für die Verwaltung eines kleinen Vereins ist er recht, für das Telefonbuch von Berlin nicht.

In einfachen Fällen werden dem `awk(1)` beim Aufruf die Befehle zusammen mit den Namen der zu bearbeitenden Dateien mitgegeben, die Befehle in Hochkommas, um sie vor der Shell zu schützen:

```
awk 'befehle' files
```

Ein `awk(1)`-Befehl besteht aus den Teilen **Muster** und **Aktion**. Jede Eingabezeile, auf die das Muster zutrifft, wird entsprechend der Aktion behandelt. Die Ausgabe geht auf `stdout`. Ein Beispiel:

```
awk '{if (NR < 8) print $0}' myfile
```

Die Datei `myfile` wird Zeile für Zeile gelesen. Die vorgegebene `awk(1)`-Variable `NR` ist die Zeilennummer, beginnend mit 1. `$0` ist die ganze jeweilige Zeile. Falls die Zeilennummer kleiner als 8 ist, wird die Zeile nach `stdout` geschrieben. Es werden also die ersten 7 Zeilen der Datei ausgegeben. Nun wollen wir das letzte Feld der letzten Zeile ausgeben:

```
awk 'END {print $NF}' myfile
```

Das Muster `END` trifft zu, wenn die letzte Zeile verarbeitet ist. Üblicherweise gilt die zugehörige Aktion irgendwelchen Abschlussarbeiten. Die Variable `NF` enthält die Anzahl der Felder der Zeile, die Variable `$NF` ist also das letzte Feld. Nun wird es etwas anspruchsvoller:

```
awk '$1 != prev { print; prev = $1 }' wortliste
```

Die Datei `wortliste` enthalte in alphabetischer Folge Wörter und gegebenenfalls weitere Bemerkungen zu den Wörtern, pro Wort eine Zeile. Der `awk(1)` liest die Datei zeilenweise und spaltet jede Zeile in durch Spaces oder Tabs getrennte Felder auf. Die Variable `$1` enthält das erste Feld, also hier das Wort zu Zeilenbeginn. Falls dieses Wort von dem Wort der vorangegangenen Zeile abweicht (Variable `prev`), wird die ganze Zeile ausgegeben und das augenblickliche Wort in die Variable `prev` gestellt. Zeilen, die im ersten Feld übereinstimmen, werden nur einmal ausgegeben. Dieser `awk(1)`-Aufruf hat eine ähnliche Funktion wie das UNIX-Kommando `uniq(1)`. Da Variable mit dem Nullstring initialisiert werden, wird auch die erste Zeile richtig bearbeitet.

Wenn die Anweisungen an den `awk(1)` umfangreicher werden, schreibt man sie in eine eigene Datei (awk-Skript). Der Aufruf sieht dann so aus:

```
awk -f awkskript textfiles
```

awk-Skripte werden in einer Sprache geschrieben, die teils an Shellskripte, teils an C-Programme erinnert. Sie bestehen – wie ein deutscher Schulaufsatz – aus Einleitung, Hauptteil und Schluss. Sehen wir uns ein Beispiel an, das mehrfache Eintragungen von Stichwörtern in einem Sachregister aussortiert und die zugehörigen Seitenzahlen der ersten Eintragung zuordnet:

```
# awk-Script fuer Sachregister

BEGIN { ORS = ""
          print "Sachregister"
        }
        {
          if ($1 == altwort)
                  print ", " $NF
          else
                  {
                  print "\n" $0
                  altwort = $1
                  nor++
                  }
        }
END     { print "\n\n"
          print "gelesen: " NR "  geschrieben: " nor "\n"
        }
```

Quelle 2.22 : awk-Skript für Sachregister

Das Doppelkreuz markiert einen Kommentar. Der Einleitungsblock wird mit `BEGIN` gekennzeichnet, der Hauptteil steht nur in geschweiften Klammern und der Schluss beginnt mit `END`. Die vorbestimmte, `awk(1)`-eigene Variable `ORS` (Output Record Separator, d. h. Trennzeichen zwischen Sätzen in der Ausgabe), defaultmäßig das Newline-Zeichen, wird mit dem Nullstring initialisiert. Dann wird die Überschrift *Sachregister* ausgegeben.

Im Hauptteil wird das aktuelle erste Feld gegen die Variable `altwort` geprüft. Bei Übereinstimmung werden ein Komma, ein Space und das letzte

Feld der aktuellen Zeile ausgegeben, nämlich die Seitenzahl. Die `awk(1)`-eigene Variable `NF` enthält die Anzahl der Felder des aktuellen Satzes, die Variable `$NF` mithin das letzte Feld.

Bei Nichtübereinstimmung (einem neuen Stichwort also) werden ein Newline-Zeichen und dann die ganze Zeile (`$0`) ausgegeben. Anschließend werden das erste Feld in die Variable `altwort` gestellt und die vom Programmierer definierte Variable `nor` inkrementiert. So wird mit der ganzen Textdatei verfahren.

Am Ende der Textdatei angelangt, werden noch zwei Newline-Zeichen, die `awk(1)`-eigene Variable `NR` (Number of Records) und die Variable `nor` ausgegeben. Die Aufgabe wäre auch mit dem `sed(1)` oder einem C-Programm zu lösen, aber ein `awk`-Skript ist der einfachste Weg. Der `awk(1)` vermag noch viel mehr.

Eine Besonderheit des `awk(1)` sind Vektoren mit Inhaltindizierung (associative array). In Programmiersprachen wie C oder FORTRAN werden die Elemente eines Arrays oder Vektors mit fortlaufenden ganzen Zahlen (Indizes) bezeichnet. Auf ein bestimmtes Element wird mittels des Arraynamens und des Index zugegriffen:

```
arrayname[13]
```

In einem `awk`-Array dürfen die Indizes nicht nur ganze Zahlen, sondern auch beliebige Strings sein:

```
telefon[''Meyer'']
```

ist eine gültige Bezeichnung eines Elementes. Es könnte die Anzahl der Telefonanschlüsse namens Meyer in einem Telefonbuch enthalten.

Neuere Alternativen zu `awk(1)` sind GNU `gawk` und Perl. Letzteres ist eine interpretierte Programmiersprache zur Verarbeitung von Textdateien, die Elemente aus C, `sed(1)`, `awk(1)` und der Shell `sh(1)` enthält. Ihre Möglichkeiten gehen über das Verarbeiten von Texten hinaus in Richtung Shellskripte, siehe Abschnitt 2.5.3 *Noch eine Skriptsprache: Perl* auf Seite 114.

2.7.11 Verschlüsseln (crypt)

2.7.11.1 Aufgaben der Verschlüsselung

Auf einem UNIX-System kann der Verwalter (Superuser) auf jede Datei zugreifen, auf Microsoft Windows mit gewissen Einschränkungen auch. Das Netz ist mit einfachen Mitteln unauffällig abzuhören. Will man seine Daten vor Unbefugten schützen, hilft nur Verschlüsseln. Man darf aber nicht vergessen, dass bereits die Analyse des Datenverkehrs einer Quelle oder eines Ziels Informationen liefert. Wer ganz unbemerkt bleiben will, muss sich mehr einfallen lassen als nur eine Verschlüsselung.

Eng verwandt mit der **Verschlüsselung** (encryption, cryptage, chiffrement) ist die **Authentifizierung** oder Authentisierung (authentication, authentification). Nachstehend geht es nur darum, einen Text oder auch andere

Daten für Unbefugte unbrauchbar zu machen; für Befugte sollen sie natürlich weiterhin brauchbar bleiben.

Das Ganze ist heute eine Wissenschaft und heißt **Kryptologie**. In den letzten Jahrzehnten hat sie einen stark mathematischen Einschlag bekommen. Trotzdem bietet sie einen gewissen Unterhaltungswert, insbesondere die **Kryptanalyse**, der Versuch, Verschlüsselungen zu knacken.

Die zu verschlüsselnden Daten nennen wir **Klartext** (plain text), die verschlüsselten Daten **Geheimtext** (cipher text).

2.7.11.2 Symmetrische Verfahren

Im einfachsten Fall wird jedes Zeichen des Klartextes nach einer Regel durch ein anderes Zeichen desselben Alphabetes ersetzt. Die einfachste Regel dieses Falles ist die Verschiebung um eine feste Anzahl von Stellen im Alphabet, beispielsweise um +3 Stellen. Aus A (Zeichen Nr. 1) wird D (Zeichen Nr. 1+3). Dieses Verfahren soll CAIUS JULIUS CAESAR benutzt haben. Er vertraute auf die Dummheit seiner Gegner. Zum Entschlüsseln des Geheimtextes nimmt man dasselbe Verfahren mit −3 Stellen. Wählt man eine Verschiebung um 13 Stellen, so führt bei einem Alphabet mit 26 Zeichen eine Wiederholung der Verschlüsselung zum Klartext zurück. Dieses Verfahren ist unter dem Namen **ROT13** bekannt und wird im Netz verwendet, um einen Text – beispielsweise die Auflösung eines Rätsels – zu verfremden. ROT26 gilt als die schwächste aller Verschlüsselungen des lateinischen Alphabetes. Man kann die Verfahren raffinierter gestalten, indem man Zeichengruppen verschlüsselt, Blindzeichen unter den Geheimtext mischt, die Algorithmen wechselt usw.

Seit 1970 unterscheidet man zwei Gruppen von Verfahren:

- Symmetrische Verfahren (Private-Key-V.),
- Unsymmetrische Verfahren (Public-Key-V.).

Dazu kommen für bestimmte Aufgaben noch die Einweg-Hash-Verfahren. Bei den **symmetrischen Verfahren** kennen Sender und Empfänger neben dem Algorithmus sowohl den Chiffrier- wie den Dechiffrierschlüssel. Beide Schlüssel sind identisch oder voneinander ableitbar. Da der Algorithmus kaum geheim zu halten ist, beruht die Sicherheit auf dem Schlüssel, der nicht zu simpel sein darf und geheim bleiben muss. Das Problem liegt darin, den Schlüssel zum Empfänger zu schaffen. Das geht nur über einen vertrauenswürdigen Kanal, also nicht über Email. Treffen Sie Ihren Brieffreund gelegentlich bei Kaffee und Kuchen, können Sie ihm einen Zettel mit dem Schlüssel zustecken. Wohnen Sie in Karlsruhe, Ihre Brieffreundin in Fatmomakke, wird der Schlüsselaustausch aufwendiger. Ein weiteres Problem liegt in der Anzahl der benötigten Schlüssel beim Datenverkehr unter mehreren Beteiligten. Geht es nur darum, Daten vor dem Superuser zu verbergen, ist kein Schlüsselaustausch nötig und daher ein symmetrisches Verfahren angebracht.

Die Verschlüsselung nach dem weit verbreiteten **Data Encryption Standard** (DES) gehört in diese Gruppe, zur Ver- und Entschlüsselung wird derselbe Schlüssel benutzt. DES wurde von IBM entwickelt und 1977 von der US-Regierung als Standard angenommen. Es gilt heute schon nicht mehr als sicher, Triple-DES ist besser. Weitere Mitglieder dieser Gruppe sind IDEA, Blowfish und CAST5. Symmetrische Verfahren arbeiten im allgemeinen schneller als unsymmetrische.

Unter UNIX stehen ein Kommando `crypt(1)` sowie eine C-Standardfunktion `crypt(3)` zur Verfügung, die ein nicht sehr ausgefeiltes symmetrisches Verfahren verwenden. Man ver- und entschlüsselt mittels des Kommandos:

```
crypt < eingabe > ausgabe
```

Das Kommando fragt nach einem Schlüssel. Dieser wird für beide Richtungen eingesetzt. Der Klartext ist erforderlichenfalls gesondert zu löschen (physikalisch, nicht nur logisch, das heißt zu überschreiben). Die Crypt Breaker's Workbench enthält alles Nötige, um diese Verschlüsselung zu knacken (`axion.physics.ubc.ca/cbw.html`).

2.7.11.3 Unsymmetrische Verfahren

Die asymmetrischen Verfahren verwenden zum Verschlüsseln und Entschlüsseln zwei völlig verschiedene, nicht voneinander ableitbare Schlüssel. Benutzer A hat sich ein Paar zusammengehöriger Schlüssel gebastelt, den ersten zum Verschlüsseln, den zweiten zum Entschlüsseln, wie, werden wir noch sehen. Den ersten Schlüssel gibt er öffentlich bekannt, daher **Public Key**. Jeder kann ihn benutzen, zum Beispiel Benutzer B, der A eine vertrauliche Email schicken möchte. Was einmal damit verschlüsselt ist, lässt sich nur noch mit dem zweiten Schlüssel entschlüsseln, und den hält Benutzer A geheim. Er teilt ihn niemandem mit, daher **Private Key**.

Jetzt kann es nur noch passieren, dass ein Benutzer C unter Missbrauch des Namens von B an A eine beleidigende Mail schickt und B darauf hin mit A Krach bekommt. Veröffentlicht A den Dechiffrierschlüssel und behält den Chiffrierschlüssel für sich, kann er chiffrierte Texte versenden, die jeder entschlüsseln und lesen kann, wobei die Texte nur von A chiffriert worden sein können. Das ist das Authentifizierungs-Problem.

Wie kommt man nun zu einem derartigen Schlüsselpaar? Ein Weg beruht auf der Tatsache, dass man leicht zwei ganze Zahlen großer Länge miteinander multiplizieren kann, sogar ohne Rechner, während die Zerlegung einer großen Zahl (um die zweihundert dezimale Stellen entsprechend etwa 500 Bits) in ihre Primfaktoren mit den heute bekannten Algorithmen und Rechnern aufwendig ist, jedenfalls wenn gewisse Voraussetzungen eingehalten werden. RON RIVEST, ADI SHAMIR und LEONARD ADLEMAN haben auf diesem Gedanken aufbauend das verbreitete **RSA-Verfahren** entwickelt.

Man wähle zufällig zwei große Primzahlen p und q, zweckmäßig von annähernd gleicher Länge. Ihr Produkt sei $n = pq$. Weiter wähle man eine Zahl e so, dass e und $(p-1)(q-1)$ teilerfremd (relativ prim) zueinander sind. Eine vierte Zahl d berechne man aus:

$$d = e^{-1} \, mod \, ((p-1)(q-1)) \tag{2.1}$$

Die Zahlen e und n bilden den öffentlichen Schlüssel, die Zahl d ist der private, geheime Schlüssel. Die beiden Primzahlen p und q werden nicht weiter benötigt, müssen aber geheim bleiben (löschen).

Wir sehen den Klartext K als eine Folge von Ziffern an. Er wird in Blöcke K_i kleiner n aufgeteilt. Die Geheimnachricht G besteht aus Blöcken G_i, die sich nach

$$G_i = K_i^e \; mod \, n \tag{2.2}$$

berechnen. Zur Entschlüsselung berechnet man

$$K_i = G_i^d \; mod \, n \tag{2.3}$$

Einzelheiten und Begründung hierzu siehe die Bücher von FRIEDRICH L. BAUER oder BRUCE SCHNEIER. Nun ein Beispiel aus dem Buch von F. L. BAUER. Wir wählen einen Text aus lateinischen Buchstaben samt Zwischenraum und ersetzen die Zeichen durch die Nummern von 00 bis 26. Er bekommt folgendes Aussehen:

$$K = 051818011805000821\ldots \tag{2.4}$$

und wählen:

$$p = 47 \qquad q = 59 \qquad n = p * q = 2773 \tag{2.5}$$

Wir teilen den Klartext in vierziffrige Blöcke kleiner n auf:

$$K_1 = 0518 \qquad K_2 = 1801 \qquad K_3 = 1805\ldots \tag{2.6}$$

Zur Bestimmung von e berechnen wir:

$$(p-1)(q-1) = 46 * 58 = 2668 \tag{2.7}$$

Die Zahl 2668 hat die Teiler 2, 4, 23, 29, 46, 58, 92, 116, 667 und 1334. Für e wählen wir 17, teilerfremd zu 2668. Dann ergibt sich d zu:

$$d = 17^{-1} \; mod \, 2668 \tag{2.8}$$

Diese vielleicht unbekannte Schreibweise ist gleichbedeutend damit, ein Paar ganzer Zahlen d, x so zu bestimmen, dass die Gleichung:

$$d * 17 = 2668 * x + 1 \tag{2.9}$$

erfüllt ist. Die Zahl $d = 157$ ist eine Lösung mit $x = 1$. Gezielt ermittelt man Lösungen mittels des Erweiterten Euklidischen Algorithmus. Nun haben wir mit n, e und d alles, was wir brauchen und gehen ans Verschlüsseln:

$$G_1 = K_1^e \; mod \, n = 0518^{17} \; mod \, 2773 = 1787 \tag{2.10}$$

und entsprechend für die weiteren Blöcke. Gleiche Klartextblöcke ergeben gleiche Geheimtextblöcke, was bereits ein Risiko ist. Zum Entschlüsseln berechnet man:

$$K_1 = G_1^d \; mod \, n = 1787^{157} \; mod \, 2773 = 518 \tag{2.11}$$

und so weiter. Die Arithmetik großer Ganzzahlen ist für Rechner kein Problem, für Taschenrechner eher. Man kann sie sogar in Silizium gießen und erhält schnelle Chips zum Ver- und Entschlüsseln, ohne Software bemühen zu müssen. Da n und e öffentlich sind, könnte man durch Zerlegen von n in seine Primfaktoren leicht den privaten Schlüssel d ermitteln, aber das Zerlegen großer Zahlen ist nach heutigem Wissensstand sehr aufwendig.

Es gibt weitere unsymmetrische Verfahren wie das von TAHER ELGAMAL. Wird das Dokument symmetrisch verschlüsselt und der dazu erforderliche Schlüssel unsymmetrisch verschlüsselt mitgeteilt, spricht man auch von hybriden Verfahren. Auf `www.rsa.com/` findet sich Material zur Vertiefung des Themas. Eine zehnteilige FAQ-Sammlung zur Kryptografie liegt im Netz.

2.7.11.4 Angriffe (Kryptanalyse)

Angriffe auf verschlüsselte Daten – wissenschaftlich als **Kryptanalyse**, sonst als Cracking bezeichnet – gehen möglichst von irgendwelchen bekannten oder vermuteten Zusammenhängen aus. Das kleinste Zipfelchen an Vorkenntnissen kann entscheidend sein[37]. Die Wahrscheinlichkeit, dass ein Benutzer seinen nur gering modifizierten Benutzernamen als Passwort verwendet, ist leider hoch. Damit fängt man an. Das Ausprobieren aller nur möglichen Schlüssel wird **Brute Force Attack** genannt und ist bei kurzen Schlüsseln dank Rechnerhilfe auch schnell von Erfolg gekrönt. Das Faktorisieren kleiner Zahlen ist ebenfalls kein Problem. Aber selbst bei großen Zahlen, die für einen einzelnen Rechner – auch wenn er zu den schnellsten gehört – eine praktisch unlösbare Aufgabe darstellen, kommt man in kurzer Zeit zum Ziel, wenn man die Leerlaufzeiten von einigen Hundert durchschnittlichen Rechnern für seinen Zweck einsetzen kann. Das ist ein organisatorisches Problem, kein mathematisches, und bereits gelöst, siehe `www.distributed.net/rc5/`. Das ganze Nachdenken über sichere Verschlüsselung erübrigt sich im übrigen bei schlampigem Umgang mit Daten und Schlüsseln. Der Benutzer ist erfahrungsgemäß das größte Risiko.

2.7.12 Formatierer

2.7.12.1 Inhalt, Struktur und Aufmachung

Ein Schriftstück - sei es Brief oder Buch - hat einen **Inhalt**, nämlich Text, gegebenenfalls auch Abbildungen, der in einer bestimmten **Form** dargestellt ist. Bei der Form unterscheiden wir zwischen der logischen **Struktur** und ihrer Darstellung auf Papier oder Bildschirm, **Aufmachung** oder **Layout** genannt. Beim Schreiben des Manuskriptes macht sich der Autor Gedanken über Struktur und Inhalt, aber kaum über Schrifttypen und Schriftgrößen, den Satzspiegel, den Seitenumbruch, die Numerierung der Abbildungen. Das ist Aufgabe des Metteurs oder Layouters im Verlag, der seinerseits möglichst

[37] Beim Knacken von Enigma spielte eine Rolle, dass der Gegner wusste, dass ein Buchstabe niemals durch sich selbst verschlüsselt wurde.

wenig am Inhalt ändert. **Schreiben** und **Setzen** sind unterschiedliche Aufgaben, die unterschiedliche Kenntnisse erfordern.

Der Rechner wird als Werkzeug für alle drei Aufgaben (Inhalt, Struktur, Layout) eingesetzt. Mit einem Editor schreibt man einen strukturierten Text, weitergehende Programme prüfen die Rechtschreibung, helfen beim Erstellen eines Sachregisters, analysieren den Stil. Ein **Satz- oder Formatierprogramm** erledigt den Zeilen- und Seitenumbruch, sorgt für die Numerierung der Abschnitte, Seiten, Abbildungen, Tabellen, Fußnoten und Formeln, legt die Schriftgrößen fest, ordnet die Abbildungen in den Text ein, stellt das Inhaltsverzeichnis zusammen usw. Während es einfache Formatierer gibt, erfüllt ein **Satzprogramm** höhere Ansprüche und besteht daher aus einem ganzen Programmpaket.

Der UNIX-Formatierer `nroff(1)` und das Satzprogramm LaTeX (TeX, LaTeX, pdftex etc.) halten Struktur und Layout auseinander. Man schreibt mit einem beliebigen Editor den strukturierten Text und formatiert anschließend. LaTeX verfolgt darüber hinaus den Gedanken, dass der Autor seine Objekte logisch beschreiben und von der typografischen Gestaltung, dem Layout, möglichst die Finger lassen soll. Der Autor soll sagen: *Jetzt kommt eine Kapitelüberschrift* oder *Jetzt folgt eine Fußnote*. LaTeX legt dann nach typografischen Regeln die Gestaltung fest. Man kann darüber streiten, ob diese Regeln das Nonplusultra der Schwarzen Kunst sind, ihr Ergebnis ist jedenfalls besser als vieles, was Laien sich ausdenken.

Sowohl `nroff(1)` wie LaTeX zielen auf die Wiedergabe der Dokumente mittels Drucker auf Papier ab. Mit Hilfe von Programmen wie `gv(1)` oder `xdvi(1)` lassen sich die Ergebnisse vor dem Ausdrucken auf dem Bildschirm beurteilen. Die Hypertext Markup Language HTML, deren Anwendung wir im Abschnitt über das World Wide Web begegnen, hat viel mit LaTeX gemeinsam, eignet sich jedoch in erster Linie für Dokumente, die auf dem Bildschirm dargestellt werden sollen. Auch sie trennt Struktur und Layout.

Textverarbeitungsprogramme wie Word, Wordperfect oder Wordstar – häufig mit einigen weiteren Anwendungen zu sogenannten Office-Paketen gebündelt – haben Formatierungsaufgaben integriert, sodass man sich am Bildschirm während des Schreibens den formatierten Text ansehen kann. Diese Möglichkeit wird als **WYSIWYG** bezeichnet: *What you see is what you get*, auf französisch *Tel écran – tel écrit*. Das erleichtert bei einfachen Aufgaben die Arbeit, bei anspruchsvollen Manuskripten hat es der Autor jedoch leichter, wenn er sich zunächst auf Inhalt und Struktur konzentrieren kann. Außerdem birgt WYSIWYG eine Versuchung in sich. Die reichen Mittel aus dem typografischen Kosmetikkoffer namens **Desktop Publishing** sind sparsam einzusetzen, unser Ziel heißt Lesbarkeit, nicht Barock.

Beim Arbeiten mit Formatierern wie LaTeX – gegebenfalls in Verbindung mit einem Versionskontrollsystem wie RCS – kommen die Schwierigkeiten am Anfang, wenn man eine Reihe von Kommandos lernen muss, wenig Vorlagen hat und viel Mühe in die Strukturierung des Projektes steckt, die sich nicht auf Papier niederschlägt. Beim Arbeiten mit WYSIWYG-Programmen hat man schnell Erfolgserlebnisse, die Schwierigkeiten kommen später, wenn die Manuskripte umfangreicher und komplexer werden, aber dann wird ein Umstieg teuer.

2.7.12.2 Ein einfacher Formatierer (adjust)

Ein einfacher Formatierer ist `adjust(1)`. Der Aufruf

```
adjust -j -m60 textfile
```

versucht, den Text in `textfile` beidseits bündig (Blocksatz) mit 60 Zeichen pro Zeile zu formatieren. Die Ausgabe geht nach `stdout`. `adjust(1)` trennt keine Silben, sondern füllt nur mit Spaces auf. Für bescheidene Anforderungen geeignet.

2.7.12.3 UNIX-Formatierer (nroff, troff)

Die Standard-Formatierer in UNIX sind `nroff(1)` für Druckerausgabe und sein Verwandter `troff(1)` für Fotosatzbelichter. Da wir letztere nicht haben, ist bei uns `troff(1)` nicht installiert. Das `n` steht für new, da der Vorgänger von `nroff(1)` ein `roff` war, und dieser hieß so, weil man damit `run off to the printer` verband.

Eine Datei für `nroff(1)` enthält den unformatierten Text und `nroff`-Kommandos. Diese stehen stets in eigenen Zeilen mit einem Punkt am Anfang. Ein `nroff`-Text könnte so beginnen:

```
.po 1c
.ll 60
.fi
.ad c

.cu 1
Ein Textbeispiel

von W. Alex

.ad b
.ti 1c
Dies ist ein Beispiel fuer einen Text, der mit nroff
formatiert werden soll. Er wurde mit dem Editor vi
geschrieben.

.ti 1c
Hier beginnt der zweite Absatz.
Die Zeilenlaenge in der Textdatei ist unerheblich.
Man soll die Zeilen kurz halten.
Fuer die Ausgabe formatiert nroff die Zeilen.
```

Die `nroff`-Kommandos bedeuten folgendes:

- `po 1c` page offset 1 cm (zusätzlicher linker Seitenrand)

- `ll 60` line length 60 characters
- `fi` fill output lines (für Blocksatz)
- `ad c` adjust center
- `cu 1` continuous underline 1 line (auch Spaces unterstreichen)
- `ad b` adjust both margins
- `ti 1c` temporary indent 1 cm

Die Kommandos können wesentlich komplexer sein als im obigen Beispiel, es sind auch Makros, Abfragen und Rechnungen möglich. S. R. BOURNE führt in seinem im Anhang L *Zum Weiterlesen* auf Seite 358 genannten Buch die Makros auf, mit denen die amerikanische Ausgabe seines Buches formatiert wurde. Es gibt ganze Makrobibliotheken zu `nroff(1)`.

Da sich Formeln und Tabellen nur schlecht mit den Textbefehlen beschreiben lassen, verwendet man für diese beiden Fälle eigene Befehle samt Präprozessoren, die die Spezialbefehle in `nroff(1)`-Befehle umwandeln. Für Tabellen nimmt man `tbl(1)`, für Formeln `neqn(1)`, meist in Form einer Pipe:

```
tbl textfile | neqn | nroff | col | lp
```

wobei `col(1)` ein Filter zur Behandlung von Backspaces und dergleichen ist.

2.7.12.4 LaTeX

TeX ist eine Formatierungssoftware, die von DONALD ERVIN KNUTH entwickelt wurde – dem Mann, der seit Jahrzehnten an dem siebenbändigen Werk *The Art of Computer Programming* schreibt und hoffentlich noch lange lebt. Die Stärke der Software sind umfangreiche mathematische Texte, seine Schwäche war die Grafik. Inzwischen gibt es Zusatzprogramme (TeX-CAD u. a.) zu LaTeX, die es erleichtern, den Text durch Zeichnungen zu ergänzen. Außerdem kann man Grafiken bestimmter Formate – Encapsulated PostScript, auch jpeg-Fotos – in den Text einbinden. Seine Grenzen findet TeX/LaTeX bei der Gestaltung von:

- Zeitungen mit mehreren Spalten, vielen Überschriften, Anzeigen und dergleichen. Für diese Aufgabe braucht man Layout-Programme wie Quark Xpress (kommerziell).
- Bildbänden oder Fotoalben mit vielen großformatigen Abbildungen. Hier kann man sich helfen, indem man Bild- und Textteil voneinander trennt. Das gesamte Werk lässt sich dann wieder mit LaTeX einrichten. Wenn jedes Bild samt Unterschrift genau eine Seite beansprucht, wird es wieder einfach.

TeX ist sehr leistungsfähig, verlangt aber vom Benutzer die Kenntnis vieler Einzelheiten, ähnlich wie das Programmieren in Assembler. **LaTeX** ist eine **Makrosammlung**, die auf TeX aufbaut. Die LaTeX-Makros von LESLIE LAMPORT erleichtern bei Standardaufgaben und -formaten die Arbeit beträchtlich, indem viele TeX-Befehle zu einfach anzuwendenden LaTeX-Befehlen zusammengefasst werden. Kleinere Modifikationen der Standardeinstellungen sind vorgesehen, weitergehende Sonderwünsche erfordern das Hinabsteigen auf TeX-Ebene. LaTeX eignet sich für alle Arten von Dokumenten, vom Brief bis zum mehrbändigen Lebenswerk. Es steht für viele Systeme kostenfrei zur Verfügung. Alle verbreiteten Linux-Distributionen bringen LaTeX mit.

Computeralgebrasysteme wie Maple arbeiten mit LaTeX zusammen; aus den Arbeitsblättern lassen sich LaTeX-Artikel erzeugen. Diese kann man entweder als selbständige Dokumente behandeln oder nach etwas Editieren (Löschen des Vorspanns) als Kapitel oder Abschnitte in andere LaTeX-Dokumente einbinden. Man hat damit sozusagen ein LaTeX, das rechnen kann.

Das Adobe-pdf-Format (Portable Document Format), eine Weiterentwicklung von PostScript, hat sich sehr verbreitet. Es führt zu kompakteren Dateien und ist vielseitiger. Zum Lesen oder Drucken von pdf-Dateien braucht man den kostenlos erhältlichen *Adobe Reader*. Es gibt Werkzeuge zum Umwandeln von PostScript nach pdf; besser jedoch ist es, aus LaTeX-Manuskripten mittels `pdflatex` unmittelbar pdf-Dateien zu erzeugen. Zwischen beiden LaTeX-Programmen bestehen kleine Unterschiede beim Einbinden von Fotos in den Text.

Dokumentklassen Das wichtigste Stilelement ist die **Dokumentklasse**. Diese bestimmt die Gliederung und wesentliche Teile der Aufmachung. Die Dokumentklasse *book* kennt Bände, Kapitel, Abschnitte, Unterabschnitte usw. Ein Inhaltsverzeichnis wird angelegt, auf Wunsch und mit etwas menschlicher Unterstützung auch ein Sachregister. Bei der Dokumentklasse *report* beginnt die Gliederung mit dem Kapitel, ansonsten ist sie dem Buch ähnlich. Das ist die richtige Klasse für Dissertationen, Diplomarbeiten, Forschungsberichte, Skripten und dergleichen. Mehrbändige Werke sind in diesem Genre eher selten. Die Dokumentklasse *article* eignet sich für Aufsätze und kurze Berichte. Die Gliederung beginnt mit dem Abschnitt, die Klasse kennt kein Inhaltsverzeichnis, dafür aber ein Abstract. Nützlich ist auch die Dokumentklasse *foils* zur Formatierung von Folien für Overhead-Projektoren. Hier wird das Dokument in Folien gegliedert, die Schriftgöße beträgt 25 oder 30 Punkte. Zum Schreiben von Briefen gibt es eine Klasse – `dinbrief` – mit deren Hilfe sich anspruchsvoll gestaltete, normgerechte Briefe erstellen lassen.

Arbeitsweise Man schreibt seinen Text mit einem beliebigen **Editor**. Dabei wird nur von den druckbaren Zeichen des US-ASCII-Zeichensatzes zuzüglich Linefeed Gebrauch gemacht. In den Text eingestreut sind **LaTeX-Anweisungen**. Der Name der Textdatei muss die Kennung `.tex` haben. Dann schickt man die Textdatei durch den **LaTeX-Compiler**. Dieser erzeugt eine Binärdatei, deren Namen die Kennung `.dvi` trägt. Das bedeutet **device independent**, die Binärdatei ist noch nicht auf ein bestimmtes Ausgabegerät hin ausgerichtet. Mittels eines geräteabhängigen Treiberprogrammes wird aus der dvi-Datei die bit-Datei erzeugt – Kennung `.bit` – das mit einem UNIX-Kommando wie `cat` durch eine hundertprozentig transparente Schnittstelle zum Ausgabegerät geschickt wird. Es gibt ein Programm `dvips(1)`, das eine PostScript-Datei erzeugt, welche auf einem beliebigen PostScript-Drucker ausgegeben werden kann. Bei `pdflatex` entfällt der Zwischenschritt über die dvi-Datei. Man erhält sofort eine pdf-Datei.

Format dieses Textes Der vorliegende Text wurde mit LaTeX2e auf einem Linux-PC formatiert, mittels `dvips(1)` auf PostScript umgesetzt und auf einem Laserdrucker von Hewlett-Packard ausgegeben. Zunehmend mache ich jedoch von `pdflatex` Gebrauch. Im wesentlichen verwende ich die Standardvorgaben. Die Zeichnungen wurden mit TeXCAD oder `xfig(1)` entworfen. TeXCAD erzeugt LaTeX-Dateien, die man editieren kann, so man das für nötig befindet.

Im Text kommen Quelltexte von Programmen vor, die wir ähnlich wie Abbildungen oder Tabellen in einer eigenen Umgebung formatieren wollten. Eine solche Umgebung war seinerzeit nicht fertig zu haben. Man musste selbst zur Feder greifen, möglichst unter Verwendung vorhandenen Codes. Die Schwierigkeit lag darin herauszufinden, wo was definiert wird, da viele Makros wieder von anderen Makros abhängen. Die neue `source`-Umgebung wird zusammen mit einigen weiteren Wünschen in einer Datei `alex.sty` definiert, das dem LaTeX-Kommando `usepackage` als Argument mitgegeben wird:

```
% alex.sty mit Erweiterungen fuer Skriptum, 1996-11-30
%
% Aenderungen von latex.tex, report.sty, rep12.sty
%
% Meldung fuer Bildschirm und main.log:
\typeout{Option alex.sty, 1996-11-30 W. Alex}
%
% Stichwoerter hervorheben und in Index aufnehmen:
% (hat sich als unzweckmaessig erwiesen,
% Text nach und nach verbessern)
\newcommand{\stw}[1]{{\em#1}\index{#1}}
%
% Hervorhebungen in Boldface mittels \em:
\renewcommand{\em}{\bf}
%
% Fussnoten-Schriftgroesse vergroessern:
\renewcommand{\footnotesize}{\small}
%
% in Tabellen:
\newcommand{\x}{\hspace*{10mm}}
\newcommand{\h}{\hspace*{25mm}}
\newcommand{\hh}{\hspace*{30mm}}
\newcommand{\hhh}{\hspace*{40mm}}
%
% falls \heute nicht bekannt:
\newcommand{\heute}{\today}
```

```
%
% Abkuerzung:
\newcommand{\lra}{\longrightarrow}
%
% Plazierung von Gleitobjekten (Bilder usw.):
% totalnumber ist die maximale Anzahl von
% Gleitobjekten pro Seite
% textfraction ist der Bruchteil einer Seite,
% der fuer Text mindestens verfuegbar bleiben muss
% (ist null erlaubt??)
\setcounter{totalnumber}{4}
\renewcommand{\textfraction}{0.1}
%
% Splitten von Fussnoten erwschweren:
\interfootnotelinepenalty=2000
%
% kein zusaetzlicher Zwischenraum nach Satzende
\frenchspacing
%
% Numerierung der Untergliederungen:
\setcounter{secnumdepth}{3}
\setcounter{tocdepth}{3}
%
% neuer Zaehler fuer Uebersicht (Overview):
\newcounter{ovrdepth}
\setcounter{ovrdepth}{1}
%
% Satzspiegel vergroessern:
\topmargin-24mm
\textwidth146mm
\textheight236mm
%
% zusaetzlicher Seitenrand fuer beidseitigen Druck:
\oddsidemargin14mm
\evensidemargin0mm
%
% eigene Bezeichnungen, lassen sich beliebig aendern:
\def\contentsname{Inhaltsverzeichnis}
\def\chaptername{Kapitel}
\def\listfigurename{Abbildungen}
\def\listtablename{Tabellen}
\def\listsourcename{Programme}
\def\indexname{Sach- und Namensverzeichnis}
\def\figurename{Abb.}
\def\tablename{Tabelle}
\def\sourcename{Programm}
\def\ovrname{"Ubersicht}
%
% wegen Quotes (Gaensefuesschen) in source-Umgebung,
% darf auch sonst verwendet werden:
\def\qunormal{\catcode`\"=12}
\def\quactive{\catcode`\"=\active}
%
% in Unterschriften Umlaute ("a, "o, "u) ermoeglichen
% (der Programmtext wird mit qunormal geschrieben):
```

```
\def\caption{\quactive \refstepcounter\@captype
                           \@dblarg{\@caption\@captype}}
%
% neue Umgebung source fuer Programmtexte
% aehnlich figure, nicht floatend, Seitenumbruch erlaubt
% abgestimmt auf \verbinput{file} aus verbtext.sty
% qunormal + quactive sind enthalten
%
% neuer Zaehler, kapitelweise:
\newcounter{source}[chapter]
\def\thesource{\thechapter.\arabic{source}\ }
\def\fnum@source{{\sl\sourcename\ \thesource}}
%
% Filekennung List of Sources (main.los) und
% Overview (main.ovr):
\def\ext@source{los}
\def\ext@overview{ovr}
%
% neue Umgebung, Aufruf: \begin{source} .... \end{source},
% vor end{source} kommt caption:
\newenvironment{source}{\vskip 12pt \qunormal
      \def\@currentlabel{\p@source\thesource}
      \def\@captype{source}}{\quactive \vskip 12pt}
%
% neuer Befehl \listofsources analog \listoffigures
% zur Erzeugung eines Programmverzeichnisses:
\def\listofsources{\@restonecolfalse\if@twocolumn
                  \@restonecoltrue\onecolumn
 \fi\chapter*{\listsourcename\@mkboth
 {{\listsourcename}}{{\listsourcename}}}\@starttoc{los}
 \if@restonecol\twocolumn\fi}
\let\l@source\l@figure
%
% Kapitelkopf, rep12.sty. Siehe Kopka 2, S. 187 + 289:
% ohne Kapitel + Nummer:
\def\@makechapterhead#1{\vspace*{36pt}
 {\parindent 0pt \raggedright
 \LARGE \bf \thechapter \hspace{6mm} #1 \par
 \nobreak \vskip 10pt } }

\def\@makeschapterhead#1{\vspace*{36pt}
 {\parindent 0pt \raggedright
 \LARGE \bf #1 \par
 \nobreak \vskip 10pt } }
%
% aus rep12.sty; ergaenzt wegen source
% (sonst fehlt der vspace im Programmverzeichnis):
\def\@chapter[#1]#2{\ifnum \c@secnumdepth >\m@ne
 \refstepcounter{chapter}
 \typeout{\@chapapp\space\thechapter.}
 \addcontentsline{toc}{chapter}{\protect
 \numberline{\thechapter}#1}\else
 \addcontentsline{toc}{chapter}{#1}\fi
 \chaptermark{#1}
 \addtocontents{lof}{\protect\addvspace{10pt}}
```

```
 \addtocontents{los}{\protect\addvspace{10pt}}
 \addtocontents{lot}{\protect\addvspace{10pt}}
 \if@twocolumn \@topnewpage[\@makechapterhead{#2}]
 \else \@makechapterhead{#2}
 \@afterheading \fi}
%
% Inhaltsverzeichnis modifiziert:
\def\tableofcontents{\@restonecolfalse
             \if@twocolumn\@restonecoltrue\onecolumn
 \fi\chapter*{\contentsname
 \@mkboth{{\contentsname}}{\contentsname}}
 \@starttoc{toc}\if@restonecol\twocolumn\fi}
%
% weniger Luft in itemize (listI), aus rep12.sty:
\def\@listI{\leftmargin\leftmargini
 \parsep 4pt plus 2pt minus 1pt
 \topsep 6pt plus 4pt minus 4pt
 \itemsep 2pt plus 1pt minus 1pt}
%
% Seitennumerierung, aus report.sty uebernommen
% in main.tex erforderlich \pagestyle{uxheadings}
% nur fuer Option twoside passend:
\def\ps@uxheadings{\let\@mkboth\markboth
\def\@oddfoot{} \def\@evenfoot{}
\def\@evenhead{\small{{\rm \thepage} \hfil
  {\rm \leftmark}}}
\def\@oddhead{\small{{\rm \rightmark} \hfil
  {\rm \thepage}}}
\def\chaptermark##1{\markboth {\ifnum \c@secnumdepth
>\m@ne \thechapter \ \ \fi ##1}{}}
\def\sectionmark##1{\markright
{\ifnum \c@secnumdepth >\z@
 \thesection \ \ \fi ##1}}}
%
% Abb., Tabelle slanted, wie Programm:
\def\fnum@figure{{\sl\figurename\ \thefigure}}
\def\fnum@table{{\sl\tablename\ \thetable}}
%
% ersetze im Index see durch \it s.:
\def\see#1#2{{\it s.\/} #1}
%
% Indexvorspann, siehe Kopka 2, Seite 66:
\def\theindex#1{\@restonecoltrue\if@twocolumn
  \@restonecolfalse\fi
\columnseprule \z@
\columnsep 35pt\twocolumn[\@makeschapterhead
  {\indexname}#1]
 \@mkboth{\indexname}{\indexname}\thispagestyle
 {plain}\parindent\z@
 \parskip\z@ plus .3pt\relax\let\item\@idxitem}
%
% Uebersicht aus Inhaltsverzeichnis entwickelt:
%
% Wenn Inhaltsverzeichnis fertig,
% fgrep chapter main.toc > main.ovr
```

```
% Dann nochmals latex main.tex (Workaround)
\def\overview{\@restonecolfalse\if@twocolumn
  \@restonecoltrue\onecolumn
 \fi\chapter*{\vspace*{-18mm}\ovrname
 \@mkboth{{\ovrname}}{\ovrname}}
 \@starttoc{ovr}\if@restonecol\twocolumn\fi}
```

Quelle 2.23 : LaTeX-Datei alex.sty mit eigenen Makros, insbesondere der source-Umgebung

Das Manuskript wurde auf mehrere Dateien in je einem Unterverzeichnis pro Kapitel aufgeteilt, die mittels `\include` in die Hauptdatei `main.tex` eingebunden werden. Die Hauptdatei sieht so aus:

```
% Hauptfile main.tex fuer das gesamte Buch
% alex.sty erforderlich, 1996-11-30, fuer source usw.
% Files bigtabular.sty und verbtext.sty erforderlich
%
% Format: LaTeX
%
% Umgestellt auf LaTeX2e 1998-05-27 W. Alex
%
\NeedsTeXFormat{LaTeX2e}
\documentclass[12pt,twoside,a4paper]{report}
\usepackage{german,makeidx}
\usepackage{bigtabular,verbatim,verbtext,alex}
\pagestyle{uxheadings}
\sloppy
%
% Universitaetslogo einziehen
\input{unilogo}
%
% Trennhilfe
\input{hyphen}
%
% Indexfile main.idx erzeugen
\makeindex
%
% nach Bedarf:
% \includeonly{einleit/vorwort,einleit/umgang}
%
\begin{document}
\unitlength1.0mm
\include{verweisU}
\include{verweisC}
\begin{titlepage}
\begin{center}
\vspace*{20mm}
\hspace*{13mm} \unilogo{32}\\
\vspace*{24mm}
\hspace*{14mm} {\Huge UNIX, C und Internet\\}
\vspace*{10mm}
\hspace*{13mm} {\Large W. Alex und G. Bern"or\\}
\vspace*{6mm}
\hspace*{13mm} {\large unter Mitarbeit von B. Alex
```

```
                 und O. Koglin\\}
\vspace*{10mm}
\hspace*{13mm} {\Large 1999\\}
\vspace*{80mm}
\hspace*{13mm} {\Large Universit"at Karlsruhe\\}
\end{center}
\end{titlepage}
\include{einleit/copyright}
\pagenumbering{roman}
\setcounter{page}{5}
\include{einleit/vorwort}
\overview
\include{einleit/gebrauch}
\tableofcontents
\listoffigures
% \listoftables
\listofsources
\cleardoublepage
\pagenumbering{arabic}
\include{einleit/umgang}
\include{hardware/hardware}
\include{unix/unix}
\include{program/program}
\include{internet/internet}
\include{recht/recht}
\begin{appendix}
\include{anhang/anhang}
\end{appendix}
\cleardoublepage
\addcontentsline{toc}{chapter}{\indexname}
\printindex
\cleardoublepage
\setcounter{page}{0}
\include{einleit/rueckseite}
\end{document}
```

Quelle 2.24 : LaTeX-Hauptdatei main.tex für Manuskript

Das Prozentzeichen leitet Kommentar ein und wirkt bis zum Zeilenende. Der Befehl makeindex im Vorspann von main.tex führt zur Eintragung der im Text mit \index markierten Wörter samt ihren Seitenzahlen in eine Datei main.idx. In der Markierung der Wörter steckt Arbeit. Die idx-Datei übergibt man einem zu den LaTeX-Erweiterungen gehörenden Programm makeindex von PEHONG CHEN (nicht zu verwechseln mit dem zuvor genannten LaTeX-Kommando \makeindex). Das Programm erzeugt eine Datei namens main.ind, die ein bisschen editiert und durch den \printindex-Befehl am Ende des Manuskripts zum Dokument gebunden und ausgegeben wird.

Der DIN-Brief Zum Schreiben von Briefen mittels LaTeX verwendet man im deutschen Sprachraum zweckmäßig die Dokumentklasse dinbrief. Die Klassenbeschreibung heißt dinbrief.cls und liegt im Verzeichnis

/usr/share/texmf/tex/latex/dinbrief/, die zugehörige Dokumentation im Verzeichnis /usr/share/doc/texmf/latex/dinbrief/. Im persönlichen Briefverzeichnis legt man sich eine Vorlage (Muster, Schablone, Template) folgender Art an:

```
\documentclass[12pt]{dinbrief}
\usepackage{german, newcent}
% \input{briefkopf}  % verwende keinen ausser dem Absender
\address{{\bf Dr.-Ing. Wulf Alex}\\
         Latexweg 999\\
         D-12345 Unixhausen (Baden)\\
         Tel. 01234/567890\\
         Mobiltel. 0160/12345\\
         Email wulf.alex@mvm.uni-karlsruhe.de}
\backaddress{W. Alex, Latexweg 999, D-12345 Unixhausen}
% \signature{W. Alex}
\place{Unixhausen (Baden)}

\begin{document}

\begin{letter}{Superputer GmbH\\
               Debianstra\3e 111\\ \\
               {\bf 01234 Linuxberg}}

\subject{Open-Source-Treffen 2003}
\nowindowrules
\opening{Sehr geehrte Damen und Herren,}

Hier folgt der Text.

\closing{Mit freundlichem Gru\3\\Ihr}

% \ps{Postscriptum}
% \cc{Verteiler}
% \encl{Bericht 2003.1}
\end{letter}
\end{document}
```

Quelle 2.25 : LaTeX-Datei dinbrief.tex als Vorlage für Briefe

die man kopiert, um einen neuen Brief abzufassen. Nach dem Editieren der Kopie geht es wie mit jedem LaTeX-Dokument weiter. Effektivitätsbewusste Faulpelze legen sich ein Makefile an:

```
# Makefile fuer Briefe, W. Alex 2003-01-10

SRC = $@      # vordefinierte make-Variable
LATEX  = /usr/bin/latex
PLATEX = /usr/bin/pdflatex
DVIPS  = /usr/bin/dvips

.DEFAULT :    # vordefiniertes Target
        ${PLATEX} ${SRC}.tex
```

```
        ${LATEX} ${SRC}.tex
        ${DVIPS} -o ${SRC}.ps ${SRC}

clean :
        rm -f *.aux *.log *.dvi
```

Quelle 2.26 : Makefile zum LaTeXen von Briefen

und rufen make mit dem Namen der Briefdatei ohne Kennung als Argument auf.

Planung eines LaTeX-Projektes Der gesamte Text samt LaTeX-Befehlen kann in einer einzigen, großen Datei untergebracht werden. Bei umfangreichen Werken wie einer Diplomarbeit ist eine Aufteilung auf mehrere Unterverzeichnisse und Dateien zweckmäßiger. Die Aufteilung sollte man zu Beginn der Arbeit vornehmen, nicht erst wenn die Textdatei schon auf einige Megabyte angewachsen ist. In das Hauptverzeichnis des Projektes kommen:

- die Haupt-LaTeX-Datei (main.tex, diplom.tex, skriptum.tex oder ähnlich),
- das zugehörige Makefile (Makefile),
- die Datei mit den Trenn-Ausnahmen (hyphen.tex),
- die Datei mit den persönlichen Stil-Anpassungen (alex.sty),
- spezielle Hilfsdateien (unilogo.tex, extarticle.cls, size14.clo),
- je ein Unterverzeichnis für jedes Kapitel (chapter),
- je ein Unterverzeichnis für Bilder, Tabellen, Programmquellen, Overhead-Folien und dergleichen.

In jedes Kapitel-Unterverzeichnis kommen:

- je eine Kapiteldatei (unix.tex, internet.tex, CC++.tex),
- je eine Datei für jeden Abschnitt (section).

Eine Kapiteldatei beginnt mit \chapter{...} und endet mit \clearpage oder \cleardoublepage. Eine Abschnittsdatei beginnt mit \section{...} und endet offen. Die Kapiteldateien werden mittels \include{...} in die Haupt-LaTeX-Datei eingebunden, die Abschnittsdateien mittels \input{...} in die Kapiteldateien.

Jedes Kapitel, jeder Abschnitt, jeder Unterabschnitt, jedes Bild, jede Tabelle und jedes Programm bekommen von vornherein mittels \label{...} ein Label, auf das man sich mit \ref{...} oder \pageref{...} im Text beziehen kann. Zweckmäßig beginnt man den Labelnamen mit einem Hinweis auf das gelabelte Objekt: \label{chap:...}, \label{sec:...} oder \label{fig:...}. Ebenso sind vor jedem Absatz die ins Sachregister aufzunehmenden Stichwörter mittels \index{...} zu nennen, pro Zeile ein Eintrag wegen besserer Übersicht. Diese Verzierungen nachträglich anzubringen, kostet unnötig Zeit. Eine Übersicht der verwendeten Label zieht man sich aus der .aux-Datei heraus:

```
grep newlabel skriptum.aux | tr -d '\\newlabel' |
sort > skriptum.lab
```

Das Makefile sieht dann ungefähr so aus:

```
all    : diplom schirm papier

diplom :
        rm -f *.dvi *.aux *.log *.idx *.ilg *.ind
        rm -f *.toc *.lof *.lot
        latex diplom
        latex diplom
        latex diplom

schirm :
        xdvi diplom

papier :
        dvips -o diplom.ps diplom

clean  :
        rm -f *.dvi *.aux *.log *.idx *.ilg *.ind
        rm -f *.toc *.lof *.lot
```

Drei LaTeX-Durchgänge wegen Referenzen und Inhaltsverzeichnis.

Technische Dokumente enthalten oft Illustrationen. Hierzu benötigt man folgende Werkzeuge (in Linux-Distributionen üblicherweise enthalten):

- zum Zeichnen von Diagrammen aus Formeln oder Wertetabellen `gnuplot(1)`,
- zum Zeichnen von Schemata und dergleichen `xfig(1)`,
- zum Scannen und Bearbeiten von Fotos `gimp(1)`.

Diese Werkzeuge werden im Abschnitt 2.9 *L'atelier graphique* auf Seite 223 erläutert.

2.7.13 Texinfo

Dem **Texinfo-System** (gesprochen *tekinfo*) aus der GNU-Sammlung begegnen wir oft im Zusammenhang mit Online-Hilfen wie den `man`-Seiten. Zum Beispiel heißt es auf der `man`-Seite zum Kommando `dvips(1)`, dass sie veraltet sei und man statt ihrer die zugehörige Texinfo-Dokumentation lesen möge. Texinfo erlaubt, Dokumente zu **strukturieren** und wahlweise auf **Bildschirm** oder **Papier** auszugeben. Auch eigene Dokumente lassen sich mit dem System verarbeiten.

Als erstes beschaffen wir uns das Paket, sofern es nicht schon eingerichtet ist, per Anonymous FTP von `ftp.gnu.org` oder einem näher gelegenen Mirror. Das Paket ist wie üblich ein komprimiertes Archiv und muss daher in einem beliebigen Verzeichnis mittels gunzip(1) auf Trinkstärke verdünnt und dann mit tar -xf ausgepackt werden. Anschließend findet man ein Unterverzeichnis texinfo-* vor und wechselt hinein. Optimisten rufen dann sofort das Kommando ./configure auf, nach dessen hoffentlich erfolgreichem Abschluss make. Geht alles gut, kann man make check ausprobieren. Das Einrichten wird vom Benutzer root (wegen der Zugriffsrechte der Systemverzeichnisse) mit make install vorgenommen. In unserem Fall lagen danach die Programme samt Zubehör unter /usr/local. Mit make clean und make distclean wird aufgeräumt.

Das Kommando zum **Lesen** lautet /usr/local/bin/info. Man kann es zunächst einmal anstelle man aufrufen, beispielsweise als info ls. Dann bekommt man mit den Möglichkeiten von Texinfo wie Vorwärts- und Rückwärtsblättern die gewohnte man-Seite angezeigt. Mit dem Kommando info info erscheint eine kurze, strukturierte Einführung auf dem Bildschirm, die man durcharbeiten sollte. Bei Schwierigkeiten versuchen Sie es mit einer Option info -f info.

Ein Texinfo-Quelltext wird mit einem beliebigen Editor geschrieben. Der GNU-emacs kennt einen Texinfo-Modus, der die Arbeit erleichtert. Metazeichen sind der Klammeraffe und die beiden geschweiften Klammern. Tabs verursachen Probleme bei der Formatierung, ansonsten dürfen alle druckbaren ASCII-Zeichen im Text vorkommen. Die Formatierkommandos lauten anders als bei nroff(1), TeX oder LaTeX, obwohl sie zum großen Teil dieselben Aufgaben haben. Man darf auch nicht vergessen, dass man für zwei verschiedene Leserkreise schreibt: die Buchleser und die Bildschirmleser. Kurze Passagen im Text lassen sich so markieren, dass sie sich nur an einen der beiden Leserkreise wenden, aber zwei fast gänzlich unterschiedliche Dokumente abzufassen, wäre ein Rückfall in die Zeit vor Texinfo. Die Referenz entnimmt man am besten dem Archiv oder dem Netz, hier nur ein kurzes Beispiel:

```
(noch zu schreiben)
```

Zur Erzeugung der Papierausgabe ist die Texinfo-Quelldatei durch das TeX-System samt den Makros Texinfo.tex zu schicken, die DVI-Datei dann wie unter TeX oder LaTeX gewohnt durch dvips(1), um eine druckbare PostScript-Datei zu erhalten. Die online lesbare info-Datei wird von dem Programm makeinfo(1) erstellt, das zum Texinfo-Paket gehört. Zweckmäßig fasst man die Kommandos in einem Makefile zusammen:

```
all :  sample.ps sample.info

sample.ps : sample.dvi
       dvips -o sample.ps sample

sample.dvi : sample.texinfo
       tex sample.texinfo

sample.info : sample.texinfo
       makeinfo sample
```

```
clean :
        rm *.aux *.toc *.log
```

und ruft nach dem Editieren nur noch make(1) auf. Im Netz finden sich Konverter von Texinfo nach HTML, nroff, IPF und RTF.

2.7.14 Hypertext

2.7.14.1 Was ist Hypertext?

Bei **Hypertext** und der **Hypertext Markup Language** (HTML) geht es auch um das Formatieren von Texten oder allgemeiner von Hypermedia-Dokumenten, aber es kommt noch etwas hinzu. Hypertexte enthalten **Hyperlinks**. Das sind Verweise, die elektronisch auswertbar sind, so dass man ohne Suchen und Blättern zu anderen Hypertexten weitergeführt wird. Man kann die Hyperlinks als aktive Querverweise bezeichnen. Auf Papier erfüllen Fußnoten, Literatursammlungen, Register, Querverweise und Konkordanzen einen ähnlichen Zweck, ohne elektronischen Komfort allerdings. Im ersten Kapitel war von WALLENSTEIN die Rede. Von diesem Stichwort könnten Verweise auf das Schauspiel von FRIEDRICH SCHILLER, die Werke von ALFRED DÖBLIN, RICARDA HUCH, PETER ENGLUND oder auf die Biografie von GOLO MANN führen, die die jeweiligen Texte auf den Bildschirm bringen, in SCHILLERs Fall sogar mit einem Film. In den jeweiligen Werken wären wieder Verweise enthalten, die auf Essays zur Reichsidee oder zur Rolle Böhmens in Europa lenken. Leseratten würden vielleicht auf dem Alexanderplatz in Berlin landen oder bei einem anderen Vertreter der schreibfreudigen Familie MANN. Von dort könnte es nach Frankreich, Indien, Ägypten, in die USA oder die Schweiz weitergehen. Vielleicht findet man auch Bemerkungen zum Verhältnis zwischen Literatur und Politik. Beim Nachschlagen in Enzyklopädien gerät man manchmal ins ziellose Schmökern. Mit Hypertext ist das noch viel, viel schlimmer. So ist jede Information eingebettet in ein Gespinst oder Netz von Beziehungen zu anderen Informationen, und wir kommen zum World Wide Web. Davon später mehr.

2.7.14.2 Hypertext Markup Language (HTML)

Zum Schreiben von Hypertext-Dokumenten ist die **Hypertext Markup Language** (HTML) entworfen worden, gegenwärtig in der Version 4 im Netz. Zum Lesen von Hypertexten braucht man HTML-Brauser wie `netscape`, `mozilla`, `galeon`, `opera`, `mosaic`, `konqueror` oder den Internet-Explorer von Microsoft. Leider halten sich die wenigsten Brauser an den gültigen HTML-Standard. Sie erkennen nicht alle standardkonformen HTML-Konstrukte und bringen eigene (proprietäre) Vorstellungen mit. Wenn man HTML-Dokumente (Webseiten) für die Öffentlichkeit schreibt, sollte man daher seine Erzeugnisse mit verschiedenen Brausern betrachten und überdies mit mehreren Werkzeugen testen.

Ein einfaches HTML-Dokument, das nicht alle Möglichkeiten von HTML ausreizt, ist schnell geschrieben:

```
<HTML>

<HEAD>
<TITLE>Institut fuer Hoeheres WWW-Wesen</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff">

<IMG SRC="http:../logowww.jpg" alt="">

<H3>
Fakult&auml;t f&uuml;r Internetwesen
</H3>

<HR>

Geb&auml;ude 30.70 <BR>
Telefon +49 721 608 2404 <BR>

<H4>
Leiter der Verwaltung
</H4>
Dipl.-Ing. Schorsch Meier

<H4>
Werkstattleiter
</H4>
Alois Hingerl

<HR>
Zur <A HREF="http://www.uni-karlsruhe.de/Uni/">
                      Universit&auml;t </A>
<HR>

http://www.ciw.uni-karlsruhe.de/hwwww/index.html <BR>
J&uuml;ngste &Auml;nderung 2002-11-25
<A HREF="mailto:webmaster@mvm.uni-karlsruhe.de">
                      webmaster@mvm.uni-karlsruhe.de
</A>

</BODY>

</HTML>
```

Das ganze Dokument wird durch `<HTML>` und `</HTML>` eingerahmt. In seinem Inneren finden sich die beiden Teile `<HEAD>` und `<BODY>`. Die Formatanweisungen `<H3>` usw. markieren Überschriften (Header). Sonderzeichen werden entweder durch eine Umschreibung (entity) (+`ä`) oder durch die Nummer im Latin-1-Zeichensatz (`ä`) dargestellt. `<BR>` ist ein erzwungener Zeilenumbruch (break), `<HR>` eine waagrechte Linie (horizontal ruler). Am Ende sollte jedes Dokument seinen **Uniform Resource Locator** (URL) enthalten, damit man es wiederfindet, sowie das Datum der jüngsten Änderung und die Email-Anschrift des Verantwortlichen.

Vor einem verbreiteten Fehler – gerade bei Anfängern – sei gewarnt. Die HTML-Kommandos beschreiben eine Struktur, nicht das Aussehen. Viele Kommandos lassen sich missbrauchen, was in manchen Zusammenhängen gut geht, in anderen nicht. Das `<BR>`-Element erzeugt einen Zeilenumbruch, mehrere aufeinander folgende `<BR>`-Elemente also Leerzeilen. Diese kann ich auch mittels mehrerer `<P>`-Elemente hervorrufen, die eigentlich zum Einrahmen von Absätzen oder Paragrafen gedacht sind. Abgesehen von der unterschiedlichen Syntax (Attribute etc.) kann ein Brauser unter einem Paragrafen etwas anderes verstehen als einen Zeilenwechsel, man weiß das nie sicher. Beliebt ist der Missbrauch von Tabellen zur Darstellung von mehrspaltigem Text. Spätestens beim Lesen des Dokumentes mit einem zeilenweise arbeitenden Screen-Reader geht das voll in die Hose. Man nehme also immer das HTML-Element, das den eigenen Wunsch logisch genau wiedergibt, und nicht eines, das unter den gegenwärtigen, zufälligen Umständen den gewünschten Effekt auf dem Bildschirm erzeugt.

HTML-Dateien tragen die Kennung `.html` oder `.htm`, letztere in der auf Dateinamen nach dem 8.3-Muster beschränkten Welt. Dokumente können Anweisungen an den WWW-Server enthalten, die dieser bei einer Anfrage nach dem Dokument vor dem Senden der Antwort ausführt. Diese **Server Parsed Documents** oder **Server Side Includes** tragen oft statt der Kennung `.html` die Kennung `.shtml`, aber das ist nicht zwingend. Beispiele sind Zähler oder bei jeder Anfrage aktualisierte Tabellen. Im Gegensatz dazu stehen Dokumente mit Anweisungen an den Brauser. Im Netz sind mehrere Kurzanleitungen und die ausführliche Spezifikation von HTML verfügbar.

Texte, Bilder und Tabellen werden gut unterstützt. Zum Schreiben von mathematischen Formeln sind zwar im HTML-Standard Wege vorgesehen, die sich an LaTeX anlehnen, die gängigen HTML-Brauser geben jedoch die Formeln nicht wieder, so dass manche Autoren getrennte LaTeX- und HTML-Fassungen ihrer Manuskripte herstellen (müssen). Aus dieser unbefriedigenden Lage ist ein Ausweg in Sicht, die **Structured Generalized Markup Language** (SGML). Man verfasst einen Text mit einer Formatierungssprache (Markup Language) eigener Wahl und vielleicht sogar eigener Zucht. In einem zweiten Dokument namens **Document Type Definition** (DTD) beschreibt man dazu in SGML, was die Konstrukte der Markup Language bedeuten. SGML ist also eine Sprache zur Beschreibung einer Sprache. HTML ist eine Sprache, die mittels SGML beschrieben werden kann. Ein SGML-Brauser erzeugt aus Text und DTD nach Wunsch ASCII-, LaTeX- oder HTML-Vorlagen für ihre jeweiligen Zwecke. Das funktioniert in Ansätzen bereits, Einzelheiten wie immer im Netz, das bei solchen Entwicklungen aktueller ist als Papier.

SGML ist für manche Aufgaben ein zu aufwendiges Werkzeug. Andererseits möchte man gelegentlich mehr machen, als HTML erlaubt. Hier hilft die **Extensible Markup Language** (XML) mit Möglichkeiten, eigene Tags (Formatieranweisungen) zu definieren. Man könnte XML als eine abgespeckte SGML bezeichnen. Ehe man an XML oder SGML geht, sollte man ein einwandfreies HTML-Dokument und Stylesheets schreiben können, was nicht bedeutet, alle Feinheiten von HTML zu beherrschen.

2.7.15 PostScript und PDF

2.7.15.1 PostScript

(kommt demnächst, oder später)

2.7.15.2 Portable Document Format (PDF)

Das **Portable Document Format** (PDF) von Adobe ist eine Erweiterung von PostScript. PostScript-Dateien lassen sich daher einfach mittels eines Programmes wie dem Adobe Distiller nach PDF umwandeln. PDF-Werkzeuge sind für viele Systeme verfügbar, PDF-Dokumente sind zwischen diesen Systemen austauschbar.

Das Portable Document Format zielt ab auf die Wiedergabe der Dokumente (Texte, Zeichnungen, Fotos) auf dem Bildschirm, ähnlich wie HTML. Im Gegensatz zu LaTeX- oder HTML-Quellen und in Übereinstimmung mit PostScript-Dateien wird in PDF-Dateien das Aussehen der Dokumente genau festgelegt. Der Autor bestimmt, was der Leser sieht. Die Erweiterung gegenüber PostScript besteht vor allem darin, dass PDF-Dokumente ebenso wie HTML-Seiten aktiv sein können; es gibt Hyperlinks, Eingabefelder und bewegte Grafiken. Vereinfacht gesagt ist PDF eine Mischung aus PostScript und HTML. Zum Lesen lassen sich WWW-Brauser oder spezielle Leseprogramme wie dem Adobe Reader `acroread(1)` verwenden. Beim Drucken auf Papier geht die Aktivität natürlich verloren, je nach Drucker auch die Farbe.

PDF-Dokumente werden entweder aus anderen Formaten oder durch Einscannen von gedruckten Vorlagen erzeugt. Hierzu gibt es Werkzeuge. Man schreibt also nicht eine PDF-Quelle so wie man ein LaTeX-Manuskript schreibt. Ein PDF-Dokument wird mit dem Werkzeug Acrobat Exchange weiter bearbeitet.

2.7.16 Computer Aided Writing

Die Verwendung von Rechnern und Programmen wirkt sich auf die Technik und das Ergebnis des Schreibens aus, insgesamt hoffentlich positiv. LaTeX führt typischerweise zu Erzeugnissen, die stark gegliedert sind und ein entsprechend umfangreiches Inhaltsverzeichnis aufweisen, aber wenig Abbildungen und nicht immer ein Sachregister haben. Von der Aufgabe her ist das selten gerechtfertigt, aber das Programm erleichtert nun einmal das eine und erschwert das andere. Manuskripte, die auf einem WYSIWYG-System hergestellt worden sind, zeichnen sich häufig durch eine Vielfalt von grafischen Spielereien aus. Eine gute Typografie drängt sich nicht vor den Inhalt, sie fällt nicht auf, der Leser bemerkt sie nicht.

Neben diesen Äußerlichkeiten weisen Computertexte eine tiefer gehende Eigenart auf. In einem guten Text beziehen sich Sätze und Absätze auf vorangegangene oder kommende Teile, sie bilden ein Kette, die nicht ohne weiteres unterbrochen werden darf. Der Rechner erleichtert das Verschieben von Textteilen und das voneinander unabhängige Arbeiten an verschiedenen Stellen des Textes. Man beginnt mit dem Schreiben nicht immer am Anfang des Manuskriptes, sondern dort, wo man den meisten Stoff bereit hat oder wo das Bedürfnis am dringendsten scheint. Von einer Kette, in der jedes Glied mit dem vorangehenden verbunden ist, bleibt nicht viel übrig, die Absätze oder Sätze stehen beziehungslos nebeneinander, oder es entstehen falsche Bezüge, manchmal auch ungewollte Wiederholungen. Während man beim Programmieren aus guten Gründen versucht, die Module oder Objekte eines Programms möglichst unabhängig voneinander zu gestalten und nur über einfache, genau definierte Schnittstellen miteinander zu verknüpfen, ist dieses Vorgehen bei einem Text selten der beste Weg. Stellen Sie sich ein Drama oder einen Roman vor, dessen Abschnitte in beliebiger Reihenfolge gelesen werden können. Dass manche Leute Bücher vom Ende her lesen, ist eine andere Geschichte.

Bei Hypertext-Dokumenten, wie sie im World Wide Web stehen, ist der Aufbau aus einer Vielzahl voneinander unabhängiger Bausteine, die in beliebiger Reihenfolge betrachtet werden können, noch ausgeprägter. Das führt zu anderen Arten des Schreibens und Lesens, die nicht schlechter zu sein brauchen als die traditionellen. Hypertext ermöglicht eine Strukturierung eines Textes, die Papier nicht bieten kann und die der Struktur unseres Wissens vielleicht besser entspricht. Hypertexte gleichen eher einem Gewebe als einer Kette. Wie ein Roman oder ein Gedicht in Hypertext aussehen könnten, ist noch nicht erprobt. Auf jeden Fall lässt sich Hypertext nicht vorlesen.

Heute schafft ein Autor am Schreibtisch buchähnliche oder eigenständige Erzeugnisse, an deren Zustandekommen früher mehrere geachtete Berufe beteiligt waren und entsprechend Zeit benötigt haben. Bücher werden nach reproduktionsreifen (camera-ready) Vorlagen gedruckt, die aus dem Rechner und dem Laser-Drucker stammen. Auch die Verteilung von Wissen geht über elektronische Medien einfacher und schneller als auf dem hergebrachten Weg. Dazu kommen bewegte Grafiken, Sound, Interaktivität und Print-on-Demand. Ergänzungen zum Buch wie unsere WWW-Seite `www.ciw.uni-karlsruhe.de/technik.html`, Aktualisierungen und die Rückkopplung vom Leser zum Autor sind im Netz eine Kleinigkeit. Das ganze technische Drumherum um ein Dokument – Text oder Manuskript trifft ja nicht ganz zu – wird als *Document Engineering* bezeichnet. GOETHEs ECKERMANN wäre heute ein Dokumenten-Ingenieur.

2.7.17 Weitere Werkzeuge (grep, diff, sort usw.)

Für einzelne Aufgaben der Textverarbeitung gibt es Spezialwerkzeuge in UNIX. Häufig gebraucht werden `grep(1)` (= global regular expression print), `egrep(1)` und `fgrep(1)`. Sie durchsuchen Textdateien nach Zeichenmustern. Ein einfacher Fall: suche in der Datei `telefon` nach einer Zeile, die das Zeichenmuster `alex` enthält. Das Kommando lautet

```
grep -i alex telefon
```

Die Option `-i` weist `grep(1)` an, keinen Unterschied zwischen Groß- und Kleinbuchstaben zu machen. Die gleiche Suche leistet auch ein Editor wie der `vi(1)`, nur ist der ein zu umfangreiches Werkzeug für diesen Zweck. Unter DOS heißt das entsprechende Werkzeug `find`, das nicht mit UNIX-`find(1)` verwechselt werden darf.

Für unsere Anlage habe ich unter Verwendung von `grep(1)` ein Shellskript namens `it` (= info Telefon) geschrieben, das erst in einem privaten, dann in einem öffentlichen Telefonverzeichnis sucht:

```
grep -s $* $HOME/inform/telefon /mnt/inform/telefon |
sed -e "s/^\/[^:]*://g"
```

Quelle 2.27 : Shellskript zum Suchen in einem Telefonverzeichnis

Um in der Datei `fluids` nach dem String `dunkles Hefe-Weizen` zu suchen, auch mit großem Anfangsbuchstaben und in allen Beugungsformen, gibt es folgende Wege:

```
grep unkle fluids | grep Hefe-Weiz
grep '[Dd]unkle[mns]* Hefe-Weizen' fluids
```

Der erste Weg könnte auch seltsame Kombinationen liefern, die unwahrscheinlich sind, erfordert aber keine vertieften Kenntnisse von regulären Ausdrücken. Der zweite Weg macht von letzteren Gebrauch, engt die Auswahl ein und wäre in unbeaufsichtigt laufenden Shellskripts vorzuziehen.

`grep(1)` ist ursprünglich nicht rekursiv, das heißt es geht nicht in Unterverzeichnisse hinein. Nimmt man `find(1)` zur Hilfe, das rekursiv arbeitet, so lässt sich auch rekursiv greppen:

```
find . -print | xargs grep suchstring
```

Das Kommando `xargs(1)` hängt die Ausgabe von `find(1)` an die Argumentliste von `grep(1)` an und führt es aus.

Mittels `diff(1)` werden die alte und die neue Version einer Datei miteinander verglichen. Bei entsprechendem Aufruf wird eine dritte Datei erzeugt, die dem Editor `ed(1)` als Kommandoskript (ed-Skript) übergeben werden kann, so dass dieser aus der alten Version die neue erzeugt. Gebräuchlich zum Aktualisieren von Programmquellen. Schreiben Sie sich eine kleine Textdatei `alt`, stellen Sie eine Kopie namens `neu` davon her, verändern Sie diese und rufen Sie dann `diff(1)` auf:

```
diff -e alt neu > edskript
```

Fügen Sie mit einem beliebigen Editor am Ende des edskript zwei Zeilen mit den ed(1)-Kommandos w und q (write und quit) hinzu. Dann rufen Sie den Editor ed(1) mit dem Kommandoskript auf:

```
ed - alt < edskript
```

Anschließend vergleichen Sie mit dem simplen Kommando cmp(1) die beiden Versionen alt und neu auf Unterschiede:

```
cmp alt neu
```

Durch den ed(1)-Aufruf sollte die alte Version genau in die neue Version überführt worden sein, cmp(1) meldet nichts.

Weitere Werkzeuge, deren Syntax man im Handbuch, Sektion 1 nachlesen muss, sollen hier nur tabellarisch aufgeführt werden:

- bfs big file scanner, untersucht große Textdateien auf Muster
- col filtert Backspaces und Reverse Line Feeds heraus
- comm common, vergleicht zwei sortierte Dateien auf gemeinsame Zeilen
- cut schneidet Spalten aus Tabellen heraus
- diff3 vergleicht drei Dateien
- expand/unexpand wandelt Tabs in Spaces um und umgekehrt
- fold faltet lange Zeilen (bricht Zeilen um)
- hyphen findet Zeilen, die mit einem Trennstrich enden
- nl number lines, numeriert Zeilen
- paste mischt Dateien zeilenweise
- ptx permuted index, erzeugt ein Sachregister
- rev reverse, mu nelieZ trhek
- rmnl remove newlines, entfernt leere Zeilen
- rmtb remove trailing blanks (lokale Erfindung)
- sort sortiert zeilenweise, nützlich
- spell prüft amerikanische Rechtschreibung[38]
- split spaltet eine Datei in gleich große Teile
- ssp entfernt mehrfache leere Zeilen
- tr translate, ersetzt Zeichen
- uniq findet wiederholte Zeilen in einer sortierten Datei
- vis zeigt eine Datei an, die unsichtbare Zeichen enthält

[38] Es gibt eine internationale Fassung ispell im GNU-Projekt.

- `wc` word counter, zählt Zeichen, Wörter, Zeilen

Die Liste lässt sich durch eigene Werkzeuge beliebig erweitern. Das können Programme oder Shellskripte sein. Hier ein Beispiel zur Beantwortung einer zunächst anspruchsvoll erscheinenden Fragestellung mit einfachen Mitteln. Ein Sachtext soll nicht unnötig schwierig zu lesen sein, die Sachzusammenhänge sind schwierig genug. Ein grobes Ma:s für die **Lesbarkeit** eines Textes ist die mittlere Satzlänge. Erfahrungsgemäß sind Werte von zehn bis zwölf Wörtern pro Satz für deutsche Texte zu empfehlen. Wie kann eine Pipe aus UNIX-Werkzeugen diesen Wert ermitteln? Schauen wir uns das Vorwort an. Als erstes müssen die LaTeX-Konstrukte herausgeworfen werden. Hierfür gibt es ein Programm `delatex`, allerdings nicht standardmäßig unter UNIX. Dann sollten Leerzeilen entfernt werden – Werkzeug `rmnl(1)` – sowie einige Satzzeichen – Werkzeug `tr -d`. Schließlich muss jeder Satz in einer eigenen Zeile stehen. Wir ersetzen also alle Linefeed-Zeichen (ASCII-Nr. 10, oktal 12) durch Leerzeichen und danach alle Punkte durch Linefeeds. Ein kleiner Fehler entsteht dadurch, dass Punkte nicht nur ein Satzende markieren, aber bei einem durchschnittlichen Text ist dieser Fehler gering. Schicken wir den so aufbereiteten Text durch das Werkzeug `wc(1)`, so erhalten wir die Anzahl der Zeilen gleich Anzahl der Sätze, die Anzahl der Wörter (wobei ein Wort ein maximaler String begrenzt durch Leerzeichen, Tabs oder Linefeeds ist) und die Anzahl der Zeichen im Text. Die Pipe sieht so aus:

```
cat textfile | rmnl | tr -d '[0-9],;"()' |
tr '\012' '\040' | tr '.' '\012' | wc
```

Quelle 2.28 : Shellskript zur Stilanalyse

Die Anzahl der Wörter geteilt durch die Anzahl der Sätze liefert die mittlere Satzlänge. Die Anzahl der Zeichen durch die Anzahl der Wörter ergibt die mittlere Wortlänge, infolge der Leerzeichen am Wortende erhöht um 1. Auch das ist ein Stilmerkmal. Die Ergebnisse für das Vorwort (ältere Fassung) sind 29 Sätze, 417 Wörter und 3004 Zeichen, also eine mittlere Satzlänge von 14,4 Wörtern pro Satz und eine mittlere Wortlänge (ohne Leerzeichen) von 6,2 Zeichen pro Wort. Zählt man von Hand nach, kommt man auf 24 Sätze. Die Punkte bei den Zahlenangaben verursachen den Fehler. Man müsste das Satzende genauer definieren. Die erste Verbesserung des Verfahrens wäre, nicht nur die Mittelwerte, sondern auch die Streuungen zu bestimmen. Hierzu wäre der `awk(1)` zu bemühen oder gleich ein C-Programm zu schreiben. Das Programm liefert nur Zahlen; ihre Bedeutung erhalten sie, indem man sie zu Erfahrungswerten in Beziehung setzt. Soweit sich Stil durch Zahlen kennzeichnen lässt, hilft der Rechner; wenn das Verständnis von Wörtern, Sätzen oder noch höheren Einheiten verlangt wird, ist er überfordert.

Es soll ein UNIX-Kommando `style(1)` geben, das den Stil eines englischen Textes untersucht und Verbesserungen vorschlägt. Dagegen ist das Kommando `diplom(1)`, das nach Eingabe eines Themas und einer Seitenanzahl eine Diplomarbeit schreibt – mit `spell(1)` und `style(1)` geprüft – noch nicht ausgereift.

2.7.18 Textdateien aus anderen Welten (DOS, Mac)

In UNIX-Textdateien wird der Zeilenwechsel (line break) durch ein newline-Zeichen `\n` markiert, hinter dem das ASCII-Zeichen Nr. 10 (LF, Line feed) steckt, das auch durch die Tastenkombination `control-j` eingegeben wird. In DOS-Textdateien wird ein Zeilenwechsel durch das Zeichenpaar Carriage return – Line feed (CR LF, ASCII Nr. 13 und 10, `control-m` und `control-j`) markiert, das Dateiende durch das ASCII-Zeichen Nr. 26, `control-z`. Auf Macs ist die dritte Möglichkeit verwirklicht, das Zeichen Carriage return (CR, ASCII Nr. 13) allein veranlasst den Sprung an den Anfang der nächsten Zeile.

Auf einer UNIX-Maschine lassen sich die störenden Carriage returns (oktal 15) der DOS-Texte leicht durch folgenden Aufruf entfernen:

```
tr -d "\015" < file1 > file2
```

Der `vi(1)` oder `sed(1)` können das natürlich auch, ebenso ein einfaches C-Programm.

Wenn Ihr Text auf einem Bildschirm oder Drucker treppenförmig dargestellt wird – nach rechts fallend – erwartet das Gerät einen Text nach Art von DOS mit CR und LF, der Text enthält jedoch nach Art von UNIX nur LF als Zeilenende. In einigen Fällen lässt sich das Gerät entsprechend konfigurieren, auf jeden Fall kann man den Text entsprechend ergänzen. Wenn umgekehrt auf dem Bildschirm kein Text zu sehen ist, erwartet das Ausgabeprogramm einen UNIX-Text ohne CR, die Textdatei stammt jedoch aus der DOS-Welt mit CR und LF. Jede Zeile wird geschrieben und gleich wieder durch den Rücksprung an den Zeilenanfang gelöscht. Viele UNIX-Pager berücksichtigen das und geben das CR nicht weiter. Auf Druckern kann sich dieses Missverständnis durch Verdoppelung des Zeilenabstandes äußern. Kein Problem, nur lästig.

Neben diesen, dem Benutzer auffallenden Unterschieden gibt es auch viele Möglichkeiten der rechnerinternen Darstellung der Zeichen. Damit in einem Netz verschiedene Rechner Texte austauschen können, hat man sich im Internet – im RFC 854 Telnet – darauf geeinigt, im Netz die Zeichen gemäß Network-Virtual-Terminal-ASCII (NVT-ASCII) darzustellen und die Umsetzung auf die lokale Darstellung dem Sender bzw. dem Empfänger zu überlassen. Dabei werden die 128 Zeichen des US-ASCII-Zeichensatzes durch ein Byte (Oktett) dargestellt, dessen oberstes Bit (most significant bit) null ist.

2.7.19 Druckerausgabe (lp, lpr, CUPS)

Drucker sind entweder über eine Schnittstelle (parallel, seriell, USB) an einen Rechner angeschlossen oder stehen als selbständige Geräte (Knoten) im Netz. Im ersten Fall kann man in den Datenstrom, der zum Drucker geht, zentral eingreifen. Im zweiten Fall kann man nur hoffen, dass die Netzteilnehmer keinen Unsinn zum Drucker schicken.

Auf einer UNIX-Anlage arbeiten in der Regel mehrere Benutzer gleichzeitig, aber auch ein einzelner Benutzer kann kurz nacheinander mehrere Textdateien zum Drucker schicken. Damit es nicht zu einem Durcheinander kommt, sorgt ein Dämon, der **Line Printer Spooler**, dafür, dass sich die **Druckaufträge** (requests) in eine Warteschlange einreihen und der Reihe nach zu dem jeweils verlangten Drucker geschickt werden. Die Schreibberechtigung auf `/dev/printer` hat nur der Dämon, nicht der Benutzer. Auch in anderen Zusammenhängen (Email) spricht man von *spoolen*, wenn Aufträge oder Dateien in Warteschlangen eingereiht werden.

Der Dämon sorgt auch dafür, dass die Drucker richtig eingestellt werden, beispielsweise auf Querformat oder deutschen Zeichensatz. Auf manchen Systemen findet sich eine Datei `/etc/printcap` mit einer Beschreibung der Drucker, ähnlich wie in `/usr/lib/terminfo` oder `/etc/termcap` die Terminals beschrieben werden.

Das Kommando zum Drucken[39] lautet:

```
lp -dlp1 textfile
lpr -Plp2 textfile
```

Die erste Form stammt aus der System-V-Welt, die zweite aus der BSD-Welt. Die Option wählt in beiden Fällen einen bestimmten Drucker aus, fehlt sie, wird der Default-Drucker genommen. Die Kommandos kennen weitere Optionen, die mittels `man` nachzulesen sind. Mit dem Kommando `lpstat(1)` oder `lpq(1)` schaut man sich den Spoolerstatus an, Optionen per `man(1)` ermitteln. Mit `cancel request-id` oder `lprm(1)` löscht man einen Druckauftrag (nicht mit `kill(1)`), auch fremde. Der Auftraggeber erhält eine Nachricht, wer seinen Auftrag gelöscht hat.

In neuerer Zeit ist das **Common Unix Printing System** (CUPS) hinzugekommen, das vor allem das Drucken im Netz unterstützt. Es versteht von Haus aus die Druckkommandos aus dem System V, kann aber auf die BSD-Druckkommandos erweitert werden.

Laserdrucker gehobener Preisklassen bieten heute eine Möglichkeit zum unmittelbaren Anschluss an ein Netz (Ethernet). Sie erhalten dann eine eigene IP-Adresse im Internet und einen Namen wie ein Rechner. Der Vorteil ist die höhere Geschwindigkeit bei der Übertragung der Daten, der Nachteil liegt darin, dass man die Daten nicht unmittelbar vor dem Drucken durch ein Skript filtern kann, das beispielsweise die Ausgabe von kompilierten Programmen oder unsinnigen Steuerzeichen abfängt. Aus mancherlei Gründen gehören Druckerstörungen in einem heterogenen Netz leider zum täglichen Brot der Verwalter.

2.7.20 Begriffe Writer's Workbench

Folgende Begriffe sollten klarer geworden sein:

- Font
- Format, Formatierprogramm
- Hypertext, Hypertext Markup Language (HTML)

[39] *Welches Druckkommando haben wir heute?* ist eine in UNIX-Umgebungen häufig gestellte Frage.

- PostScript, Portabel Document Format (pdf)
- Regulärer Ausdruck
- (Text-)Editor
- Verschlüsseln, symmetrisches und unsymmetrisches V.
- Zeichensatz, ASCII, Latin-1, Unicode

Folgende Kommandos sollten beherrscht werden:

- `vi` (oder `joe` oder `emacs`)
- `grep`
- `sort`
- Anfänge von `awk`
- Anfänge von `sed`
- `lp` oder]tt lpr

2.7.21 Memo Writer's Workbench

- Zeichen werden im Rechner durch Nummern (Bitfolgen) dargestellt. Die Zuordnung Zeichen-Nummer findet sich in Zeichensatz-Tabellen wie US-ASCII. Die Tabelle legt damit auch fest, welche Zeichen überhaupt verfügbar sind, nicht jedoch wie sie aussehen. Werden bei Ein- und Ausgabe unterschiedliche Tabellen verwendet, gibt es Zeichensalat.
- Ein Font legt fest, wie Zeichen auf Bildschirm oder Papier aussehen.
- Ein Editor ist ein Werkzeug zum Schreiben von Text. Auf irgendeine Weise müssen die Editorkommandos vom Text unterschieden werden (Vergleiche `vi(1)` und `emacs(1)`).
- Soll der Text in einer bestimmten Form ausgegeben werden, muss er Formatierkommandos enthalten, die sich von dem eigentlichen Text unterscheiden. Die Formatierung vor der Ausgabe auf Drucker oder Bildschirm nehmen Formatierprogramme vor. Verbreitete Formatiersprachen sind `nroff(1)`, LaTeX, Texinfo und HTML:
  - `nroff(1)` ist das klassische UNIX-Werkzeug zum Formatieren von Texten zur Ausgabe auf Druckern (Aufsätze, Berichte, Bücher).
  - TeX und LaTeX haben dasselbe Ziel wie `nroff(1)`, sind aber leistungsfähiger und weiter verbreitet.
  - Texinfo ermöglicht, einen strukturierten Text sowohl auf dem Bildschirm als auf Papier auszugeben. Besonders geeignet für Anleitungen und Hilfetexte. Die Aufbereitung zum Drucken geht über TeX.
  - HTML kennt Hyperlinks, iruhende und bewegte Grafiken sowie Sound und formatiert in erster Linie für die Ausgabe auf Bildschirmen.

- Im Gegensatz zu Editoren stehen Wortprozessoren (What You See Is What You Get), bei denen man sofort beim Eingeben die Formatierung sieht. Hier gibt es jedoch unterschiedliche, nicht miteinander verträgliche Welten. Außerdem kann man mit den vorgenannten Formatierprogrammen mehr machen als mit Wortprozessoren, bei entprechendem Lernaufwand.
- Für kurze, einfache Dokumente, die gedruckt werden sollen, sind Wortprozessoren geeignet. Für lange, komplex strukturierte Dokumente, die gedruckt werden sollen, ist `nroff(1)` oder TeX/LaTeX am besten geeignet, wobei TeX/LaTeX in der Formelschreibung unübertroffen ist. Nicht zu umfangreiche Dokumente, die auf dem Bildschirm wiedergegeben werden sollen, lassen sich am besten in HTML erzeugen. Lange Dokumente, die am Bildschirm gelesen, aber unter Verzicht auf einige Möglichkeiten auch gedruckt werden sollen, sind am besten als PDF-Dokumente zu veröffentlichen.
- Neben Editoren und Formatierern enthält UNIX eine Vielzahl kleinerer Werkzeuge zur Textbearbeitung (`grep(1)`, `sort(1)`, `diff(1)`, `awk(1)` usw.).
- Zum Abfassen eines technischen Dokumentes braucht man einen Editor (`vi(1), emacs(1) oder joe(1)`) zum Schreiben und Ändern, ein Werkzeug zum Erstellen von Diagrammen (`gnuplot(1)`), ein Werkzeug zum Zeichnen (`xfig(1)`), ein Werkzeug zum Scannen und Bearbeiten von Fotos (`gimp(1)`) und ein Formatierprogramm (`latex(1)`, `pdftex(1)`).
- Die Zeilenstruktur eines Textes wird in UNIX, in DOS und auf Macs durch unterschiedliche Zeichen dargestellt, so dass gelegentlich Umformungen nötig werden.
- Die Verschlüsselung ist beim Arbeiten in Netzen der einzige Schutz vor unbefugten Zugriffen auf Daten während einer Übertragung.
- Ein symmetrischer Schlüssel dient sowohl zum Ver- wie zum Entschlüsseln und muss daher auf einem sicheren Weg dem Empfänger der verschlüsselten Nachrichten überbracht werden.
- Bei einer unsymmetrischen Verschlüsselung besitzt man ein Paar von Schlüsseln, einer davon darf veröffentlicht werden. Entweder verschlüsselt man mit dem geheimen, privaten Schlüssel und entschlüsselt mit dem öffentlichen oder umgekehrt.

2.7.22 Übung Writer's Workbench

Anmelden wie gewohnt. Schreiben Sie mit dem Editor `vi(1)` oder `emacs(1)` einen knapp zweiseitigen Text mit einer Überschrift und einigen Absätzen. Die Textdatei heiße `beispiel`. Spielen Sie mit folgenden und weiteren Werkzeugen:

```
tr "[A-Z]" "[a-z]" < beispiel > beispiel.k
cmp beispiel beispiel.k
sed 's/[A-Z]/[a-z]/g' beispiel
grep -i unix beispiel
spell beispiel
fold -50 beispiel
adjust -j -m60 beispiel
wc beispiel
```

Verzieren Sie das Beipiel mit `nroff(1)`-Kommandos, lassen Sie es durch `nroff(1)` laufen und sehen Sie sich die Ausgabe auf dem Bildschirm und auf Papier an. Zum Drucken `nroff(1)` und Druckkommando durch Pipe verbinden.

Bearbeiten Sie Ihren Text mit dem Shellskript `frequenz`. Welche Wörter kommen häufig vor, welche selten? Wo tauchen Tippfehler wahrscheinlich auf? Suchen Sie die Tippfehler in Ihrem Text mit dem `vi(1)` (Schrägstrich).

Schreiben Sie eine unsortierte zweispaltige Liste mit Familiennamen und Telefonnummern. Die Datei namens `liste` soll auch einige mehrfache Eintragungen enthalten. Bearbeiten Sie es wie folgt:

```
sort liste
sort -u liste
sort -d liste
sort +1 -2 liste
sort liste | uniq
sort liste | cut -f1
sort liste | awk '$1 != prev {print; prev = $1 }'
```

Untersuchen Sie mit dem Shellskript zur Textanalyse einen leichten Text - aus einer Tageszeitung etwa - und einen schwierigen. Wir empfehlen IMMANUEL KANT *Der Streit der Fakultäten*, immer aktuell. Wo sind Ihre eigenen Texte einzuordnen? Beenden der Sitzung mit `exit`.

2.7.23 Fragen Writer's Workbench

- Was ist ein Zeichensatz?
- Was ist eine Kodetafel?
- Was ist ein Font?
- Was ist die ASCII-Tabelle? Was ist Latin-1?
- Was ist ein regulärer Ausdruck? Wozu braucht man REs?
- Wie suche ich nach Zeichenfolgen in Dateien?

- Was ist ein Editor? Welches sind die beiden großen UNIX-Editoren?
- Was macht ein Stream-Editor wie der `sed`?
- Was kann man mit dem `awk` machen?
- Welche beiden Arten der Verschlüsselung gibt es? Warum braucht man beide?
- Was erledigen Formatier- oder Satzprogramme?
- Vor- und Nachteile von Formatierprogrammen gegenüber WYSIWYG-Programmen?
- Was ist LaTeX? Stärken von LaTeX?
- Was ist Hypertext? Was ist ein Hyperlink? Was ist HTML?
- Wie unterscheiden sich Textdateien aus verschiedenen Betriebssystemen (DOS/Microsoft Windows, Macintosh, UNIX)?
- Welche Drucksysteme sind unter UNIX gebräuchlich? Welche Aufgaben erledigt ein Drucksystem?

2.8 Programmer's Workbench

Unter der *Werkbank des Programmierers* werden UNIX-Werkzeuge zusammengefaßt, die zum Programmieren benötigt werden. Auf Maschinen, die nicht zur Programmentwicklung eingesetzt werden, können sie fehlen. Das Werkzeug `make(1)` und die Revisionskontrolle sind auch bei Projekten außerhalb der Programmierung nützlich, vor allem beim Bearbeiten umfangreicher Manuskripte.

2.8.1 Nochmals die Editoren

Editoren wurden bereits im UNIX-Kapitel, Abschnitt 2.7 *Writer's Workbench* auf Seite 134 erläutert. Hier geht es nur um einige weitere Eigenschaften des Editors `vi(1)`, die beim Schreiben von Programmquellen von Belang sind.

Im Quellcode werden üblicherweise Schleifenrümpfe und dergleichen um eine Tabulatorbreite eingerückt, die als Default 8 Leerzeichen entspricht. Bei geschachtelten Schleifen stößt der Text schnell an den rechten Seitenrand. Es empfiehlt sich, in dem entsprechenden Verzeichnis eine Datei `.exrc` mit den Zeilen:

```
set tabstop=4
set showmatch
set number
```

anzulegen. Die Option `showmatch` veranlaßt den `vi(1)`, bei jeder Eingabe einer rechten Klammer kurz zur zugehörigen linken Klammer zu springen. Die Option `number` führt zur Anzeige der Zeilennummern, die jedoch nicht

Bestandteil des Textes werden. Eine Zeile `set lisp` ist eine Hilfe beim Eingeben von LISP-Quellen.

Steht der Cursor auf einer Klammer, so läßt das Kommando `%` den Cursor zur Gegenklammer springen und dort verbleiben.

Auch beim `emacs(1)` gibt es einige Wege, das Schreiben von Quellen zu erleichtern, insbesondere natürlich, falls es um LISP geht. Der Editor `nedit(1)` lässt sich auf den Stil aller gängigen Programmiersprachen einschließlich LaTeX einstellen und ist in vielen Linux-Distributionen enthalten.

2.8.2 Compiler und Linker (cc, ccom, ld)

Auf das Schreiben der Quelltexte mit einem Editor folgt ihre Übersetzung in die Sprache der jeweiligen Maschine mittels eines Übersetzungsprogrammes, meist eines **Compilers**. Jedes vollständige UNIX-System enthält einen C-Compiler; Compiler für weitere Programmiersprachen sind optional. Auf unserer Anlage sind zusätzlich ein FORTRAN- und ein PASCAL-Compiler vorhanden, wobei von FORTRAN gegenwärtig die Versionen 77 und 90 nebeneinander laufen.

Kompilieren bedeutete vor der EDV-Zeit zusammentragen. Im alten Rom hatte es auch noch die Bedeutung von plündern. In unseren Herzensergießungen haben wir viel aus Büchern, Zeitschriften, WWW-Seiten und Netnews kompiliert.

Ein Compiler übersetzt den Quellcode eines Programmes in Maschinensprache. Die meisten Programme enthalten Aufrufe von externen Programmodulen, die bereits vorübersetzt und in Bibliotheken zusammengefaßt sind. Beispiele sind Ausgaberoutinen oder mathematische Funktionen. Der ausführbare Code dieser externen Module wird erst vom **Linker**[40] mit dem Programmcode vereinigt, so daß ein vollständiges ausführbares Programm entsteht. Es gibt die Möglichkeit, die externen Module erst zur Laufzeit hinzuzunehmen; das heißt **dynamisches Linken** und spart Speicherplatz. Dabei werden die Module entweder beim Laden des Programms in den Arbeitsspeicher oder erst bei ihrem Aufruf hinzugeladen (load on demand). Benutzen mehrere Programme ein in den Arbeitsspeicher kopiertes Modul gemeinsam anstatt jeweils eine eigene Kopie anzulegen, so kommt man zu den **Shared Libraries** und spart nochmals Speicherplatz.

Die Aufrufe lauten `cc(1)`, `f77(1)`, `f90(1)` und `pc(1)`. Diese Kommandos rufen **Compilertreiber** auf, die ihrerseits die eigentlichen Compiler `/lib/ccom`, `f77comp`, `f90comp` und `pascomp` starten und noch weitere Dinge erledigen. Ohne Optionen rufen die Compilertreiber auch noch den Linker `/bin/ld(1)` auf, so dass das Ergebnis ein lauffähiges Programm ist, das als Default den Namen `a.out(4)` trägt. Mit dem Namen `a.out(4)` sollte man nur vorübergehend arbeiten (mit `mv(1)` ändern). Der Aufruf des C-Compilers sieht beispielsweise so aus:

---

[40]Linker werden auch Binder, Mapper oder Loader genannt. Manchmal wird auch zwischen Binder und Loader unterschieden, soll uns hier nicht beschäftigen.

```
cc source.c
cc -g source.c -lm -L./lib -I. -DMAX=100
```

Die erste Zeile stellt den minimalen Aufruf dar, die zweite einen um gängige Optionen erweiterten. Die Option `-g` veranlaßt den Compiler, zusätzliche Informationen für den symbolischen Debugger zu erzeugen. Weitere Optionen sind:

- `-v` (verbose) führt zu etwas mehr Bemerkungen beim Übersetzen,
- `-o` (output) benennt die ausführbare Datei mit dem auf die Option folgenden Namen, meist derselbe wie die Quelle, nur ohne Kennung: `cc -o myprogram myprogram.c`,
- `-c` hört vor dem Linken auf, erzeugt Objektfile mit der Kennung `.o`,
- `-p` (profile) erzeugt beim Ablauf des Programmes eine Datei `mon.out`, das mit dem Profiler `prof(1)` ausgewertet werden kann, um Zeitinformationen zum Programm zu erhalten,
- `-O` optimiert das ausführbare Programm oder auch nicht.

Der Quelltext des C-Programmes steht in der Datei `source.c`, die einen beliebigen Namen tragen kann, nur sollte der Name mit der Kennung `.c` enden. Die anschließende Option `-lm` fordert den Linker auf, die mathematische Standard-Bibliothek einzubinden. Die Option `-L./lib` wendet sich ebenfalls an den Linker und teilt ihm mit, dass sich im Verzeichnis `./lib` weitere Bibliotheken befinden. Die Reihenfolge, in der Bibliotheken eingebunden werden, ist wichtig. Die Option `-I.` veranlasst den Präprozessor, Include-Dateien auch im aktuellen Verzeichnis zu suchen, was er nicht immer automatisch tut. Es könnte auch ein anderes Verzeichnis angegeben werden. Die Option `-DMAX=100` definiert eine symbolische Konstante namens `MAX` und weist ihr den Wert 100 zu, genau wie eine Zeile:

```
#define MAX 100
```

im Quelltext, nur eben hier mit der Möglichkeit, den Wert bei der Übersetzung zu bestimmen. Speichermodelle wie unter PC-DOS gibt es in UNIX nicht. Hat man Speicher, kann man ihn uneingeschränkt nutzen.

Für C-Programme gibt es einen **Syntax-Prüfer** namens `lint(1)`, den man unbedingt verwenden sollte. Er reklamiert nicht nur Fehler, sondern auch Stilmängel. Manchmal beanstandet er auch Dinge, die man bewußt gegen die Regeln geschrieben hat. Man muß seinen Kommentar sinnvoll interpretieren. Aufruf:

```
lint mysource.c
```

Ein verbesserter `lint`, ein *Secure Programming Lint* findet sich bei der University of Virginia unter:

```
http://www.splint.org/
```

Unter Linux ist lint(1) nicht überall vorhanden, dann kann man den Compiler gcc(1) mit einer Option aufrufen, die ihn nur zu einer Prüfung der Syntax veranlasst:

```
gcc -fsyntax-only -pedantic -Wall mysource.c
```

Ferner gibt es unter einigen UNIXen für C-Quelltexte einen **Beautifier** namens cb(1), der den Text in eine standardisierte Form mit Einrückungen usw. bringt und die Lesbarkeit erleichtert:

```
cb source.c > source.b
```

Wenn man mit dem Ergebnis source.b zufrieden ist, löscht man die ursprüngliche Datei source.c und benennt source.b in source.c um.

2.8.3 Unentbehrlich (make)

Größere Programme sind stark gegliedert und auf mehrere bis viele Dateien und Verzeichnisse verteilt. Der Compileraufruf wird dadurch länglich, und die Wahrscheinlichkeit, etwas zu vergessen, steigt. Hier hilft make(1). Man schreibt einmal alle Angaben für den Compiler in ein makefile (auch Makefile) und ruft dann zum Kompilieren nur noch make(1) auf. Für Manuskripte ist make(1) ebenfalls zu gebrauchen. Statt Makefiles ließen sich auch Shellskripte einsetzen, die Stärke von make(1) liegt jedoch im Umgang mit Dateien unter Beachtung des Zeitstempels mtime (jüngster schreibender Zugriff). Werkzeuge wie make(1) werden als *Builder* bezeichnet.

Man lege für das Projekt ein eigenes Unterverzeichnis an, denn make(1) sucht zunächst im Arbeits-Verzeichnis. Das makefile beschreibt die Abhängigkeiten (dependencies) der Programmteile voneinander und enthält die Kommandozeilen zu ihrer Erzeugung. Ein einfaches makefile sieht so aus (Zeilen mit Kommandos müssen durch einen Tabulatorstop – *nicht* durch Spaces – eingerückt sein):

```
pgm:  a.o  b.o
        cc  a.o  b.o  -o pgm
a.o:  incl.h  a.c
        cc  -c a.c
b.o:  incl.h  b.c
        cc  -c  b.c
```

Quelle 2.29 : Einfaches Makefile

und ist folgendermaßen zu verstehen:

- Das ausführbare Programm (Ziel, Target) namens pgm hängt ab von den Modulen im Objektcode a.o und b.o. Es entsteht durch den Compileraufruf cc a.o b.o -o pgm.
- Das Programmodul a.o hängt ab von der include-Datei incl.h und dem Modul im Quellcode a.c. Es entsteht durch den Aufruf des Compilers mit cc -c a.c. Die Option - c unterbindet das Linken.

- Das Programmodul `b.o` hängt ab von derselben include-Datei und dem Modul im Quellcode `b.c`. Es entsteht durch den Compileraufruf `cc -c b.c`.

Ein `makefile` ist ähnlich aufgebaut wie ein Backrezept: erst werden die Zutaten aufgelistet, dann folgen die Anweisungen. Zu beachten ist, daß man am Ziel startet und rückwärts bis zu den Quellen geht. Kommentar beginnt mit einem Doppelkreuz und reicht bis zum Zeilenende. Leerzeilen werden ignoriert.

`make(1)` verwaltet auch verschiedene Versionen der Programmodule und paßt auf, daß eine neue Version in alle betroffenen Programmteile eingebunden wird. Umgekehrt wird eine aktuelle Version eines Moduls nicht unnötigerweise kompiliert. Warum wird im obigen Beispiel die include-Datei `incl.h` ausdrücklich genannt? Der Compiler weiß doch auf Grund einer entsprechenden Zeile im Quelltext, daß diese Datei einzubinden ist? Richtig, aber `make(1)` muß das auch wissen, denn die include-Datei könnte sich ändern, und dann müssen alle von ihm abhängigen Programmteile neu übersetzt werden. `make(1)` schaut nicht in die Quellen hinein, sondern nur auf die Zeitstempel (`mtime`) der Zutaten. Unveränderliche include-Dateien wie `stdio.h` brauchen nicht im `makefile` aufgeführt zu werden.

Nun ein etwas umfangreicheres Beispiel, das aber längst noch nicht alle Fähigkeiten von `make(1)` ausreizt:

```
# Kommentar, wie ueblich

CC = /bin/cc
CFLAGS =
FC = /usr/bin/f77
LDFLAGS = -lcl

all: csumme fsumme clean

csumme: csumme.c csv.o csr.o
        $(CC) -o csumme csumme.c csv.o csr.o

csv.o: csv.c
        $(CC) -c csv.c

csr.o: csr.c
        $(CC) -c csr.c

fsumme: fsumme.c fsr.o
        $(CC) -o fsumme fsumme.c fsr.o $(LDFLAGS)

fsr.o: fsr.f
        $(FC) -c fsr.f

clean:
        rm *.o
```

Quelle 2.30 : Makefile mit Makros und Dummy-Zielen

Zunächst werden einige Makros definiert, z. B. der Compileraufruf `CC`. Überall, wo im Makefile das Makro mittels `$(CC)` aufgerufen wird, wird es vor der Ausführung wörtlich ersetzt. Auf diese Weise kann man einfach einen anderen Compiler wählen, ohne im ganzen Makefile per Editor ersetzen zu müssen. Dann haben wir ein Dummy-Ziel `all`, das aus einer Aufzählung weiterer Ziele besteht. Mittels `make all` wird dieses Dummy-Ziel erzeugt, d. h. die aufgezählten Ziele. Unter diesen befindet sich auch eines namens `clean`, das ohne Zutaten daherkommt und offenbar nur bestimmte Tätigkeiten wie das Löschen temporärer Dateien bezweckt. Ein Dummy-Ziel ist immer out-of-date, die zugehörigen Kommandos werden immer ausgeführt.

`make(1)` darf rekursiv aufgerufen werden, ein Makefile darf `make`-Aufrufe enthalten, die sich auf *andere* Makefiles beziehen. Das kann so aussehen:

```
......
        cd subdirectory ; make all ; make clean
......
```

Gelangt `make` an diese Zeile, springt es in das Unterverzeichnis, sucht dort ein Makefile und erzeugt die Ziele `all` und `clean`. Anschließend macht es im Makefile des aktuellen Verzeichnisses weiter. Eine Anwendung ist ein aus mehreren Kapiteln bestehendes Skriptum, das komplett als Report formatiert werden soll, dazu noch die Kapitel in jeweils einem eigenen Unterverzeichnis als Artikel samt Folien. Natürlich sind die Makefiles in den Unterverzeichnissen weiche Links auf ein einziges Makefile. Ohne `make(1)` und eine entsprechende Verzeichnisstruktur würde man sich dumm und dämlich tippen.

Im GNU-Projekt wird Software im Quellcode für verschiedene Systeme veröffentlicht. In der Regel muß man die Quellen auf der eigenen Anlage kompilieren. Infolgedessen gehören zu den GNU-Programmen fast immer umfangreiche Makefiles oder sogar Hierarchien davon. Übung im Gebrauch von `make(1)` erleichtert die Einrichtung von GNU-Software daher erheblich. Oft wird ein an das eigene System angepaßtes Makefile erst durch ein Kommando `./configure` erzeugt. Die Reihenfolge bei solchen Programmeinrichtungen lautet dann:

```
./configure
(vi Makefile)
make
make install
make clean
```

wobei `make install` Schreibrechte in den betroffenen Verzeichnissen erfordert, also meist Superuserrechte. Gelegentlich wird `make(1)` aus einem Shellskript heraus aufgerufen, das einige Dinge vorbereitet. So wird zum Beispiel `sendmail(1)` durch den Aufruf des mitgelieferten Shellskripts `Build` erzeugt.

Das Skript `configure` erlaubt oft die Option `-prefix=DIR`, wobei `DIR` das Verzeichnis ist, in dem das ganze Gerödel eingerichtet werden soll, defaultmäßig meist `/usr/local`, aber manchmal besser `/usr` oder `/opt`.

Da von `configure` alles Weitere abhängt, sollte man sich die zugehörige Protokoll-Datei `config.log` ansehen, auch wenn anscheinend keine Probleme aufgetreten sind.

Statt `make clean` kann man auch `make distclean` versuchen, das räumt noch gründlicher auf, so daß hinterher wieder mit `./configure` ein Neubeginn möglich ist.

2.8.4 Debugger (xdb, gdb)

Programme sind Menschenwerk und daher fehlerhaft[41]. Es gibt keine Möglichkeit, die Fehlerfreiheit eines Programmes festzustellen oder zu beweisen außer in trivialen oder idealen Fällen.

Die Fehler lassen sich in drei Klassen einteilen. Verstöße gegen die Regeln der jeweiligen Programmiersprache heißen **Grammatikfehler** oder **Syntaxfehler**. Sie führen bereits zu einem Abbruch des Kompiliervorgangs und lassen sich schnell lokalisieren und beheben. Der C-Syntax-Prüfer `lint` ist das beste Werkzeug zu ihrer Entdeckung. `wihle` statt `while` wäre ein einfacher Syntaxfehler. Fehlende oder unpaarige Klammern sind auch beliebt, deshalb enthält der `vi(1)` eine Funktion zur Klammerprüfung. Unzulässige Operationen mit Pointern sind ebenfalls an der Tagesordnung. Geht es um Texte, so fallen Tippfehler und Grammatikfehler in diese Klasse.

Falls das Programm die Kompilation ohne Fehlermeldung hinter sich gebracht hat, startet man es. Dann melden sich die **Laufzeitfehler**, die unter Umständen nur bei bestimmten und womöglich seltenen Parameterkonstellationen auftreten. Ein typischer Laufzeitfehler ist die Division durch eine Variable, die manchmal den Wert Null annimmt. Die Fehlermeldung lautet *Floating point exception*. Ein anderer häufig vorkommender Laufzeitfehler ist die Überschreitung von Arraygrenzen oder die Verwechslung von Variablen und Pointern, was zu einem *Memory fault*, einem Speicherfehler führt.

Die dritte Klasse bilden die **logischen Fehler** oder **Denkfehler**. Sie werden auch **semantische Fehler** genannt. Das Programm arbeitet einwandfrei, nur tut es nicht das, was sich der Programmierer vorgestellt hat. Ein typischer Denkfehler ist das Verzählen bei den Elementen eines Arrays oder bei Schleifendurchgängen um genau eins. Hier hilft der Computer nur wenig, da der Ärmste ja gar nicht weiß, was sich der Programmierer vorstellt. Diese Fehler kosten viel Mühe, doch solcherlei Verdrüsse pflegen die Denkungskräfte anzuregen, meint WILHELM BUSCH und hat recht.

Eine vierte Fehlerklasse liegt fast schon außerhalb der Verantwortung des Programmierers. Wenn das mathematische **Modell** zur Beschreibung eines

[41]Es irrt der Mensch, so lang er strebt. GOETHE, Faust. Oder *errare humanum est*, wie wir Lateiner sagen. Noch etwas älter: *αμαρτωλαι εν ανθρωποισιν επονται θνητοις*. Die entsprechende Aussage in babylonischer Keilschrift aus dem Codex Kombysis können wir leider aus Mangel an einem TeX-Font vorläufig nicht wiedergeben. In der nächsten Auflage werden wir jedoch eine eingescannte Zeichnung aus der Höhle von Rienne-Vaplus zeigen, die als älteste Dokumentation obiger Weisheit gilt.

realen Problems ungeeignet ist, mag das Programm so fehlerarm sein wie es will, seine Ergebnisse gehen an der Wirklichkeit vorbei. Für bestimmte Zwecke ist eine Speisekarte ein brauchbares Modell einer Mahlzeit, für andere nicht.

In diese Klasse fallen auch Fehler, die dadurch entstehen, dass wir im Computer stets mit Zahlen endlicher Länge rechnen, während in Wirklichkeit die Zahl π unendlich viele Dezimalstellen hat und gemessene Größen statistischen Schwankungen unterliegen, also unscharf begrenzte Intervalle darstellen. Grundkenntnisse in moderner numerischer Mathematik bewahren vor blindem Glauben an den Computer.

Ob und wie inhaltliche Fehler in Texten – falsche oder fehlende Angaben – einer der vorstehenden Klassen sinnvoll zugeordnet werden können, ist noch zu überlegen. Dann gibt es noch den Fehler *Thema verfehlt*, der vielleicht zum Modell-Fehler passt. Hintergrund dieser Gedanken ist die Anwendung von Software-Werkzeugen in Text-Projekten, was erwiesenermaßen die Arbeit erleichtert.

Ein Fehler wird im Englischen auch als *bug* bezeichnet, was soviel wie Wanze oder Laus bedeutet. Ein Programm zu entlausen heißt Debugging. Dazu braucht man einen Debugger (déverminateur, déboguer). Das sind Programme, unter deren Kontrolle das verlauste Programm abläuft. Man hat dabei vielfältige Möglichkeiten, in den Ablauf einzugreifen. Ein **absoluter Debugger** wie der `adb(1)` bezieht sich dabei auf das lauffähige Programm im Arbeitsspeicher – nicht auf den Quellcode – und ist somit für die meisten Aufgaben wenig geeignet. Ein **symbolischer Debugger** wie der `sdb(1)`, der GNU `gdb(1)` oder der `xdb(1)` bezieht sich auf die jeweilige Stelle im Quelltext[42]. Debugger sind mächtige und hilfreiche Werkzeuge. Manche Programmierer gehen so weit, daß sie das Schreiben eines Programms als Debuggen einer leeren Datei bzw. eines weißen Blattes Papier ansehen. In der Übung wird eine einfache Anwendung des Debuggers vorgeführt.

Falls Sie auch mit dem UNIX-Debugger nicht alle Würmer in Ihrem Programm finden und vertreiben können, möchten wir Ihnen noch ein altes Hausrezept verraten, das aus einer Handschrift des 9. Jahrhunderts stammt. Das Rezept ist im Raum Wien – München entstanden und unter den Namen *Contra vermes* oder *Pro nescia* bekannt. Leider ist die `README`-Datei, welche die Handhabung erklärt, verlorengegangen. Wir schlagen vor, die Zeilen als Kommentar in das Programm einzufügen. Hier der Text:

> Gang út, nesso, mid nigun nessiklinon,
> ût fana themo marge an that bên,
> fan thêmo bêne an that flêsg,
> ût fan themo flêsgke an thia hûd,
> ût fan thera hûd an thesa strâla.
> Drohtin. Uuerthe sô!

[42]Real programmers don't use source language debuggers.

2.8.5 Profiler (time, gprof)

Profiler sind ebenfalls Programme, unter deren Kontrolle ein zu untersuchendes Programm abläuft. Ziel ist die Ermittlung des Zeitverhaltens in der Absicht, das Programm schneller zu machen. Ein einfaches UNIX-Werkzeug ist `time(1)`:

```
time prim 1000000
```

Die Ausgabe sieht so aus:

```
real    0m 30.65s
user    0m 22.53s
sys     0m  1.07s
```

und bedeutet, daß die gesamte Laufzeit des Programms `prim` 30.65 s betrug, davon entfielen 22.53 s auf die Ausführung von Benutzeranweisungen und 1.07 s auf Systemtätigkeiten. Die Ausgabe wurde durch einen Aufruf des Primzahlenprogramms aus dem Skriptum *Programmieren in C/ C++* erzeugt, das selbst Zeiten mittels des Systemaufrufs `time(2)` misst und rund 22 s für die Rechnung und 4 s für die Bildschirmausgabe meldet.

Ein weiterer Profiler ist `gprof(1)`. Seine Verwendung setzt voraus, daß das Programm mit der Option `-G` kompiliert worden ist. Es wird gestartet und erzeugt neben seiner normalen Ausgabe eine Datei `gmon.out`, das mit `gprof(1)` betrachtet wird. Besser noch lenkt man die Ausgabe von `gprof(1)` in eine Datei um, die sich lesen und editieren läßt:

```
gprof prim > prim.gprofile
```

Eine stark gekürzte Analyse mittels `gprof(1)` sieht so aus:

```
%time   the percentage of the total running time of the
        program used by this function.

cumsecs a running sum of the number of seconds accounted
        for by this function and those listed above it.

seconds the number of seconds accounted for by this
        function alone.  This is the major sort for this
        listing.

calls   the number of times this function was invoked, if
        this function is profiled, else blank.

name    the name of the function.  This is the minor sort
        for this listing.

%time cumsecs seconds  calls msec/call name
 52.1   12.18   12.18                  $$remU
 22.2   17.38    5.20                  $$mulU
```

```
 20.8   22.25    4.87 333332       0.01 ttest
  2.1   22.74    0.49   9890       0.05 _doprnt
  0.8   22.93    0.19                   _mcount
  0.6   23.08    0.15                   $$divide_by_const
  0.6   23.22    0.14      1     140.00 main
  0.3   23.29    0.07   9890       0.01 _memchr
  0.2   23.34    0.05                   _write_sys
  0.1   23.36    0.02   9890       0.00 _printf
  0.0   23.37    0.01   9887       0.00 _write
  0.0   23.38    0.01   9887       0.00 _xflsbuf
  0.0   23.39    0.00   9890       0.00 _wrtchk
  0.0   23.39    0.00      1       0.00 _sscanf
  0.0   23.39    0.00      1       0.00 _start
  0.0   23.39    0.00      1       0.00 _strlen
  0.0   23.39    0.00      1       0.00 atexit
  0.0   23.39    0.00      1       0.00 exit
  0.0   23.39    0.00      1       0.00 ioctl
```

Wir sehen, daß die Funktion `ttest()` sehr oft aufgerufen wird und 4,87 s verbrät. Die beiden ersten Funktionen werden vom Compiler zur Verfügung gestellt (Millicode aus `/usr/lib/milli.a`) und liegen außerhalb unserer Reichweite.

Für genauere Auskünfte zieht man den Systemaufruf `times(2)`, den Debugger oder das UNIX-Kommando `prof(1)` in Verbindung mit der Subroutine `monitor(3)` heran.

2.8.6 Archive, Bibliotheken (ar)

Viele Teilaufgaben in den Programmen wiederholen sich immer wieder. Das sind Aufgaben, die mit dem System zu tun haben, Befehle zur Bildschirmsteuerung, mathematische Berechnungen wie Logarithmus oder trigonometrische Funktionen, Datenbankfunktionen oder Funktionen zur Abfrage von Meßgeräten am Bus.

Damit man diese Funktionen nicht jedesmal neu zu erfinden braucht, werden sie in **Bibliotheken** gepackt, die dem Programmierer zur Verfügung stehen. Teils stammen sie vom Hersteller des Betriebssystems (also ursprünglich AT&T), teils vom Hersteller der Compiler (bei uns Hewlett-Packard und GNU) oder der Anwendungssoftware, teils von Benutzern. Bibliotheken enthalten Programmbausteine, es lassen sich aber auch andere Dateien (Texte, Grafiken) in gleicher Weise zusammenfassen. Dann spricht man allgemeiner von **Archiven**. Außer den Dateien enthalten Archive Verwaltungsinformationen (Index) zum schnellen Finden der Inhalte. Diese Informationen wurden früher mit dem Kommando `ranlib(1)` eigens erzeugt, heute erledigt `ar(1)` das mit. Die Verwendung von Bibliotheken beim Programmieren wird im Skriptum *Programmieren in C/C++* erläutert.

Außer den mit dem Compiler gelieferten Bibliotheken kann man zusätzlich erworbene oder selbst erstellte Bibliotheken verwenden. Im Handel sind beispielsweise Bibliotheken mit Funktionen für Bildschirmmasken, zur Verwaltung index-sequentieller Dateien, für Grafik, zur Meßwerterfassung und -aufbereitung und für besondere mathematische Aufgaben. Auch aus dem Netz laufen Bibliotheken zu. Eigene Bibliotheken erzeugt man mit dem UNIX-Kommando `ar(1)`; das Datei-Format ist unter `ar(4)` beschrieben. Ein Beispiel zeige den Gebrauch. Wir haben ein Programm `statistik.c` zur Berechnung von Mittelwert und Varianz der in der Kommandozeile mitgegebenen ganzen Zahlen geschrieben:

```
/* Statistische Auswertung von eingegebenen Werten
   Privat-Bibliothek ./libstat.a erforderlich
   Compileraufruf cc statistik.c -L . -lstat
*/

#define  MAX 100        /* max. Anzahl der Werte */
#include <stdio.h>

void exit(); double mwert(), varianz();

main(int argc, char *argv[])

{
int i, a[MAX];

if (argc < 3) {
    puts("Zuwenig Werte"); exit(-1);
}

if (argc > MAX + 1) {
    puts("Zuviel Werte"); exit(-1);
}

/* Uebernahme der Werte in ein Array */

a[0] = argc - 1;

for (i = 1; i < argc; i++) {
    sscanf(argv[i], "%d", a + i);
}

/* Ausgabe des Arrays */

for (i = 1; i < argc; i++) {
    printf("%d\n", a[i]);
}

/* Rechnungen */

printf("Mittelwert: %f\n", mwert(a));
printf("Varianz:    %f\n", varianz(a));

return 0;
}
```

Quelle 2.31 : C-Programm Statistik mit Benutzung einer eigenen Funktionsbibliothek

Das Programm verwendet die Funktionen `mwert()` und `varianz()`, die wir aus einer hausgemachten Funktionsbibliothek namens `libstat.a` entnehmen. Der im Kommentar genannte Compileraufruf mit der Option `-L .` veranlaßt den Linker, diese Bibliothek im Arbeits-Verzeichnis zu suchen. Die Funktionen sehen so aus:

```
double mwert(x)
int *x;
{
int j, k;
double m;

for (j = 1, k = 0; j <= *x; j++) {
    k = k + x[j];
}
m = (double)k / (double)*x;
return m;
}
```

Quelle 2.32 : C-Funktion Mittelwert ganzer Zahlen

```
extern double mwert();

double varianz(x)
int *x;
{
int j;
double m, s, v;

m = mwert(x);

for (j = 1, s = 0; j <= *x; j++) {
s = s + (x[j] - m) * (x[j] - m);
}
v = s / (*x - 1);
return v;
}
```

Quelle 2.33 : C-Funktion Varianz ganzer Zahlen

Diese Funktionen werden mit der Option `-c` kompiliert, so daß wir zwei Objektfiles `mwert.o` und `varianz.o` erhalten. Mittels des Aufrufes

```
ar -r libstat.a mwert.o varianz.o
```

erzeugen wir die Funktionsbibliothek `libstat.a`, auf die mit der Compileroption `-lstat` zugegriffen wird. Der Vorteil der Bibliothek liegt darin, daß man sich nicht mit vielen einzelnen Funktionsfiles herumzuschlagen braucht, sondern mit der Compileroption gleich ein ganzes Bündel verwandter Funktionen erwischt. In das Programm eingebunden werden nur die Funktionen, die wirklich benötigt werden.

Merke: Ein Archiv ist weder verdichtet noch verschlüsselt. Dafür sind andere Werkzeuge (`gzip(1)`, `crypt(1)`) zuständig.

2.8.7 Weitere Werkzeuge

Das Werkzeug `cflow(1)` ermittelt die Funktionsstruktur zu einer Gruppe von C-Quell- und Objektfiles. Der Aufruf:

```
cflow statistik.c
```

liefert auf `stdout`

```
1 main: int(), <statistik.c 15>
2 puts: <>
3 exit: <>
4 sscanf: <>
5 printf: <>
6 mwert: <>
7 varianz: <>
```

was besagt, daß die Funktion `main()` vom Typ `int` ist und in Zeile 15 des Quelltextes `statistik.c` definiert wird. `main()` ruft seinerseits die Funktionen `puts`, `exit`, `sscanf` und `printf` auf, die in `statistik.c` nicht definiert werden, da sie Teil der Standardbibliothek sind. Die Funktionen `mwert` und `varianz` werden ebenfalls aufgerufen und nicht definiert, da sie aus einer Privatbibliothek stammen.

Das Werkzeug `cxref(1)` erzeugt zu einer Gruppe von C-Quellfiles eine Kreuzreferenzliste aller Symbole, die nicht rein lokal sind. Der Aufruf

```
cxref fehler.c
```

gibt nach `stdout` eine Liste aus, deren erste Zeilen so aussehen:

```
fehler.c:

SYMBOL        FILE                    FUNCTION LINE

BUFSIZ        /usr/include/stdio.h    --       *10
EOF           /usr/include/stdio.h    --        70 *71
FILE          /usr/include/stdio.h    --       *18 78 123
                                                127 128 201
                                                223
FILENAME_MAX  /usr/include/stdio.h    --       *67
FOPEN_MAX     /usr/include/stdio.h    --       *68
L_ctermid     /usr/include/stdio.h    --       *193
L_cuserid     /usr/include/stdio.h    --       *194
L_tmpnam      /usr/include/stdio.h    --       *61
```

```
NULL          /usr/include/stdio.h  --        35 *36
PI            fehler.c              --       *27
P_tmpdir      /usr/include/stdio.h  --       *209
SEEK_CUR      /usr/include/stdio.h  --       *55
SEEK_END      /usr/include/stdio.h  --       *56
SEEK_SET      /usr/include/stdio.h  --        53 *54
TMP_MAX       /usr/include/stdio.h  --        63 *64
_CLASSIC_ANSI_TYPES /usr/include/stdio.h --   162
```

Durch die `include`-Datei `stdio.h` und gegebenenfalls durch Bibliotheksfunktionen geraten viele Namen in das Programm, von denen man nichts ahnt. Ferner gibt es einige Werkzeuge zur Ermittlung und Bearbeitung von Strings in Quellfiles und ausführbaren Programmen, teilweise beschränkt auf C-Programme (tt strings(1), `xstr(1)`).

Weitere wichtige Werkzeuge sind ein Lineal und Buntstifte, mit denen man zusammengehörende Namen oder Teile im Quelltext markiert.

2.8.8 Versionsverwaltung mit RCS, SCCS und CVS

Größere Projekte werden von zahlreichen, unter Umständen wechselnden Programmierern oder Autoren gemeinsam bearbeitet. In der Regel werden die so entstandenen Programmpakete über Jahre hinweg weiterentwickelt und vielleicht auf mehrere Systeme portiert. Die Arbeit vollzieht sich in mehreren **Stufen** parallel zur Zeitachse:

- Aufgabenstellung
- Aufgabenanalyse
- Umsetzung in eine Programmiersprache
- Testen
- Dokumentieren
- vorläufige Freigabe
- endgültige Freigabe
- Weiterentwicklung, Pflege

Des weiteren wird ein Programmpaket in viele überschaubare **Module** aufgeteilt. Von jedem Modul entstehen im Verlauf der Arbeit mehrere Fassungen oder **Versionen**. Der Zustand des ganzen Projektes läßt sich in einem dreidimensionalen Koordinatensystem mit den Achsen Modul, Stufe (Zeit) und Version darstellen. Das von WALTER F. TICHY entwickelte **Revision Control System RCS** ist ein Werkzeug, um bei dieser Entwicklung Ordnung zu halten. Es ist einfach handzuhaben und verträgt sich gut mit `make(1)`. Das RCS erledigt drei Aufgaben:

- Es führt Buch über die Änderungen an den Texten.
- Es ermöglicht, ältere Versionen wiederherzustellen, ohne daß diese vollständig gespeichert zu werden brauchen (Speichern von Differenzen).

- Es verhindert gleichzeitige schreibende Zugriffe mehrerer Benutzer auf denselben Text.

Sowie es um mehr als Wegwerfprogramme geht, sollte man `make(1)` und RCS einsetzen. Der geringe Aufwand zum Einarbeiten wird bei der weiteren Arbeit mehr als wett gemacht. Arbeiten mehrere Programmierer an einem Projekt, kommt man um RCS oder ähnliches nicht herum. Beide Werkzeuge sind auch für Manuskripte oder WWW-Dateien zu verwenden. RCS ist in den meisten Linux-Distributionen enthalten. Man beginnt folgendermaßen:

- Unterverzeichnis anlegen, hineinwechseln.
- Mit einem Editor die erste Fassung des Quelltextes schreiben. Irgendwo im Quelltext - z. B. im Kommentar - sollte `$Header: /home/debian/unix/unix8.tex,v 1.1.1.1 2005/02/06 20:01:11 wulf` oder `$Id: unix8.tex,v 1.1.1.1 2005/02/06 20:01:11 wulf Exp $` vorkommen, siehe unten. Dann übergibt man mit dem Kommando `ci filename` (check in) die Datei dem RCS. Dieses ergänzt die Datei durch Versionsinformationen und macht eine nur lesbare RCS-Datei (444) mit der Kennung `,v` daraus. Die ursprüngliche Datei löschen.
- Mit dem Kommando `co filename` (check out, ohne `,v`) bekommt man eine Kopie seiner Datei zurück, und zwar nur zum Lesen. Diese Kopie kann man mit allen UNIX-Werkzeugen bearbeiten, nur das Zurückschreiben mittels `ci` verweigert das RCS.
- Mit dem Kommando `co -l filename` wird eine les- und schreibbare Kopie erzeugt. Dabei wird die RCS-Datei für weitere, gleichzeitige Schreibzugriffe gesperrt (l = lock). Die Kopie kann man mit allen UNIX-Werkzeugen bearbeiten, Umbenennen wäre jedoch ein schlechter Einfall.
- Beim Zurückstellen mittels `ci filename` hat man Gelegenheit, einen kurzen Kommentar in die Versionsinformationen zu schreiben wie Grund und Umfang der Änderung. Mittels `rlog filename` werden die Versionsinformationen auf den Schirm geholt. Enthält der Quelltext die Zeichenfolge `$Log: unix8.tex,v $` – zweckmäßig im Kommentar am Anfang – so werden die Versionsinformationen auch dorthin übernommen. Dann hat man alles im Quellfile beisammen.
- Falls Sie sich mit `co -l filename` eine Kopie zum Editieren geholt und damit gleichzeitig das Original für weitere Schreibzugriffe gesperrt haben, anschließend die Kopie mit `rm(1)` löschen, so haben Sie nichts mehr zum Zurückstellen. In diesem Fall läßt sich die Sperre mit `rcs -u filename` aufheben. Besser ist es jedoch, auf die UNIX-Kommandos zu verzichten und nur mit den RCS-Kommandos zu arbeiten.

Das ist für den Anfang alles. Die RCS-Kommandos lassen sich in Makefiles verwenden. Die vom RCS vergebenen Zugriffsrechte können von UNIX-Kommandos (`chmod(1)`) überrannt werden, aber das ist nicht Sinn der Sache; der Einsatz von RCS setzt voraus, daß sich die Beteiligten diszipliniert verhalten.

Hier ein Makefile mit RCS-Kommandos für das nachstehende Sortierprogramm:

```
# makefile zu mysort.c, im RCS-System
# $Header: /home/debian/unix/quellen/sortmakef.tex,v 1.1.1.1 2005/02/06

CC = /bin/cc
CFLAGS = -Aa -DDEBUG

all: mysort clean

mysort: mysort.o bubble.o
        $(CC) $(CFLAGS) -o mysort mysort.o bubble.o

mysort.o: mysort.c myheader.h
        $(CC) $(CFLAGS) -c mysort.c

bubble.o: bubble.c myheader.h
        $(CC) $(CFLAGS) -c bubble.c

mysort.c: mysort.c,v
        co mysort.c

bubble.c: bubble.c,v
        co bubble.c

myheader.h: myheader.h,v
        co myheader.h

clean:
        /bin/rm  -f *.c *.o *.h makefile
```

Quelle 2.34 : Makefile zum Sortierprogramm mysort.c

Da dieses Beispiel sich voraussichtlich zu einer kleinen Familie von Quelltexten ausweiten wird, legen wir eine privates include-Datei mit unseren eigenen, für alle Teile gültigen Werten an:

```
/* myheader.h zum Sortierprogramm, RCS-Beispiel
   W. Alex, Universitaet Karlsruhe, 04. Juli 1995
*/

/*
$Header: /home/debian/unix/quellen/sortinclude,v 1.1.1.1 2005/02/06 20:0
*/

int bubble(char *text);
int insert(char *text);

#define  USAGE  "Aufruf: mysort filename"
#define  NOTEXIST  "File existiert nicht"
#define  NOTREAD  "File ist nicht lesbar"
#define  NOTSORT  "Problem beim Sortieren"

#define  LINSIZ  64           /* Zeilenlaenge */
```

```
#define  MAXLIN  256          /* Anzahl Zeilen */
```

Quelle 2.35 : Include-Datei zum Sortierprogramm mysort.c

Nun das Hauptprogramm, das die Verantwortung trägt, aber sonst nicht viel tut. Hier ist der Platzhalter
`$Header: /home/debian/unix/unix8.tex,v 1.1.1.1 2005/02/06 20:01:11 wulf Exp`
Bestandteil des Codes, die Versionsinformationen stehen also auch im ausführbaren Programm. Man könnte sogar mit ihnen etwas machen, ausgeben beispielsweise:

```
/* Sortierprogramm mysort, als Beispiel fuer RCS */

/*
$Log: mysort.tex,v $
Revision 1.1.1.1  2005/02/06 20:01:11  wulf
Anlegen des unix-Reps

*/

static char rcsid[] =
"$Header: /home/debian/unix/quellen/mysort.tex,v 1.1.1.1 2005/02/06 20:01:1

#include <stdio.h>
#include "myheader.h"

int main(int argc, char *argv[])
{
long time1, time2;

/* Pruefung der Kommandozeile */

if (argc != 2) {
    puts(USAGE); return(-1);
}

/* Pruefung des Textfiles */

if (access(argv[1], 0)) {
    puts(NOTEXIST); return(-2);
}

if (access(argv[1], 4)) {
    puts(NOTREAD); return(-3);
}

/* Sortierfunktion und Zeitmessung */

time1 = time((long *)0);

if (bubble(argv[1])) {
    puts(NOTSORT); return(-4);
}
```

```
time2 = time((long *)0);

/* Ende */

printf("Das Sortieren dauerte %ld sec.\n", time2 - time1);
return 0;
}
```

Quelle 2.36 : C-Programm Sortieren, für RCS

Hier die Funktion zum Sortieren (Bubblesort, nicht optimiert). Der einzige Witz in dieser Funktion ist, daß wir nicht die Strings durch Umkopieren sortieren, sondern nur die Indizes der Strings. Ansonsten kann man hier noch einiges verbessern und vor allem auch andere Sortieralgorithmen nehmen. Man sollte auch das Einlesen und die Ausgabe vom Sortieren trennen:

```
/* Funktion bubble() (Bubblesort), als Beispiel fuer RCS
   W. Alex, Universitaet Karlsruhe, 04. Juli 1995 */

/*
   $Header: /home/debian/unix/quellen/bubble.tex,v 1.1.1.1 2005/02/06 20
*/

#include &#60;stdio.h&#62;
#include &#60;string.h&#62;
#include "myheader.h"

int bubble(char *text)
{
int i = 0, j = 0, flag = 0, z, line[MAXLIN];
char array[MAXLIN][LINSIZ];
FILE *fp;

#if DEBUG
printf("Bubblesort %s\n", text);
#endif

/* Einlesen */

if ((fp = fopen(text, "r")) == NULL) return(-1);

while ((!feof(fp)) && (i < MAXLIN)) {
    fgets(array[i++], LINSIZ, fp);
}

fclose(fp);

#if DEBUG
puts("Array:");
j = 0;
while (j < i) {
    printf("%s", array[j++]);
}
puts("Ende Array");
```

```
#endif

/* Sortieren (Bubblesort) */

for (j = 0; j < MAXLIN; j++)
    line[j] = j;

while (flag == 0) {
  flag = 1;
  for (j = 0; j < i; j++) {
    if (strcmp(array[line[j]], array[line[j + 1]]) > 0) {
         z = line[j + 1];
         line[j + 1] = line[j];
         line[j] = z;
         flag = 0;
    }
  }
}

/* Ausgeben nach stdout */

#if DEBUG
puts("Array:");
j = 0;
while (j < i) {
    printf("%d\n", line[j++]);
}
puts("Ende Array");
#endif

j = 0;
while (j < i) {
    printf("%s", array[line[j++]]);
}

/* Ende */

return 0;
}
```

Quelle 2.37 : C-Funktion Bubblesort

Bubblesort eignet sich für kleine Sortieraufgaben mit bis zu etwa hundert Elementen. Kopieren Sie sich die Bausteine in ein eigenes Verzeichnis und entwickeln Sie das Programm unter Verwendung des RCS weiter. Näheres siehe `rcsintro(5)`.

Anfangs erscheint das Arbeiten mit RCS bei kleinen Projekten als lästig, ähnlich wie das Anlegen eines Makefiles. Man gewöhnt sich aber schnell daran und spart sofort das Eintragen des Änderungsdatums von Hand. Nach kurzer Zeit ist man für die selbst auferlegte Ordnung dankbar.

Das **Source Code Control System SCCS** verwaltet die Versionen der Module, indem es die erste Fassung vollständig speichert und dann jeweils die

Differenzen zur nächsten Version, während RCS die jüngste Version speichert und die älteren aus den Differenzen rekonstruiert.

Alle Versionen eines Programmes samt den Verwaltungsdaten werden in einer einzigen SCCS-Datei namens `s.filename` abgelegt, auf das schreibend nur über besondere SCCS-Kommandos zugegriffen werden kann. Das erste dieser Kommandos ist `admin(1)` und erzeugt aus einem C-Quellfile `program.c` das zugehörige SCCS-Dokument:

```
admin -iprogram.c s.program.c
```

Mit `admin(1)` lassen sich noch weitere Aufgaben erledigen, siehe Referenz-Handbuch. Mittels `get(1)` holt man das Quellfile wieder aus dem SCCS-Dokument heraus, mitttels `delta(1)` gibt man eine geänderte Fassung des Quellfiles an das SCCS-Dokument zurück.

RCS und SCCS arbeiten auf Datei-Ebene. Bei größeren Projekten ist es wünschenswert, mehrere Dateien gemeinsam oder ganze Verzeichnisse in die Versionsverwaltung einzubeziehen. Dies leistet das **Concurrent Versions System** (CVS). Es baut auf RCS auf und erweitert dessen Funktionalität außerdem um eine Client-Server-Architektur. Die beteiligten Dateien und Verzeichnisse können auf verschiedenen Computern im Netz liegen. Im Gegensatz zu RCS, das zu einem Zeitpunkt immer nur einem Benutzer das Schreiben gestattet, verfolgt CVS eine sogenannte optimistische Kooperationsstrategie. Mehrere Programmierer können gleichzeitig auf Kopien derselben Version (Revision) arbeiten. Beim Zurückschreiben wird ein Abgleich mit der in der zentralen Versionsbibliothek (Repository) abgelegten Fassung erzwungen, um zu verhindern, daß parallel durchgeführte und bereits zurückgeschriebene Versionen überschrieben werden. Diese Strategie kann zu Konflikten führen, die per Hand aufgelöst werden müssen. Während das Einrichten eines CVS-Projektes Überblick erfordert, ist das Arbeiten unter CVS nicht schwieriger als unter RCS. Einzelheiten wie so oft am einfachsten aus dem Netz, wo außer dem Programmpaket selbst auch kurze oder ausführliche, deutsche oder englische Anleitungen zu finden sind. Unter den Namen *WinCVS* und *MacCVS* liegen Fassungen für weitere Betriebssysteme im Netz.

Der Oberbegriff des ganzen Gebietes lautet *Software Configuration Management* (SCM) oder allgemeiner *Configuration Management* (CM). Lassen Sie einmal eine Suchmaschine darauf los, es gibt mehrere freie oder kommerzielle Produkte sowie Übersichten, Einführungen und Tutorials dazu.

Ist die Entwicklung einer Software oder eines Manuskriptes vorläufig abgeschlossen, geht es an die Pflege. Dazu gehört unter anderem das Management der im Betrieb der Software auftauchenden Probleme. Auch hierfür gibt es Werkzeuge, beispielsweise `gnats` aus dem GNU-Projekt. Aber das sprengt den Rahmen dieses Buches.

CASE bedeutet *Computer Aided Software Engineering*. An sich ist das nichts Neues, beim Programmieren hat man schon immer Computer eingesetzt. Das Neue bei CASE Tools wie SoftBench von Hewlett-Packard besteht darin, daß die einzelnen Programmierwerkzeuge wie syntaxgesteuerte Editoren, Compiler, Linker, Builder (`make(1)`), Analysewerkzeuge, Debugger, Versionskontrollsysteme sowie die Dokumentation unter einer einheitlichen, heutzutage grafischen Oberfläche – hier das X Window System und Motif - zusammengefaßt werden. Allgemein heißt das Ganze Programmier- oder Entwicklungsumgebung (Integrated development environment, IDE). Damit zu arbeiten ist die moderne Form des Programmierens und kann effektiv sein.

2.8.9 Systemaufrufe

2.8.9.1 Was sind Systemaufrufe?

Dem Programmierer stehen zwei Hilfsmittel[43] zur Verfügung, um seine Wünsche auszudrücken:

- die Schlüsselwörter (Wortsymbole) der Programmiersprache,
- die Systemaufrufe des Betriebssystems.

Die **Schlüsselwörter** (keyword, mot-clé) der Programmiersprache (C/C++, FORTRAN oder PASCAL) sind auch unter verschiedenen Betriebssystemen (PC-DOS, OS/2 oder UNIX) dieselben. Sie gehören zur Programmiersprache, das heißt zum Compiler. Die **Systemaufrufe** (system call, system primitive, fonction système) eines Betriebssystems (UNIX) sind für alle Programmiersprachen (C, FORTRAN, PASCAL, COBOL) dieselben. Sie gehören zum Betriebssystem. Man findet auch die Bezeichnung Kernschnittstellenfunktion, die besagt, dass ein solcher Aufruf sich unmittelbar an den Kern des Betriebssystems richtet. Der Kreis der Systemaufrufe liegt fest und kann nicht ohne Eingriffe in den Kern des Betriebssystems verändert werden. Da UNIX zum großen Teil in C geschrieben ist, sind die Systemaufrufe von UNIX C-Funktionen, die sich in ihrer Syntax nicht von eigenen oder fremden C-Funktionen unterscheiden. Deshalb müssen auch FORTRAN- oder PASCAL-Programmierer etwas von der Programmiersprache C verstehen. Im Handbuch werden die Systemaufrufe in Sektion (2) beschrieben.

Bei POSIX-konformen Betriebssystemen spricht man statt von Systemaufrufen besser von POSIX-Funktionen, da der POSIX-Standard offen lässt, ob diese vom Betriebssystem zur Verfügung gestellten Funktionen als Systemaufrufe oder als Bibliothek verwirklicht sind. Auf jeden Fall gehören sie zum Betriebssystem, nicht zum Compiler. Die Unterscheidung spielt eine Rolle, wenn man für verschiedene Betriebssysteme und/oder Compiler programmiert. Der Programmierer muss wissen, woher seine Funktionen stammen.

In Sektion (3) finden sich vorgefertigte **Unterprogramme**, **Subroutinen** oder **Standardfunktionen** (standard function, fonction élémentaire) für häufig vorkommende Aufgaben. Für den Anwender besteht kein Unterschied zu den Systemaufrufen. Streng genommen gehören diese Standardfunktionen jedoch zu den jeweiligen Programmiersprachen (zum Compiler) und nicht zum Betriebssystem. Der Kreis der Standardfunktionen ist beliebig ergänzbar. Um den Benutzer zu verwirren, sind die Systemaufrufe und die Standardfunktionen in *einer* Funktionsbibliothek (`/lib/libc.a` und andere) vereinigt.

[43] Standardfunktionen sind erst verfügbar, nachdem andere Programmierer sie geschrieben haben.

Die Aufgabenverteilung zwischen Schlüsselwörtern, Systemaufrufen und Standardfunktionen ist in gewissem Umfang willkürlich. Systemaufrufe erledigen Aufgaben, die aus dem Aufbau und den kennzeichnenden Eigenschaften des Betriebssystems herrühren, bei UNIX also in erster Linie

- Ein- und Ausgabe auf unterster Stufe,
- Umgang mit Prozessen,
- Umgang mit dem Datei-System,
- Sicherheitsvorkehrungen.

Das Öffnen einer Datei zum Lesen oder Schreiben ist Sache eines Systemaufrufs (`open(2)`), Sortieren hingegen Sache einer Standardfunktion (`qsort(3)`). Es gibt aber zusätzlich auch Standardfunktionen zum Umgang mit Dateien, die den jeweiligen Systemaufruf komfortabel verpacken (`fopen(3)`). Nach außen definiert die Menge der Systemaufrufe das Betriebssystem. Zwei Systeme, die in ihren Aufrufen übereinstimmen, sind für den Benutzer identisch. Neue Funktionalitäten des Betriebssystems stellen sich dem Programmierer als neue Systemaufrufe dar, siehe zum Beispiel unter `stream(2)`.

Einige UNIX-Systemaufrufe haben gleiche oder ähnliche Aufgaben wie Shell-Kommandos. Wenn man die Zeit wissen möchte, verwendet man im Dialog das Shell-Kommando `date(1)`. Will man diese Information aus einem eigenen Programm heraus abfragen, greift man auf den Systemaufruf `time(2)`[44] zurück. Das Shell-Kommando ist ein in ein C-Programm verpackter Systemaufruf.

In Linux/UNIX sind Systemaufrufe **Funktionen** der Programmiersprache C. Eine Funktion übernimmt beim Aufruf Argumente oder Parameter und gibt ein Ergebnis zurück. Dieser Mechanismus wird **Parameterübergabe** genannt. Man muss ihn verstanden haben, um Funktionen in eigenen Programmen verwenden zu können. Eine Erklärung findet sich im Skriptum *Programmieren in C/ C++*.

2.8.9.2 Beispiel Systemzeit (time)

Im folgenden Beispiel wird der Systemaufruf `time(2)` verwendet. `time(2)` liefert die Zeit in Sekunden seit 00:00:00 Greenwich Mean Time, 1. Januar 1970. Computeruhren laufen übrigens erstaunlich ungenau, falls sie nicht durch eine Funkuhr oder über das Netz synchronisiert werden. Ferner brauchen wir die Standardfunktion `gmtime(3)`, Beschreibung unter `ctime(3)`, die aus den obigen Sekunden eine Struktur erzeugt, die Datum und Uhrzeit enthält. Die Umrechnung von Greenwich auf Karlsruhe nehmen wir selbst vor. Eleganter wäre ein Rückgriff auf die Zeitzonen-Variable der Umgebung. Laut Referenz-Handbuch hat `time(2)` die Syntax

```
long time ((long *) 0)
```

[44] In HP-UX. In ANSI-C ist eine Standardfunktion `time(3)` enthalten.

Die Funktion verlangt ein Argument vom Typ Pointer auf long integer, und zwar im einfachsten Fall den Nullpointer. Der Returnwert ist vom Typ long integer. Der größte Wert dieses Typs liegt etwas über 2 Milliarden. Damit läuft diese Uhr etwa 70 Jahre. Die Subroutine `gmtime(3)` hat die Syntax

```
#include <time.h>
struct tm *gmtime(clock)
long *clock
```

Die Funktion `gmtime(3)` verlangt ein Argument `clock` vom Typ Pointer auf long integer. Wir müssen also den Returnwert von `time(2)` in einen Pointer umwandeln (referenzieren). Der Rückgabewert der Funktion `gmtime(3)` ist ein Pointer auf eine Struktur namens `tm`. Diese Struktur ist in der include-Datei `time.h` definiert. Die include-Dateien sind lesbarer Text; es ist ratsam hineinzuschauen. In der weiteren Beschreibung zu `ctime(3)` wird die Struktur `tm` erläutert:

```
struct tm {
       int tm_sec;        /* seconds (0 - 59) */
       int tm_min;        /* minutes (0 - 59) */
       int tm_hour;       /* hours (0 - 23) */
       int tm_mday;       /* day of month (1 - 31) */
       int tm_mon;        /* month of year (0 - 11) */
       int tm_year;       /* year - 1900 */
       int tm_wday;       /* day of week (sunday = 0) */
       int tm_yday;       /* day of year (0 - 365) */
       int tm_isdst;      /* daylight saving time */
}
```

Von den beiden letzten Komponenten der Struktur machen wir keinen Gebrauch. Da die Komponenten alle vom selben Typ sind, ist statt der Struktur auch ein Array denkbar. Vermutlich wollte sich der Programmierer den Weg offenhalten, künftig auch andere Typen aufzunehmen (Zeitzone). Das Programm, das die Quelle zu dem Kommando `zeit` aus der ersten Übung ist, sieht folgendermaßen aus:

```
/* Ausgabe der Zeit auf Bildschirm */
/* Compileraufruf cc -o zeit zeit.c */

#include <stdio.h>
#include <time.h>

char *ptag[] = {"Sonntag,   ", "Montag,    ",
                "Dienstag,  ", "Mittwoch,  ",
                "Donnerstag,", "Freitag,   ",
                "Samstag,   "};
char *pmon[] = {"Januar", "Februar", "Maerz", "April",
                "Mai", "Juni", "Juli", "August",
                "September", "Oktober", "November",
                "Dezember"};
```

```
main()
{
long sec, time();
struct tm *gmtime(), *p;

sec = time((long *) 0) + 3600;    /* MEZ = GMT + 3600 */
p = gmtime(&sec);
printf("%s %d. ", ptag[p->tm_wday], p->tm_mday);
printf("%s %d     ", pmon[p->tm_mon], p->tm_year +1900);
printf("%d:%02d MEZ\n", p->tm_hour, p->tm_min);
}
```

Quelle 2.38 : C-Programm zur Anzeige der Systemzeit

Nun wollen wir dieselbe Aufgabe mit einem FORTRAN-Programm bewältigen. Der UNIX-Systemaufruf `time(2)` bleibt, für die C-Standardfunktion `gmtime(3)` suchen wir die entsprechende FORTRAN-Routine. Da wir keine finden, müssen wir sie entweder selbst schreiben (was der erfahrene Programmierer scheut) oder nach einem Weg suchen, eine beliebige C-Standardfunktion in ein FORTRAN-Programm hineinzuquetschen.

Der Systemaufruf `time(2)` macht keinen Kummer. Er benötigt ein Argument vom Typ Pointer auf long integer, was es in FORTRAN gibt. Der Rückgabewert ist vom Typ long integer, auch kein Problem. Die C-Standardfunktion `gmtime(3)` erwartet ein Argument vom Typ Pointer auf long integer, was machbar wäre, aber ihr Ergebnis ist ein Pointer auf eine Struktur. Das hat FORTRAN noch nie gesehen[45]. Deshalb weichen wir auf die C-Standardfunktion `ctime(3)` aus, deren Rückgabewert vom Typ Pointer auf character ist, was es in FORTRAN näherungsweise gibt. In FORTRAN ist ein Zeichen ein String der Länge eins. Strings werden per Deskriptor übergeben. Ein **String-Deskriptor** ist der Pointer auf das erste Zeichen *und* die Anzahl der Zeichen im String als Integerwert. Das Programm sieht dann so aus:

```
      program zeit

$ALIAS foratime = 'sprintf' c

      integer*4 time, tloc, sec, ctime
      character atime*26

      sec = time(tloc)

      call foratime(atime, '%s'//char(0), ctime(sec))
      write(6, '(a)') atime

      end
```

Quelle 2.39 : FORTRAN-Programm zur Anzeige der Systemzeit

Die **ALIAS-Anweisung** ist als Erweiterung zu FORTRAN 77 in vielen

[45]FORTRAN 90 kennt Strukturen.

Compilern enthalten und dient dazu, den Aufruf von Unterprogrammen anderer Sprachen zu ermöglichen. Der Compiler weiß damit, dass das Unterprogramm außerhalb des Programms – zum Beispiel in einer Bibliothek – einen anderen Namen hat als innerhalb des Programms. Wird eine Sprache angegeben (hier C), so erfolgt die Parameterübergabe gemäß der Syntax dieser Sprache. Einzelheiten siehe im Falle unserer Anlage im HP FORTRAN 77/HP-UX Reference Manual im Abschnitt *Compiler Directives*.

Die Anweisung teilt dem Compiler mit, dass hinter der FORTRAN-Subroutine `foratime` die C-Standard-Funktion `sprintf(3)` steckt und dass diese nach den Regeln von C behandelt werden soll. Der Rückgabewert von `sprintf(3)` (die Anzahl der ausgegebenen Zeichen) wird nicht verwertet, deshalb ist `foratime` eine FORTRAN-Subroutine (keine Funktion), die im Programm mit `call` aufgerufen werden muss.

Der Systemaufruf `time(2)` verlangt als Argument einen Pointer auf `long integer`, daher ist `tloc` als vier Bytes lange Integerzahl deklariert. `tloc` spielt weiter keine Rolle. Die Übergabe als Pointer (by reference) ist in FORTRAN Standard für Zahlenvariable und braucht nicht eigens vereinbart zu werden. Der Rückgabewert von `time` geht in die Variable `sec` vom Typ `long integer` = `integer*4`.

Die `call`-Zeile ruft die Subroutine `foratime` alias C-Funktion `sprintf(3)` auf. Diese C-Funktion erwartet drei Argumente: den Ausgabestring als Pointer auf `char`, einen Formatstring als Pointer auf `char` und die auszugebende Variable von einem Typ, wie er durch den Formatstring bezeichnet wird. Der Rückgabewert der Funktion `ctime(3)` ist ein Pointer auf `char`. Da dies kein in FORTRAN zulässiger Typ ist, deklarieren wir die Funktion ersatzweise als vom Typ 4-Byte-integer. Der Pointer lässt sich auf jeden Fall in den vier Bytes unterbringen. Nach unserer Erfahrung reichen auch zwei Bytes, ebenso funktioniert der Typ `logical`, nicht jedoch `real`.

Der Formatstring besteht aus der Stringkonstanten `%s`, gefolgt von dem ASCII-Zeichen Nr. 0, wie es bei Strings in C Brauch ist. Für `sprintf(3)` besagt dieser Formatstring, das dritte Argument – den Rückgabewert von `ctime(3)` – als einen String aufzufassen, das heißt als Pointer auf das erste Element eines Arrays of characters.

`atime` ist ein FORTRAN-String-Deskriptor, dessen erste Komponente ein Pointer auf character ist. Damit weiß `sprintf(3)`, wohin mit der Ausgabe. Die `write`-Zeile ist wieder pures FORTRAN.

An diesem Beispiel erkennen Sie, dass Sie auch als FORTRAN- oder PASCAL-Programmierer etwas von C verstehen müssen, um die Systemaufrufe und C-Standardfunktionen syntaktisch richtig zu gebrauchen.

Bei manchen FORTRAN-Compilern (Hewlett-Packard, Microsoft) lassen sich durch einen einfachen **Interface-Aufruf** Routinen fremder Sprachen so verpacken, dass man sie übernehmen kann, ohne sich um Einzelheiten kümmern zu müssen.

2.8.9.3 Beispiel Datei-Informationen (access, stat, open, close)

In einem weiteren Beispiel wollen wir mithilfe von Systemaufrufen Informationen über eine Fatei gewinnen, dazu noch eine Angabe aus der Sitzungsumgebung. Die Teile des Programms lassen sich einfach in andere C-Programme übernehmen.

Dieses Programm soll beim Aufruf (zur Laufzeit, in der Kommandozeile) den Namen der Datei als Argument übernehmen, wie wir es von UNIX-Kommandos her kennen. Dazu ist ein bestimmter Formalismus vorgesehen:

```
int main(int argc, char *argv[], char *envp[])
```

Die Funktion `main()` übernimmt die Argumente `argc`, `argv` und gegebenenfalls `envp`. Das Argument `argc` ist der **Argument Counter**, eine Ganzzahl. Sie ist gleich der Anzahl der Argumente in der Kommandozeile beim Aufruf des Programms. Das Kommando selbst ist das erste Argument, also hat `argc` mindestens den Wert 1. Das Argument `argv` ist der **Argument Vector**, ein Array of Strings, also ein Array of Arrays of Characters. Der erste String, Index 0, ist das Kommando; die weiteren Strings sind die mit dem Kommando übergebenen Argumente, hier der Name der gefragten Datei. Der **Environment Pointer** `envp` wird nur benötigt, falls man Werte aus der Umgebung abfragt. Es ist wie `argv` ein Array of Strings. Die Namen `argc`, `argv` und `envp` sind willkürlich, aber üblich. Typ und Reihenfolge sind vorgegeben.

Die Umgebung besteht aus Strings (mit Kommando `set (Shell)` anschauen). In der `for`-Schleife werden die Strings nacheinander mittels der Funktion `strncmp(3)` (siehe `string(3)`) mit dem String LOGNAME verglichen. Das Ergebnis ist der Index `i` des gesuchten Strings im Array `envp[]`.

Den Systemaufruf `access(2)` finden wir in der Sektion (2) des Referenz-Handbuches. Er untersucht die Zugriffsmöglichkeiten auf eine Datei und hat die Syntax:

```
int access(char *path, int mode)
```

Der Systemaufruf erwartet als erstes Argument einen String, nämlich den Namen der Datei. Wir werden hierfür `argv[1]` einsetzen. Als zweites steht eine Ganzzahl, die die Art des gefragten Zugriffs kennzeichnet. Falls der gefragte Zugriff möglich ist, liefert `access(2)` den Wert null zurück, der in einem C-Programm zugleich die Bedeutung von logisch falsch (FALSE) hat und deshalb in den `if`-Zeilen negiert wird.

Den Systemaufruf `stat(2)` finden wir ebenfalls in Sektion 2. Er ermittelt Dateiinformationen aus der **Inode** und hat die Syntax

```
#include <sys/types.h>
#include <sys/stat.h>

int stat(path, buf)
char *path;
struct stat *buf;
```

Sein erstes Argument ist wieder der Dateiname, das zweite der Name eines Puffers zur Aufnahme einer Struktur, die die Informationen enthält. Diese Struktur vom Typ `stat` ist in der include-Datei `/usr/include/sys/stat.h` deklariert, das seinerseits Bezug nimmt auf Deklarationen in `/usr/include/types.h`. Auch einige Informationen wie `S_IFREG` sind in `sys/stat.h` definiert. Die Zeitangaben werden wie im vorigen Abschnitt umgerechnet.

In UNIX-Datei-Systemen enthält jede Datei am Anfang eine **Magic Number**, die über die Art der Datei Auskunft gibt (`man magic`). Mittels des Systemaufrufs `open(2)` wird die fragliche Datei zum Lesen geöffnet, mittels `lseek(2)` der Lesezeiger auf die Magic Number gesetzt und mittels `read(2)` die Zahl gelesen. Der Systemaufruf `close(2)` schließt die Datei wieder. Die Systemaufrufe findet man unter ihren Namen in Sektion (2), eine Erläuterung der Magic Numbers unter `magic(4)`. Nun das Programm:

```
/* Informationen ueber eine Datei */

#define  MEZ 3600

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <time.h>
#include <fcntl.h>
#include <magic.h>

void exit(); long lseek();

int main(argc, argv, envp)

    int argc; char *argv[], *envp[];

{

int i, fildes;
struct stat buffer;
long asec, msec, csec;
struct tm *pa, *pm, *pc;

if (argc < 2) {
    puts("Dateiname fehlt"); return (-1);
}

/* Informationen aus dem Environment */

for (i = 0; envp[i] != NULL; i++)
    if (!(strncmp(envp[i], "LOGNAME", 4)))
        printf("\n%s\n", envp[i]);

/* Informationen mittels Systemaufruf access(2) */

printf("\nFile heisst: %8s\n", argv[1]);
```

```
if (!access(argv[1], 0))
    puts("File existiert");
else
    puts("File existiert nicht");

if (!access(argv[1], 1))
    puts("File darf ausgefuehrt werden");
else
    puts("File darf nicht ausgefuehrt werden");

if (!access(argv[1], 2))
    puts("File darf beschrieben werden");
else
    puts("File darf nicht beschrieben werden");

if (!access(argv[1], 4))
    puts("File darf gelesen werden");
else
    puts("File darf nicht gelesen werden");

/* Informationen aus der Inode, Systemaufruf stat(2) */

if (!(stat(argv[1], &buffer))) {
    printf("\nDevice:          %ld\n", buffer.st_dev);
    printf("Inode-Nr.:       %lu\n", buffer.st_ino);
    printf("File Mode:       %hu\n\n", buffer.st_mode);

    switch(buffer.st_mode & S_IFMT) {
        case S_IFREG:
            {
            puts("File ist regulaer");
            break;
            }
        case S_IFDIR:
            {
            puts("File ist ein Verzeichnis");
            break;
            }
        case S_IFCHR:
        case S_IFBLK:
        case S_IFNWK:
            {
            puts("File ist ein Special File");
            break;
            }
        case S_IFIFO:
            {
            puts("File ist eine Pipe");
            break;
            }
        default:
            {
            puts("Filetyp unbekannt (Inode)");
            }
    }
```

```
    printf("\nLinks:           %hd\n", buffer.st_nlink);
    printf("Owner-ID:        %hu\n", buffer.st_uid);
    printf("Group-Id:        %hu\n", buffer.st_gid);
    printf("Device-ID:       %ld\n", buffer.st_rdev);
    printf("Filegroesse:     %ld\n", buffer.st_size);

    asec = buffer.st_atime + MEZ; pa = gmtime(&asec);
    msec = buffer.st_mtime + MEZ; pm = gmtime(&msec);
    csec = buffer.st_ctime + MEZ; pc = gmtime(&csec);

    printf("Letzter Zugriff: %d. %d. %d\n",
            pa->tm_mday, pa->tm_mon + 1, pa->tm_year);
    printf("Letzte Modifik.: %d. %d. %d\n",
            pm->tm_mday, pm->tm_mon + 1, pm->tm_year);
    printf("Letzte Stat.Ae.: %d. %d. %d\n",
            pc->tm_mday, pc->tm_mon + 1, pc->tm_year);

}
else
    puts("Kein Zugriff auf Inode");

/* Pruefung auf Text oder Code (magic number) */
/* Systemaufrufe open(2), lseek(2), read(2), close(2) */
/* Magic Numbers siehe magic(4) */

{
    MAGIC   magbuf;

    fildes = open(argv[1], O_RDONLY);
    if (lseek(fildes, MAGIC_OFFSET, 0) >= (long)0) {
        read(fildes, &magbuf, sizeof magbuf);
        switch(magbuf.file_type) {
            case RELOC_MAGIC:
                {
                puts("File ist relocatable");
                break;
                }
                case EXEC_MAGIC:
                case SHARE_MAGIC:
                case DEMAND_MAGIC:
                {
                puts("File ist executable");
                break;
                }
                case DL_MAGIC:
                case SHL_MAGIC:
                {
                puts("File ist Library");
                break;
                }
            default:
                puts("Filetyp unbekannt (Magic Number)");
                lseek(fildes, 0L, 0);
        }
    }
```

```
    else {
        puts("Probleme mit dem Filepointer");
    }

}
close(fildes);
}
```

Quelle 2.40 : C-Programm zum Abfragen von Informationen über eine Datei

Die Verwendung von Systemaufrufen oder Standardfunktionen in C-Programmen ist nicht schwieriger als der Gebrauch anderer Funktionen. Man muss sich nur an die im Referenz-Handbuch Sektionen (2) und (3) nachzulesende Syntax halten. Es empfiehlt sich, die genannten Sektionen einmal durchzublättern, um eine Vorstellung davon zu gewinnen, wofür es Systemaufrufe und Standardfunktionen gibt. Die Ausgabe des Programms sieht folgendermaßen aus:

```
LOGNAME=wualex1

Datei heisst:        a.out
Datei existiert
Datei darf ausgefuehrt werden.
Datei darf nicht beschrieben werden.
Datei darf gelesen werden.

Device:             13
Inode-Nr.:          43787
File Mode:          33216

Datei ist regulaer

Links:              1
Owner-ID:           101
Group-ID:           20
Device-ID:          102536
Dateigroesse:        53248
Letzter Zugriff:    24. 1. 91
Letzte Modifik.:    24. 1. 91
Letzte Stat.Ae.:    24. 1. 91
Datei ist executable
```

Die Bedeutung von `File Mode` finden Sie bei `mknod(2)`. Es handelt sich um ausführliche Informationen über die Zugriffsrechte usw. Ähnliche Auskünfte über eine Datei liefert das Kommando `chatr(1)`.

2.8.9.4 Beispiel Prozesserzeugung (exec, fork)

Zunächst ein kleines, aber fieses Programm namens `forkbomb`, mit dem man die Robustheit seines Systems auf die Probe stellen kann.

```
(kommt demnaechst)
```

Quelle 2.41 : C-Programm zum Erzeugen vieler Prozesse (Fork-Bombe)

Der Systemaufruf `fork(2)` erzeugt eine Kopie des aufrufenden Prozesses mit einer neuen Prozess-ID. Im Beispiel wird `fork(2)` in einer ewigen `for`-Schleife aufgerufen.

2.8.10 Begriffe Programmer's Workbench

Folgende Begriffe sollten klarer geworden sein:

- Archiv, Bibliothek
- Builder (`make`)
- Debugger
- Linker
- Profiler
- Quelltext oder -code, ausführbarer Code
- Übersetzer, Compiler, Interpreter
- Systemaufruf – Standardfunktion
- Versionsverwaltung

Folgende Kommandos sollten beherrscht werden:

- `cc` oder `gcc`
- `lint`
- Anfänge von `make`
- Anfänge von RCS (`ci`, `co`)

2.8.11 Memo Programmer's Workbench

- Die Programmquellen werden mit einem Editor geschrieben.
- Mit dem Syntaxprüfer `lint(1)` läßt sich die syntaktische Richtigkeit von C-Programmen prüfen, leider nicht die von C++-Programmen.
- Schon bei kleinen Programmierprojekten ist das Werkzeug `make(1)` dringend zu empfehlen. Der Compileraufruf vereinfacht sich wesentlich. Auch für Texte verwendbar.
- Mit einem Compiler wird der Quellcode in den Maschinencode des jeweiligen Prozessors übersetzt.
- Der schwerste Hammer bei der Fehlersuche ist ein Debugger, lernbedürftig, aber nicht immer vermeidbar.
- Programmfunktionen (aber auch andere Dateien) lassen sich in Bibliotheken archivieren, die bequemer zu handhaben sind als eine Menge von einzelnen Funktionen.

- Bei größeren Projekten kommt man nicht um ein Kontrollsystem wie RCS oder CVS herum, vor allem dann, wenn mehrere Personen beteiligt sind. Das Lernen kostet Zeit, die aber beim Ringen mit dem Chaos mehr als wettgemacht wird.
- CASE-Tools vereinigen die einzelnen Werkzeuge unter einer gemeinsamen Benutzeroberfläche. Der Programmierer braucht gar nicht mehr zu wissen, was ein Compiler ist.
- Systemaufrufe sind die Verbindungen des Betriebssystems nach oben, zu den Anwendungsprogrammen hin. Sie sind Teil des Betriebssystems.
- Systemaufrufe haben vorwiegend mit Prozessen, den Datei-Systemen und der Ein- und Ausgabe zu tun.
- UNIX-Systemaufrufe sind C-Funktionen, die sich im Gebrauch nicht von anderen C-Funktionen unterscheiden.
- C-Standardfunktionen gehören zum C-Compiler, nicht zum Betriebssystem.
- Ein FORTRAN-Programmierer auf einem UNIX-System ist auf die UNIX-Systemaufrufe angewiesen, nicht aber auf die C-Standardfunktionen (dafür gibt es FORTRAN-Standardfunktionen). Dasselbe gilt für jede andere Programmiersprache.

2.8.12 Übung Programmer's Workbench

Anmelden wie gewohnt. Zum Üben brauchen wir ein kleines Programm mit bestimmten Fehlern. Legen Sie mit `mkdir prog` ein Unterverzeichnis `prog` an, wechseln Sie mit `cd prog` dorthin und geben Sie mit `vi fehler.c` folgendes C-Programm (ohne den Kommentar) unter dem Namen `fehler.c` ein:

```
/* Uebungsprogramm mit mehreren Fehlern */

/* 1. Fehler: Es wird eine symbolische Konstante PI
   definiert, die nicht gebraucht wird. Dieser Fehler
   hat keine Auswirkungen und wird von keinem
   Programm bemerkt.
   2. Fehler: Eine Ganzzahl-Variable d wird deklariert,
   aber nicht gebraucht. Dieser Fehler hat keine
   Auswirkungen, wird aber von lint beanstandet.
   3. Fehler: Die Funktion scanf verlangt Pointer als
   Argument, es muss &a heissen. Heimtueckischer
   Syntaxfehler. lint gibt eine irrefuehrende Warnung
   aus, der Compiler merkt nichts. Zur Laufzeit ein
   memory fault.
   4. Fehler: Es wird durch nichts verhindert, dass fuer
   b eine Null eingegeben wird. Das kann zu einem
   Laufzeitfehler fuehren, wird weder von lint noch
   vom Compiler bemerkt.
   5. Fehler: Es sollte die Summe ausgerechnet werden,
   nicht der Quotient. Logischer Fehler, wird weder
```

```
   von lint noch vom Compiler bemerkt.
   6. Fehler: Abschliessende Klammer fehlt. Syntaxfehler,
   wird von lint und Compiler beanstandet.

   Darueberhinaus spricht lint noch Hinweise bezueglich
   main, printf und scanf aus. Diese Funktionen sind
   aber in Ordnung, Warnungen ueberhoeren.  */

#define PI 3.14159
#include <stdio.h>

int main()
{

    int a, b, c, d;

    puts("Bitte 1. Summanden eingeben: ");
    scanf("%d", a);
    puts("Bitte 2. Summanden eingeben: ");
    scanf("%d", &b);
    c = a / b;
    printf("Die Summe ist: %d\n", c);
```

Quelle 2.42 : C-Programm mit Fehlern

Als erstes lassen wir den Syntaxprüfer `lint(1)` auf das Programm los:

```
lint fehler.c
```

und erhalten das Ergebnis:

```
fehler.c
==============
(36)  warning: a may be used before set
(41)  syntax error
(41)  warning: main() returns random value to environment

==============
function returns value which is always ignored
    printf     scanf
```

Zeile 41 ist das Programmende, dort steckt ein Fehler. Die Warnungen sind nicht so dringend. Mit dem `vi(1)` ergänzen wir die fehlende geschweifte Klammer am Schluß. Der Fehler hätte uns eigentlich nicht unterlaufen dürfen, da der `vi(1)` eine Hilfe zur Klammerprüfung bietet (Prozentzeichen). Neuer Lauf von `lint(1)`:

```
fehler.c
==============
(36)  warning: a may be used before set
(33)  warning: d unused in function main
(41)  warning: main() returns random value to environment
```

```
==============
function returns value which is always ignored
    printf     scanf
```

Wir werfen die überflüssige Variable d in der Deklaration heraus. Nochmals `lint(1)`.

```
fehler.c
==============
(36)  warning: a may be used before set
(41)  warning: main() returns random value to environment

==============
function returns value which is always ignored
    printf     scanf
```

Jetzt ignorieren wir die Warnung von `lint(1)` bezüglich der Variablen a (obwohl heimtückischer Fehler, aber das ahnen wir noch nicht). Wir lassen kompilieren und rufen das kompilierte Programm `a.out(4)` auf:

```
cc fehler.c
a.out
```

Der Compiler hat nichts zu beanstanden. Ersten Summanden eingeben, Antwort: `memory fault` oder `Bus error - core dumped`. Debugger[46] einsetzen, dazu nochmals mit der Option -g und dem vom Debugger verwendeten Objektfile `/usr/lib/xdbend.o` kompilieren und anschließend laufen lassen, um einen aktuellen Speicherauszug (Coredump) zu erzeugen:

```
cc -g fehler.c /usr/lib/xdbend.o
chmod 700 a.out
a.out
xdb
```

Standardmäßig greift der Debugger auf die ausführbare Datei `a.out(4)` und das beim Zusammenbruch erzeugte Corefile `core(4)` zurück. Er promptet mit >. Wir wählen mit der Eingabe `s` Einzelschritt-Ausführung. Mehrmals mit RETURN weitergehen, bis Aufforderung zur Eingabe von a kommt (kein Prompt). Irgendeinen Wert für a eingeben. Fehlermeldung des Debuggers `Bus error`. Wir holen uns weitere Informationen vom Debugger:

```
T   (stack viewing)
s   (Einzelschritt)
q   (quit)
```

Nachdem wir wissen, daß der Fehler nach der Eingabe von a auftritt, schauen wir uns die Zeile mit `scanf( ..., a)` an und bemerken, daß wir der Funktion `scanf(3)` eine Variable statt eines Pointers übergeben haben (`man scanf`

[46]Real programmers can read core dumps.

oder im Anhang nachlesen). Wir ersetzen also `a` durch `&a`. Das Compilieren erleichtern wir uns durch `make(1)`. Wir schreiben eine Datei namens `makefile` mit folgenden Zeilen:

```
fehler: fehler.c
    cc fehler.c -o fehler
```

und rufen anschließend nur noch das Kommando `make(1)` ohne Argumente auf. Das Ergebnis ist ein lauffähiges Programm mit Namen `fehler`. Der Aufruf von `fehler` führt bei sinnvollen Eingaben zu einer Ausgabe, die richtig sein könnte. Wir haben aber noch einen Denkfehler darin. Statt der Summe wird der Integer-Quotient berechnet. Wir berichtigen auch das und testen das Programm mit einigen Eingaben. Da unser Quelltext richtig zu sein scheint, verschönern wir seine vorläufig endgültige Fassung mit dem Beautifier `cb(1)`:

```
cb fehler.c > fehler.b
rm fehler.c
mv fehler.b fehler.c
```

Schließlich löschen wir das nicht mehr benötigte Corefile und untersuchen das Programm noch mit einigen Werkzeugen:

```
time fehler
cflow fehler.c
cxref fehler.c
strings fehler
nm fehler
size fehler
ls -l fehler
strip fehler
ls -l fehler
```

`strings(1)` ist ein ziemlich dummes Werkzeug, das aus einer ausführbaren Datei alles heraussucht, was nach String aussieht. Das Werkzeug `nm(1)` gibt eine Liste aller Symbole aus, die lang werden kann. `strip(1)` wirft aus einer ausführbaren Datei die nur für den Debugger, nicht aber für die Ausführung wichtigen Informationen heraus und verkürzt dadurch die Datei. Abmelden mit `exit`.

Schreiben Sie in einer Programmiersprache Ihrer Wahl (ich empfehle C) ein Programm, das

- eine Datei mittels `creat(2)` erzeugt,
- dessen Zugriffsrechte mittels `chmod(2)` und seine Zeitstempel mittels `utime(2)` setzt,
- die verwendeten Werte mittels `fprintf(3)` als Text in die Datei schreibt. `fprintf(3)` finden Sie unter `printf(3)`.

Schreiben Sie ein Programm ähnlich `who(1)`. Sie brauchen dazu `getut(3)` und `utmp(4)`.

2.8.13 Fragen Programmer's Workbench

- Wozu braucht man einen Compiler? Einen Linker?
- Was ist `lint`?
- Was macht `make`? Wie sieht ein einfaches Makefile aus?
- Wozu braucht man Debugger?
- Was ist eine Funktionsbibliothek? Vorteil?
- Wozu braucht man eine Versionsverwaltung? Wie benutzt man RCS?
- Was sind Systemaufrufe? Wer braucht sie?
- Unterschied zu Standardfunktionen?
- Welche Aufgaben erledigen die Systemaufrufe hauptsächlich?

2.9 L'atelier graphique

2.9.1 Übersicht

Ein computer-unterstützter Student (CAS) braucht:

- das Verständnis einiger Grundbegriffe.
- ein Werkzeug zur grafischen Darstellung von Versuchsdaten,
- ein Werkzeug zum Zeichnen,
- ein Werkzeug zum Bearbeiten von Fotos.

Architekten, Bauingenieure, Elektrotechniker oder Maschinenbauer wünschen sich darüber hinaus Werkzeuge zum Konstruieren am Bildschirm. Das Erstellen von Computerspielen oder Trickfilmen sprengt den Rahmen des Manuskripts und wird nicht behandelt.

2.9.2 Was heißt Grafik?

Grafische Informationen sind **sichtbare Informationen**, die *nicht* aus Zeichen bestehen. Zeichen sind – wie wir im Abschnitt 2.15 *Exkurs über Informationen* auf Seite 282 lesen – Elemente aus einem zwischen Sender und Empfänger vereinbarten Zeichenvorrat. In der Grafik gibt es keinen vereinbarten Vorrat an Elementen, sondern einfache Grundelemente, aus denen komplexe Grafiken aufgebaut werden: **Punkte, Linien, Flächen, Körper**. Dazu kommen Komponenten wie **Licht und Schatten, Farbe, Kontrast, Perspektive, Raum, Bewegung**. Die Vielfalt grafischer Objekte und der zugehörigen Operationen ist theoretisch unbeschränkt. Bei der Leistungsfähigkeit heutiger Hardware heißt Grafik Darstellung komplizierter farbiger dreidimensionaler Objekte oder Modelle (Modeling) – auf Bildschirm oder Papier – aus verschiedenen Blickwinkeln betrachtet, mit strukturierten Oberflächen und unterschiedlicher Beleuchtung (Rendering), gegebenenfalls in Bewegung

(Animation). Die Computer-Grafik ist ein eigenes, umfangreiches Wissensgebiet geworden; der Benutzer braucht einige Grundlagen und Vorkenntnisse, um zielgerichtet arbeiten zu können.

Die Bearbeitung grafischer oder akustischer Daten mit durchschnittlicher Hardware ist inzwischen möglich, aber über das Wie und Womit besteht noch lange keine Einigkeit. Auch über die Schnittstelle der Werkzeuge zum Menschen ist noch nicht alles gesagt, während bei den Zeichen die mechanische Schreibmaschine Vorarbeit geleistet hat. In Linux-Distributionen sind viele Grafikwerkzeuge enthalten, und es werden laufend mehr.

Es gibt in UNIX keine **Standardprogramme** für die Bearbeitung grafischer Daten analog etwa zu den Werkzeugen für die Bearbeitung alphanumerischer Daten und keine **Standard-Bibliotheken** mit Grafik-Funktionen. Selbstverständlich kann man unter UNIX Grafik machen, aber man braucht dazu zusätzliche Programme und Bibliotheken, die nicht standardisiert sind und gelegentlich Geld kosten. Mühe bereitet auch der schnelle Wechsel der Grafikpakete; man kommt mit dem Lernen kaum nach. Das Gleiche gilt für die Bearbeitung akustischer Daten, was technisch möglich, bisweilen wünschenswert, aber weit entfernt von jeder Standardisierung ist. Die Ursachen hierfür sind:

- Als UNIX begann, war die Hardware noch zu leistungsschwach für die Bearbeitung grafischer Daten (langsame CPUs, kleine Arbeitsspeicher, serielle, auf Zeichen begrenzte Terminals). Deshalb waren grafische Werkzeuge – im Gegensatz zu Textwerkzeugen – nicht von Anfang an dabei.
- Die Grafik ist enger an die Hardware gebunden als die Ein- und Ausgabe von Zeichen. UNIX versucht aber, hardware-unabhängig zu sein.
- Die mathematischen Grundlagen der Grafikverarbeitung sind zum Teil erst parallel zur Entwicklung der Computer geschaffen worden.

Die Aufgaben der Verarbeitung grafischer Daten lassen sich in zwei Gruppen einteilen:

- die Erzeugung (**Synthese**) und anschließende Weiterverarbeitung von grafischen Objekten (CAD, Finite Elemente, Simulationen, Spiele),
- die Verarbeitung (**Analyse**) von grafischen Objekten, die außerhalb des Computers entstanden sind (Schrifterkennung, Mustererkennung, Fernerkundung, Bildanalyse).

2.9.3 Raster- und Vektorgrafik

Alle Grafikgeräte arbeiten entweder nach dem Raster- oder dem Vektorverfahren. Beim **Vektorverfahren** bestehen die Grafiken aus Linien, die jeweils zwei Punkte verbinden. Diese Linien werden im Computer durch Gleichungen dargestellt. Beim **Rasterverfahren** besteht die Grafik aus einer großen Anzahl von Punkten unterschiedlicher Helligkeit und gegebenenfalls Farbe (Bit- oder Pixmap). Beide Verfahren haben ihre Stärken und Schwächen.

Bildschirme und Laserdrucker sind Rastergeräte, ihre Bilder bestehen aus Punkten (Pixeln). Jede Grafik, die auf diesen Geräten ausgegeben werden soll, muss letzten Endes in einem Rasterformat vorliegen. Das ist auch kein Problem. Will ich ein Rasterbild skalieren – vergrößern oder verkleinern – stellt sich die Frage, wie man aus einem Punkt zwei oder vier macht oder im umgekehrten Fall, wie benachbarte Punkte zusammengefasst werden, ohne Verlust an Bildqualität.

Vektorgrafiken werden durch Formeln beschrieben. Ob ich als Längeneinheit Millimeter oder Meter oder Zoll einsetze, hat auf die Gestalt der Grafik keinen Einfluss, nur auf die Bildgröße. Vektorgrafiken lassen sich ohne Qualitätseinbuße beliebig skalieren, die Qualität (Auflösung, Einzelheiten) hängt nur von dem Aufwand ab, der bei der Aufstellung der Formeln getrieben wurde. CAD verwendet typischerweise Vektorgrafik, während die Verarbeitung von Fotos meist Rastergrafik bedeutet.

Aus einer Vektorgrafik lässt sich einfach eine Rastergrafik erzeugen, man berechnet einfach die entsprechenden Punkte. Das Umgekehrte ist schwierig: Wie soll ein Programm erkennen, dass einige Punkte aus einer Million eine Kreislinie bilden? Der Mensch sieht alle Punkte eines Bildes gleichzeitig, der Computer nacheinander.

2.9.4 Koordinatensysteme

Ein Bild oder Modell wird durch Koordinaten beschrieben, es wird in ein **Koordinatensystem** eingebettet. Sowohl das Bild wie auch jedes Ausgabegerät hat jedoch sein eigenes System. Im Bild wiederum führen einzelne Objekte eigene Koordinatensysteme mit sich. Alle diese Systeme müssen aufeinander abgebildet werden, gegebenenfalls verzerrt. Üblich sind zwei- oder dreidimensionale rechtwinklige Koordinatensysteme mit linearer Teilung (kartesische Koordinaten), deren Ursprung nicht immer in der linken unteren Bildecke oder der Bildmitte liegt. Rechtssysteme überwiegen, dürfen jedoch nicht blindlings vorausgesetzt werden.

Unser Modell wird eingebettet in ein sogenanntes **Welt-Koordinatensystem** (world coordinate system, WCS), oft dreidimensional. Dieses System ist der Ausgangspunkt für alle weiteren Abbildungen und Rechnungen. Es hat nichts mit dem Computer oder irgeneiner Abbildung zu tun.

Wird das Modell auf einem Gerät wie einem Bildschirm oder Drucker dargestellt, so haben diese Geräte ihr eigenes Koordinatensystem, meist zweidimensional. Bei Raster-Ausgabegeräten ist der Bildpunkt die natürliche Einheit. Ein einfacher Bildschirm hat beispielsweise in der Horizontalen (Abszissenachse) 640 Punkte (Pixel), in der Vertikalen (Ordinatenachse) 480[47]. Für einen Laserdrucker lauten die Zahlenwerte anders, auch das Verhältnis beider Werte ist verschieden. Hieraus folgt, dass ein Kreis in Welt-Koordinaten auf dem Bildschirm zu einer Ellipse werden kann, ebenso ein Kreis auf dem Bildschirm zu einer Ellipse auf dem Papier. Wir sprechen

[47]Genauer gesagt ist der Bildschirmspeicher maßgebend, aus dem der Bildschirm einen Ausschnitt zeigt.

hier vom **Geräte-Koordinatensystem**, im einzelnen von **Bildschirm-** oder **Drucker-Koordinaten**. Will man sich zunächst nicht auf ein Gerät festlegen, arbeitet man mit einem virtuellen, nur gedachten Gerät, das einfach auf reale Geräte umgerechnet werden kann.

Heutzutage arbeitet man meist in Fenstern auf einem Bildschirm. Auch auf einem Drucker bedeckt eine Abbildung meist nicht die gesamte bedruckbare oder adressierbare Fläche des Papiers. Wir brauchen also **Fenster-** oder **Bild-Koordinaten**. Bei der Ausgabe müssen diese in Geräte-Koordinaten umgerechnet werden.

Innerhalb des Modells existieren Objekte (Bälle, Fahrzeuge, Teekannen, Planeten), die ihr persönliches **Objekt-Koordinatensystem** mit sich führen. Ein Objekt kann zusammengesetzt sein, ein Fahrzeug zum Beispiel aus einem Kasten und vier Rädern mit jeweils eigenen Koordinatensystemen. Starre Objekte ändern sich in Bezug auf ihr Objekt-Koordinatensystem nicht, aber was passiert mit einem fallenden Wassertropfen oder einem laufenden Menschen?

An irgendeiner Stelle müssen die dreidimensionalen Welt-Koordinaten auf zwei Dimensionen umgerechnet werden. Da kommt die Perspektive ins Spiel. Auch ist die Frage zu klären, wie mit verdeckten Kanten und Flächen umgegangen werden soll.

2.9.5 Grafische Elemente

2.9.5.1 Linien

Gerade Strecken werden durch einfache Gleichungen beschrieben, beliebig geformte Linien werden durch Polygonzüge angenähert. Die Genauigkeit, aber auch der Aufwand an Speicherkapazität und Rechenoperationen steigen mit zunehmender Anzahl der Polygonabschnitte. Naheliegenderweise versucht man, mit einer möglichst groben Näherung auszukommen.

2.9.5.2 Flächen

2.9.5.3 Körper

2.9.6 Transformationen

Unter Tranformationen im weiten Sinn verstehen wir:

- Skalieren (Vergrößern, Verkleinern),
- Translation (geradlinige Bewegung),
- Rotation (Drehung),
- Projektion,
- Umwandlung verschiedener Koordinatensysteme (Kugel-, Zylinder-, kartesische Koordinaten) ineinander.

Transformationen treten auch kombiniert auf.

2.9.7 Modellbildung

2.9.8 Farbe

Seit Farb-Bildschirme, Farb-Scanner und Farb-Drucker erschwinglich geworden sind, wird von Farbe bei Text- und Grafik-Darstellungen zunehmend Gebrauch gemacht. Das World Wide Web ist ohne Farbe nicht denkbar. Damit wachsen die Datenmengen und die Anforderungen an den Benutzer. Während sich der erste Punkt durch die Leistungssteigerungen der Hardware erledigt, muss sich der Benutzer heute auch noch mit Farbmodellen und Drucktechniken auseinandersetzen.

Farbe, was ist das? Farbe ist eine Wahrnehmung, die aus drei Komponenten besteht:

- einer physikalischen Ursache (Licht),
- einem Sinnesorgan (Auge) und
- einer Datenverarbeitung (Auge und Gehirn).

Das Licht geht von einer Quelle aus und trifft direkt oder auf Umwegen auf einen Empfänger. Nachdem der Streit zwischen Wellen- und Korpuskulartheorie des Lichtes beigelegt ist, macht die Physik die geringsten Schwierigkeiten. Auch die physikalischen und chemischen Vorgänge im Auge sind bekannt. Die Verarbeitung der Daten beginnt jedoch schon in der Netzhaut und setzt sich im Gehirn fort. Hier sind fast alle Fragen offen.

Sichtbares Licht ist eine elektromagnetische Welle mit einer Wellenlänge zwischen 380 und 780 nm. In der Natur kommen elektromagnetische Wellen längerer und kürzerer Wellenlänge vor, nämlich Radiowellen beziehungsweise Röntgen- und Gammastrahlung. Wir haben für diese Wellen keine Sinnesorgane. Der Bereich des sichtbaren Lichtes liegt bei Tieren zum Teil geringfügig anders.

Licht kann einfarbig (monochromatisch) sein, es enthält dann nur Wellen einer einheitlichen Wellenlänge. Laserlicht ist ein Beispiel. Einige künstliche Lichtquellen wie Natriumdampflampen erzeugen eine Lichtmischung mit wenigen Wellenlängen. Alltägliche Lichtquellen wie die Sonne, Glühlampen oder Leuchtstoffröhren erzeugen eine Lichtmischung, in der alle Wellenlängen des sichtbaren Bereiches vorkommen und mehr. Sind alle Wellenlängen gleichmäßig vertreten, bezeichnen wir das Licht als weiß[48].

Mischlicht lässt sich durch Prismen oder Gitter in seine Bestandteile aufspalten, da die Brechung oder Beugung von der Wellenlänge abhängt. Man erhält so ein Spektrum, das bei der Aufspaltung weißen Lichtes kontinuierlich die Farben des Regenbogens zeigt. Andere Lichtarten ergeben ein Spektrum,

---

[48]Weißes Licht scheint es nicht zu geben, sondern nur die Normlichtart D65, die einem Tagelicht mit einer Farbtemperatur von 6500 K nahekommt. Einzelheiten in den DIN-Normen ab DIN 5030, in Physikbüchern – beispielsweise FRIEDRICH KOHLRAUSCH, Praktische Physik – unter dem Stichwort *Optik/ Colorimetrie* oder bei der Physikalisch-Technischen Bundesanstalt `www.ptb.de`.

das nur aus einigen Linien oder aus einem Kontinuum besteht, dem einige Linien überlagert sind. Greift man sich aus dem Spektrum zwei Farben heraus und mischt sie wieder zusammen, erhält man eine Mischfarbe.

2.9.9 Oberfläche und Beleuchtung (Rendering)

Die Gegenstände unserer Umgebung – so weit sie nicht selbst leuchten – sehen wir, weil ihre Oberfläche das Licht einer Lichtquelle zurückwirft. Rückseitige Flächen sind unsichtbar, andere Flächen werfen Schatten. Der Sinneseindruck entsteht aus dem Zusammenwirken von Lichtquelle, Oberflächen und Augen. Entsprechend vielfältig sind die Kombinationen (und entsprechend aufwendig die Programme, die aus einem wenig anschaulichen Linienmodell ein annähernd realistisches Bild errechnen).

2.9.10 Nachbearbeitung

Die großen Datenmengen haben zu verschiedenen Versuchen geführt, sie teils ohne, teils mit Verlust zu komprimieren. Die vier bekanntesten Grafikformate sind:

- Compuserve Graphics Interchange Format (.gif),
- Joint Photographic Experts Group Format (.jpg, .jpeg), mit wählbarem Kompressionsfaktor,
- Tag Image File Format (.tiff), verbreitet bei Scannern,
- Portable Network Graphics (.png) des W3-Konsortiums.

Von vielen Grafikformaten sind im Lauf der Jahre verschiedene Versionen oder Releases herausgekommen. Es gibt sowohl freie wie kommerzielle Werkzeuge zur Umwandlung einiger Grafikformate ineinander. Insbesondere zum png-Format findet sich im WWW eine gute Dokumentation, naheliegenderweise, zusammengestellt von THOMAS BOUTELL.

2.9.11 Grafik-Bibliotheken (GKS, OpenGL, Mesa)

Zur Verarbeitung von Zahlen braucht man Zahlenfunktionen wie Addition, Logarithmus, Regula falsi. Grafikfunktionen sind Verschieben (Translation), Drehen (Rotation), Spiegeln, Verzerren. Ein weit verbreitetes und vom American National Standards Institute (ANSI) genormtes Paket mit Grafikfunktionen ist das **Graphical Kernel System** (GKS), mittlerweile veraltet und durch neuere Grafikpakete ersetzt. Das unter ANSI X3.124-1985, DIN 66 292 und ISO 7942 beschriebene System enthält Grundfunktionen zur Bewältigung grafischer Aufgaben auf dem Computer. Die Norm legt die Funktionalität und die Syntax fest, Softwarehersteller bieten GKS-Pakete als kompilierte C- oder FORTRAN-Funktionen für eine Reihe von Prozessoren an. Die Sammlung enthält Funktionen:

- zur Ausgabe grafischer Grundelemente,

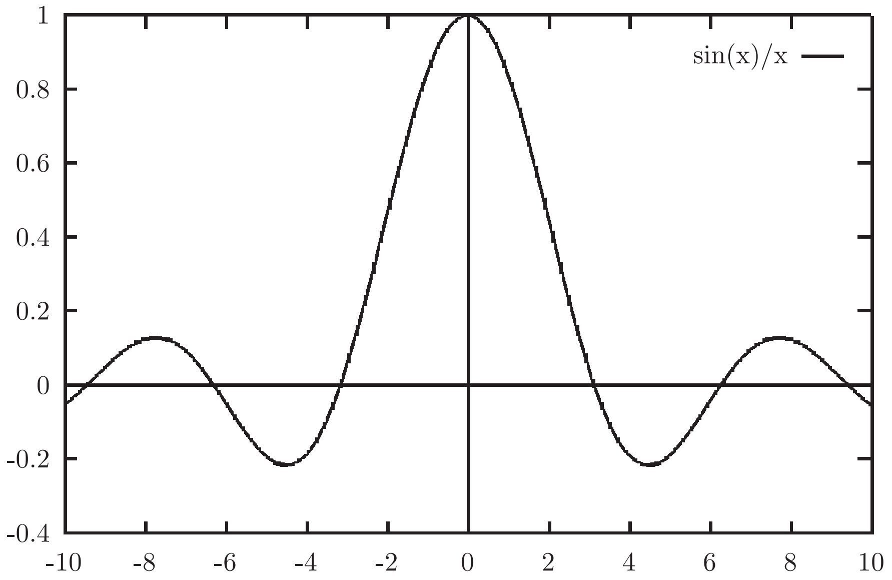

Abb. 2.10: Diagramm von (sin x)/x, erzeugt mit gnuplot

- für die Attribute der Grundelemente,
- zur Steuerung der Workstation,
- für Transformationen und Koordinatensysteme,
- zur Bearbeitung von Elementgruppen (Segmenten),
- zur Eingabe,
- zur Bearbeitung von Metafiles,
- für Statusabfragen,
- zur Fehlerbehandlung.

Inzwischen hat allerdings die modernere, von Silicon Graphics (SGI) entwickelte Grafik-Software **OpenGL** samt dem Toolkit **GLUT** und dem unter GNU Copyleft veröffentlichten Klone **Mesa** dem GKS-System den Rang abgelaufen. OpenGL ist systemunabhängig, netzfähig (Client-Server-Modell) und beherrscht drei Dimensionen. Es gibt eine eigene Bibliothek für die Zusammenarbeit mit X11, ferner eine Bibliothek namens GLIDE zur optimalen Ausnutzung bestimmter Grafikhardware. Mit OpenGL sind Computerspiele und wissenschaftliche Simulationen geschrieben worden. Da die Entwicklung rasch voranschreitet, sind aktuelle Informationen am einfachsten im Netz zu finden, insbesondere die Spezifikation. OpenGL-Bibliotheken zum Einbinden in C-Programme sind gegen ein mäßiges Entgelt auch für Linux zu erhalten.

2.9.12 Datenrepräsentation

`gnuplot(1)` ist ein Programm zum Zeichnen von Diagrammen, das GNU-üblich als Quellcode vorliegt, aber nicht aus dem GNU-Projekt stammt. Sofern man es nicht nachgeworfen bekommt, holt man es sich bei `www.gnuplot.org` oder einem Mirror in Form eines gzippten tar-Archivs.

Nach dem Entzippen, Enttaren und Lesen der Datei `0INSTALL` editiert man die Datei `term.h`, um eine sinnvolle Auswahl von Ausgabegeräten einzubinden. In unserem Fall unter HP-UX 10.20 waren das vor allem HP- und LaTeX-Treiber. Dann ruft man:

```
CC=/usr/bin/cc ./configure
```

auf, mit der Anweisung, statt des `gcc` den HP-C-Compiler zu verwenden, wird empfohlen. Im Makefile fügt man dann zu den `DEFS` noch `-DSHORT_TERMLIST` hinzu – siehe `term.h` – und ruft `make` auf. Nachdem die Compilierung erfolgreich abgeschlossen ist, befiehlt der Superuser `make install`. Das ist alles. Wer dazu neigt, Anleitungen zu lesen, holt sich vom Server die rund hundertseitige Dokumentation zu `gnuplot`.

Ausgangspunkt eines Plots ist entweder eine Funktionsgleichung oder eine Wertetabelle. Sowohl cartesische wie Polarkoordinaten können verwendet werden. Dreidimensionale Darstellungen in cartesischen, Kugel- oder Zylinderkoordinaten sind ebenfalls möglich. Die Achsen können linear oder logarithmisch geteilt sein. Andere Teilungen muss man selbst programmieren. Soweit sinnvoll, werden reelle und komplexe Argumente verarbeitet. Das Programm wird entweder interaktiv (Terminal-Dialog) oder durch ein Skript gesteuert.

Die Ausgabe geht in eine Datei oder auf ein Gerät. Treiber für einige Terminals und den HP Laserjet gehören dazu, ebenso die Möglichkeit, PostScript-, LaTeX- oder HPGL-Dateien zu erzeugen. Hier ein einfaches Beispiel. Wir schreiben ein Skript `plotscript`:

```
set term latex              # Ausgabe im LaTeX-Format
set output "plot.tex"       # Ausgabe nach File plot.tex
plot sin(x)/x               # zu zeichnende Funktion
```

Quelle 2.43 : gnuplot-Skript zum Zeichnen der Funktion y = (sin x)/x, Ausgabe im LaTeX-Format in Datei plot.tex

und rufen `gnuplot` mit dem Skript als Argument auf:

```
gnuplot plotscript
```

Interaktiv wären die Kommandos:

```
gnuplot
set term latex
set output "plot.tex"
plot sin(x)/x
quit
```

einzugeben. Für alle nicht genannten Parameter werden Default-Werte genommen. Als Ausgabe erhalten wir eine LaTeX-Picture-Umgebung, die sich in ein LaTeX-Dokument einbinden lässt, siehe Abbildung 2.10 auf Seite 229.

`gnuplot(1)` beherrscht auch die zweidimensionale Darstellung von Versuchsergebnissen mit drei Variablen, siehe Abbildung 2.11 auf Seite 231. Hier

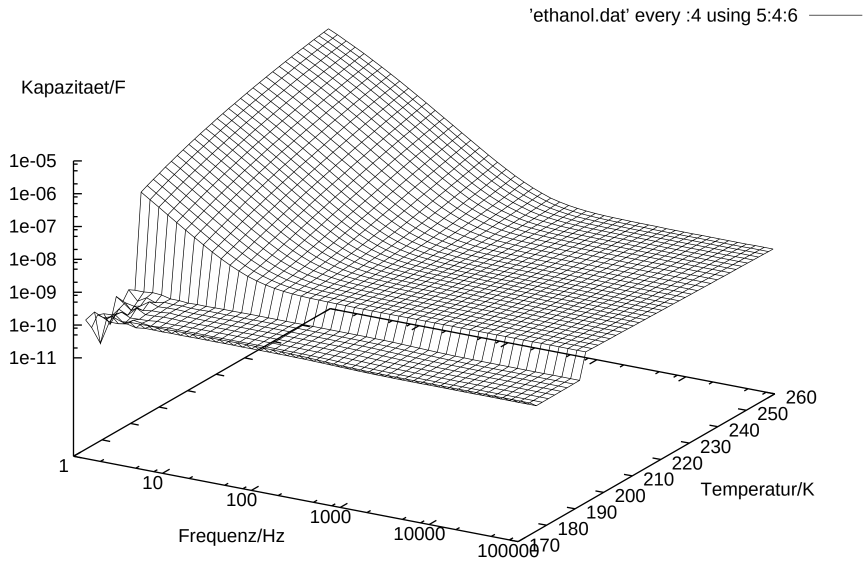

Abb. 2.11: Versuchsdaten, dargestellt mittels `gnuplot`

wurde der Versuch von einem PC gesteuert, die Ergebnisse wurden sofort gespeichert und nachträglich ausgewertet.

Für Konstruktions-Zeichnungen oder Illustrationen ist `gnuplot(1)` nicht gedacht. Unter

`http://www.uni-karlsruhe.de/~ig25/gnuplot-faq/`

findet sich ein FAQ-Text zu `gnuplot(1)`.

2.9.13 Zeichnungen

`xfig(1)` und `xpaint(1)` sind Werkzeuge, die unter dem X Window System laufen. Sie machen von dessen Funktionen Gebrauch und sind infolgedessen netzfähig. `xfig(1)` dient zum Erstellen von Zeichnungen (Vektorgrafiken) mit Linien und Text.

Es verwendet eine Drei-Tasten-Maus, ist menugesteuert und kennt verschiedene Ausgabeformate, darunter LaTeX (Picture-Umgebung), (Encapsulated) PostScript, HP-GL, GIF, JPEG und PNG. Einzelheiten am schnellsten auf der man-Page, rund 60 Seiten Papier. Die Abbildung 2.12 auf Seite 232 wurde mit dem Aufruf:

```
xfig -me -p -e latex -lat
```

erstellt. Die Zeichnungen lassen sich natürlich nicht nur erzeugen, sondern auch nachträglich ändern (editieren).

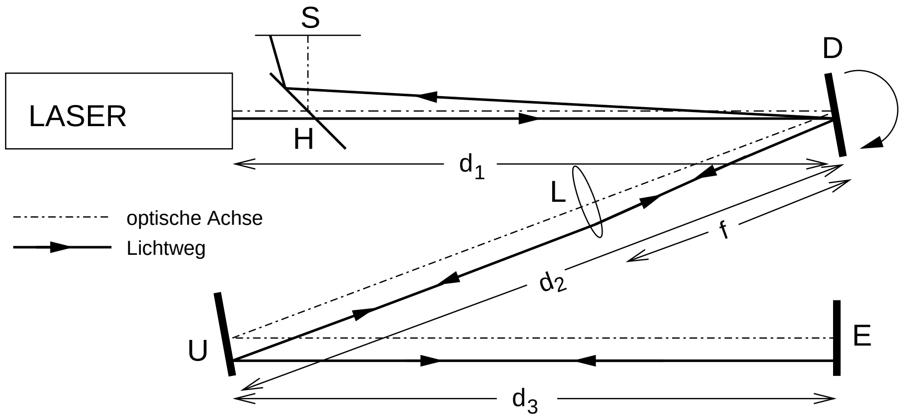

Abb. 2.12: Mittels xfig erstellte Zeichnung

2.9.14 Fotos

(GIMP)

Das Werkzeug `xpaint(1)` erlaubt das Erzeugen und Editieren von Farbbildern. Die man-Page ist knapp, dafür steht eine ausführliche online-Hilfe zur Verfügung. Editiert man bereits vorhandene Bilder, sollte man nur mit einer Kopie arbeiten, da `xpaint(1)` die Farbinformationen (Farbtiefe) dem jeweiligen Display anpasst, man also unter Umständen Informationen verliert. Auch `xpaint(1)` kennt alle gängigen Grafikformate.

Aus dem GNU-Projekt stammt `gimp(1)` (The Gimp), ein Werkzeug zur Erzeugung und Bearbeitung von Bildern einschließlich der Retusche von Fotos. Der Name bedeutet *GNU Image Manipulation Program*. Es setzt ebenfalls auf X11 auf und kennt zahlreiche Formate. Auch zu deren Umwandlung kann es verwendet werden. Näheres im WWW unter `http://www.gimp.org/`. Im GNU-Projekt finden sich weitere Werkzeuge für grafische Arbeiten in den Verzeichnissen `.../gnu/graphics/` und `.../gnu/plotutils/` der entsprechenden FTP-Server.

Ein kommerzielles Programm zur Bearbeitung digitalisierter Fotos ist Adobe Photoshop, in vereinfachter Form für den Heimgebrauch als PhotoDeluxe auf dem Markt. Adobe ist zugleich in der Textverarbeitung stark (PostScript, pdf-Format) und bietet ein ganzes Bündel von Programmen zur Gestaltung illustrierter Texte an, die zusammenarbeiten.

2.9.15 Animation

2.9.16 Computer Aided Design

Computer Aided Desigen (CAD) heißt Konstruieren am Bildschirm anstelle des Reißbrettes. Wenn es nur um eine einzelne Zeichnung geht, bietet CAD keine Vorteile gegenüber Papier und Bleistift. Braucht man jedoch Varianten eines Bauteiles, Stücklisten, Modelle, automatische Verbindungen zur Lagerhaltung und zur Fertigung auf CNC-Maschinen, dann ist der Computer in seinem Element.

Weit verbreitet ist das CAD-Programm *AutoCAD*. Leider kostet es richtig Geld, auch als Studentenversion. Freie Alternativen – unter der GPL bespielsweise – mit ähnlichem Leistungsumfang sind mir nicht bekannt. Unter:

`www.computer.privatweb.at/tech-edv/CADlinks.htm`

findet sich eine Zusammenstellung von CAD-Werkzeugen für Linux, teils kommerziell, teils frei. Manche wie `gnucad`, `gtkcad` oder `powercad` sind zur Zeit (Anfang 2003) *in stage of planning*. Andere wie `arcad` (für Architekten) werden bereits beruflich eingesetzt.

2.9.17 Präsentationen

Unter einer **Präsentation** versteht man einen Vortrag mit computerunterstützter, multimedialer Begleitung. Was früher aus Episkopen, Dia- und Kinoprojektoren kam, kommt heute aus dem Beamer. Damit lassen sich weit mehr Effekte erzielen als mit der alten (auch nicht billigen) Technik. Außerdem lässt sich eine Präsentation über das Netz verbreiten. Bis zu einem gewissen Umfang kann man sogar den Vortragenden einsparen. Wenn man dann noch ein Eliza-Programm zur Beantwortung etwaiger Fragen einsetzt ... vielleicht doch besser ein Expertensystem. Oder einen Experten mit System.

Gegenwärtig sind zwei Systeme verbreitet: Microsoft Powerpoint und ein offenes System aus Webseiten, das nur einen Brauser voraussetzt. Eine frei Alternative zu Powerpoint ist in OpenOffice enthalten und auf Linux-Rechnern verbreitet.

2.9.18 Begriffe Grafik

Die folgenden Begriffe sollten klarer geworden sein:

- Animation
- Computer Aided Design (CAD)
- Datenrepräsentation (`gnuplot`)
- Farbmodell
- Grafikformate (eps, jpeg, tiff, png)
- grafische Daten (im Gegensatz zu Zeichen)
- Koordinatensysteme (Welt, Gerät, Objekt)
- Präsentation (Vortrag mit multimedialer Unterstützung)
- Rastergrafik – Vektorgrafik
- Rendering

Folgende Kommandos (Werkzeuge) sollten beherrscht werden:

- Anfänge von `gnuplot`
- Anfänge von `xfig`

2.9.19 Memo Grafik

- Für die Verarbeitung grafischer Daten gibt es keinen Standard (oder zuviele, was auf dasselbe hinausläuft). Das ganze Gebiet ist noch im Fluss.
- Unter UNIX gibt es keine Standard-Werkzeuge oder -Bibliotheken, wohl aber mehrere, zum Teil freie Grafik-Pakete.
- `gnuplot(1)` ist ein interaktives Werkzeug zur Erzeugung von Diagrammen, ausgehend von Wertetabellen oder Funktionsgleichungen.
- Auch das X Window System (X11) enthält Grafikfunktionen, der Schwerpunkt liegt jedoch in der Gestaltung von Fenstern, die über das Netz gehen.
- `xfig(1)`, `xpaint(1)` und `gimp(1)` sind interaktive Werkzeuge zum Herstellen und Bearbeiten von Zeichnungen und Bildern. Die Werkzeuge bauen auf dem X Window System auf und sind in Linux-Distributionen enthalten.
- Das Graphical Kernel System (GKS) ist eine international genormte Bibliothek mit Grafikfunktionen.
- OpenGL ist eine modernere Bibliothek mit Grafikfunktionen (netzfähig, dreidimensional) und verbreitet sich rasch.

2.9.20 Übung Grafik

- Erkundigen Sie sich bei Ihrem System-Manager nach den verfügbaren Grafik-Werkzeugen.
- Zeichnen Sie ein cartesisches Koordinatensystem und darin einen Kreis mit dem Koordinatenursprung als Mittelpunkt. Was mathematisch ein Kreis ist, braucht auf dem Bildschirm wegen unterschiedlicher Maßstäbe in x- und y-Richtung noch lange nicht wie ein Kreis auszusehen.
- Verzerren Sie die Maßstäbe so, dass der Kreis auch wie ein Kreis aussieht.
- Fügen Sie eine Beschriftung in obiges Diagramm ein.
- Drucken Sie das Diagramm aus. Ist der Kreis immer noch kreisförmig?

2.9.21 Fragen zur Grafik

- Was heißt Rastergrafik? Vektorgrafik? Vergleich?
- Was macht `gnuplot`? `xfig`?

2.10 Das digitale Tonstudio

2.10.1 Grundbegriffe

Um mit einem Rechner akustische Daten zu verarbeiten, braucht man:

- Hardware:
  - Signalquellen (Mikrofon, Radio, Tonbandgerät, Plattenspieler, CD/DVD-Spieler, Keyboard usw.),
  - Kabel, Adapter,
  - Rechner mit Audiochip auf dem Mainboard oder getrennter Soundkarte,
  - Ausgabegeräte (Verstärker, Lautsprecher, Aktivboxen, CD/DVD-Brenner usw.),
- Software:
  - Treiber für den Audiochip oder die Soundkarte, das übliche Problem,
  - Anwendungsprogramme (Mischer, Ripper, Recorder, Formatwandler, Audio-Editoren usw.).

Schall (E: sound, F: son) besteht aus Druckschwankungen der Luft im Frequenzbereich von 20 Hz bis 20 kHz. Am Anfang der ganzen Maschinerie steht immer ein Gerät, das die Druckschwankungen in Schwankungen elektrischer Spannung umwandelt, typischerweise ein Mikrofon, welches ein analoges, elektrisches Signal liefert. Das Signal kann analog verstärkt, gespeichert und in andere physikalische Größen wie Licht, Radiowellen oder auch wieder Schall umgewandelt werden. Die Technik der analogen Schallverarbeitung ist hoch entwickelt, wird aber langsam von der digitalen Technik überholt.

Solange die gebräuchlichen Schallquellen ein analoges Signal liefern, steckt im Eingang jeder Soundkarte ein Analog-Digital-Wandler. Ähnliches gilt für den Ausgang jeder Soundkarte: solange Lautsprecher ein analoges Signal brauchen, steckt im Ausgang jeder Soundkarte ein Digital-Analog-Wandler. Zwischen den beiden Wandlern vollzieht sich die digitale Signalverarbeitung. Soundkarten haben mehrere Anschlüsse:

- extern:
  - Mikrofon-Eingang,
  - Line-Eingang (von einem Verstärker, Radio oder dergleichen),
  - Line-Ausgang, evntl. für rechts und links getrennt (zu einem Verstärker oder zu Aktivboxen),
  - Gameport (15polig, für Freudenknüppel),

  - intern (auf der Karte):

   - CD-Eingang (vom analogen Audio-Ausgang eines eingebauten CD-Laufwerks),
   - Sonstige (Modem, Anrufbeantworter, SPDIF . . .).

Kabel und Adapter werden besonders erwähnt, weil sie eine ständige Quelle der Freude sind. Die Anzahl der Steckertypen in der Audiotechnik ist Legion, und die Möglichkeiten, die Adern eines mehradrigen Kabels auf die Stifte eines mehrpoligen Steckers zu verteilen, sind vielfältig. Dazu kommt noch, dass jeder Ein- oder Ausgang einen bestimmten elektrischen Widerstand hat und für eine bestimmte elektrische Maximalspannung ausgelegt ist. Man kann nicht ein niederohmiges Mikrofon an einen hochohmigen Verstärkereingang anschließen, selbst falls der Stecker passt. Von Brummschleifen wollen wir gar nicht erst reden. Die Lehre daraus ist, dass man für den Aufbau und Test einer umfangreicheren Audioanlage viel Zeit einplanen sollte.

Die Treiber kommen entweder vom Hersteller der Karte oder des Mainboards (meist für MS-Windows, manchmal auch für Linux) oder vom Hersteller oder Distributor des Betriebssystems wie Debian.

((Fortsetzung folgt))

2.10.2 Datei-Formate

((noch in Arbeit))

2.10.3 Begriffe Tonstudio

Folgende Begriffe sollten klarer geworden sein:

- Sound-Formate (mp3, wav, ra)

Folgende Kommandos sollten beherrscht werden:

- Anfänge eines Audio-Editors (??)

2.10.4 Memo Tonstudio

2.10.5 Übung Tonstudio

2.10.6 Fragen Tonstudio

2.11 Kommunikation

2.11.1 Message (write, talk)

Unter **Kommunikation** verstehen wir den Nachrichtenaustausch unter Benutzern, zunächst beschränkt auf die Benutzer einer Anlage. Zur Kommunikation im Netz kommen wir im Skriptum *Internet*. Zwei gleichzeitig angemeldete Benutzer (mit `who(1)` abfragen) können über ihre Terminals einen **Dialog** miteinander führen. Das Kommando lautet `write(1)`:

```
write username ttynumber
```

also beispielsweise

```
write gebern1 tty1p1
```

Die Angabe des Terminals darf entfallen, wenn der Benutzer nur auf einem Terminal angemeldet ist. Der eingegebene Text wird mit der RETURN-Taste abgeschickt, der Dialog wie üblich mit `control-d` beendet. Da der Bildschirm eigene und fremde Zeichen wiedergibt, wie sie kommen, ist Disziplin angebracht, genau wie beim Wechselsprechen über Funk. Eine Konferenz mit mehreren Teilnehmern ist technisch möglich, praktisch aber kaum durchzuführen.

Das nicht überall vorhandene Kommando `talk(1)` teilt den Bildschirm unter den beiden Gesprächspartnern auf, sodass auch bei gleichzeitigem Senden die Übersicht gewahrt bleibt. Jeder Buchstabe wird sofort gesendet.

Ein Benutzer verhindert mit dem Kommando `mesg(1)` mit dem Argument `n`, dass er während seiner Sitzung durch Messages gestört wird. Er entzieht der Allgemeinheit die Schreiberlaubnis für sein Terminal `/dev/tty....` Das entspricht allerdings nicht dem Geist von UNIX. Die Standardeinstellung unserer Anlage ist `mesg y` (in `/etc/profile` gesetzt).

2.11.2 Mail (mail, mailx, elm, mutt)

Ein elektronisches Mailsystem ermöglicht, einem Benutzer, der momentan nicht angemeldet zu sein braucht, eine Nachricht zu schreiben. Bei nächster Gelegenheit findet er den elektronischen Brief; ob er ihn liest, ist seine Sache. Eine Rückmeldung kommt nicht zum Absender. Man kann auch Rundschreiben an Benutzergruppen oder an alle versenden. Das System selbst macht ebenfalls von dieser Möglichkeit Gebrauch, wenn es einen Benutzer nicht im Dialog erreichen kann. Eine nützliche Sache, sowohl als Hauspost wie als weltweite **Electronic Mail**, nur die Briefmarkensammler trauern. Die herkömmliche, auf dem Transport von Papier beruhende Post wird demgegenüber als Snail-Mail oder kurz **Snail** bezeichnet, was im Englischen Schnecke heißt. Mailsysteme befördern grundsätzlich nur Texte, oft in 7-bit-ASCII, keine Grafiken oder andere binäre Dateien (komprimierte Dateien, kompilierte Programme). Hat man binäre Dateien per Mail zu übertragen, muss man sie erst in Textdateien umwandeln (siehe `uuencode(1)` oder Metamail). Andere Wege wie `ftp(1)` sind für binäre Daten geeigneter.

Mit dem Kommando `mail(1)` wird der Inhalt der eigenen Mailbox (die Datei `/var/mail/username` oder `/var/spool/mail/username`) angezeigt, Brief für Brief, der jüngste zuerst. `mail(1)` fragt bei jedem Brief mit dem Prompt `?`, was es damit machen soll: im Briefkasten lassen, in einer Datei `mbox` ablegen oder löschen. Mit der Antwort `*` auf den Mail-Prompt erhalten Sie eine Auskunft über die Kommandos von `mail(1)`.

`mail(1)` mit einem Benutzernamen als Argument aufgerufen öffnet einen einfachen Editor zum Schreiben eines Briefes. Mit `return control-d` wird der Brief beendet und abgeschickt. Man kann auch ein Textfile per Redirektion als Briefinhalt einlesen:

```
mail wualex1 < textfile
```

oder `mail(1)` in einer Pipe verwenden:

```
who | mail wualex1
```

`mail(1)` kommt mit den einfachsten Terminals zurecht und ist daher die Rettung, wenn bessere Mail-Programme wegen fehlender oder falscher Terminalbeschreibung versagen.

Die Umgebungsvariable MAILCHECK bestimmt, in welchen Zeitabständen während einer Sitzung die Mailbox auf neue Mail überprüft werden soll. Üblich sind 600 s. Durch das Kommando `mail(1)` in `/etc/profile` wird automatisch beim Anmelden die Mailbox angezeigt. Ein `dead.letter` ist ein unzustellbarer Brief, aus welchen Gründen auch immer. Enthält eine Mailbox als erste Zeile:

```
Forward to person
```

(mit großem F) so wird alle Mail für den Inhaber der Mailbox an den Benutzer `person` auf dieser Maschine weitergeleitet. Damit kann man Mail an logische Benutzer wie `root` bestimmten natürlichen Benutzern zuweisen, je nach Abwesenheit (Urlaub, Krankheit) an verschiedene. Lautet die Zeile:

```
Forward to person@abc.xyz.de
```

geht die Mail an einen Benutzer auf der Maschine `abc.xyz.de`. Das ist praktisch, falls ein Benutzer Mailboxen auf mehreren Systemen hat, die Mail aber nur auf seinem wichtigsten System liest. Die Mailboxen müssen als Gruppe `mail` sowie Lese- und Schreiberlaubnis für die Gruppe (660) haben.

Das UNIX-Kommando `mailx(1)` bietet erweiterte Möglichkeiten, insbesondere die Konfiguration mittels einer Datei `$HOME/.mailrc`. Auf vielen Systemen ist auch das bildschirmorientierte und benutzerfreundlichere Mailkommando `elm(1)` vorhanden. Es setzt die richtige Terminalbeschreibung (Umgebungsvariable TERM, `terminfo` oder `termcap`) voraus, fragt nach den notwendigen Informationen, ruft zum Schreiben den gewohnten Editor auf und lässt sich durch eine Datei `$HOME/.elm/elmrc` an persönliche Wünsche anpassen. In `$HOME/.elm/elmheaders` werden zusätzliche Kopfzeilen – z. B. die Organisation – festgelegt, in `$HOME/.signature` eine Signatur am Ende der Mail. `elm(1)` ist mit `mail(1)` verträglich, man kann sie durcheinander benutzen. Ähnlich vie `elm(1)` verhält sich `mutt(1)`; der Umstieg fällt leicht. `mutt(1)` kann mehr und wird weiter entwickelt, während es um `elm(1)` ruhig geworden ist. Zu einem Zeitpunkt darf immer nur ein Mailprogramm aktiv sein, sonst gerät die Mailverwaltung durcheinander. Wird durch Lock-Dateien geregelt.

Es empfiehlt sich, ein Email-Alias namens `postmaster` für `root` oder sonst einen vertrauenswürdigen Benutzer in `/etc/mail/aliases` einzurichten. Ein Benutzer namens `postmaster` ist nicht erforderlich, außerdem ist sein Name zu lang für die übliche `passwd`-Datei. Der **Postmaster** erhält als Default die Problemfälle des Mail-Systems zugeschickt; außerdem kann man ihn als Anschrift für alle Benutzer gebrauchen, die nicht wissen, was eine Mailbox ist. Während die Mail innerhalb *einer* Anlage einfach ist, erfordert eine weltweite Mail einen größeren Aufwand, ist aber auch viel spannender, siehe das Skriptum *Internet*.

2.11.3 Neuigkeiten (news)

Neuigkeiten oder **News** sind Mitteilungen, die jedermann schreiben und lesen darf. Die Dateien sind in `/var/spool/news` zu finden. Falls Sie eine Runde locker machen wollen, tippen Sie

```
vi /var/spool/news/freibier
a
Heute gibt es Freibier.
escape
:wq
chmod 644 /var/spool/news/freibier
```

Vergessen Sie nicht, Ihren News die Leseerlaubnis für alle (644) mitzugeben und das Bier bereitzustellen. News innerhalb einer Maschine sind wie die Mail eine harmlose Angelegenheit, im Netz wird es aufwendiger.

Die Datei `.news_time` im Home-Verzeichnis hält die Zeit der letzten News-Anzeige fest, sodass man im Regelfall nur neue Mitteilungen zu lesen bekommt. Das Kommando `news(1)` in der Datei `/etc/profile` sorgt dafür, dass bei jeder Anmeldung die Neuigkeiten angezeigt werden. Sie können es aber auch gesondert eingeben. Mittels `news -a` werden alle, neue wie alte, Nachrichten angezeigt. Diese News des UNIX-Systems haben nichts mit den Netnews im Internet zu tun, die im Skriptum *Internet* erläutert werden.

2.11.4 Message of the Day

Mittels der **Message of the Day** – das Wort zum Alltag – schickt der System-Manager eine Mitteilung an alle Benutzer, die sie jedesmal beim Einloggen zu lesen bekommen. Der Text steht in `/etc/motd`, Anzeige mittels `cat /etc/motd` in `/etc/profile`. Hinweise auf neue Programmversionen, drohende Reparaturen oder ein neues, fabelhaftes Buch über UNIX, C und das Internet gehören hierhin.

2.11.5 Ehrwürdig: UUCP

UUCP heißt *unix-to-unix-copy* und ist ein Programmpaket zur Übertragung von Dateien zwischen UNIX-Anlagen über serielle Kabel oder Modemstrecken, eine frühe Alternative zu den Internet-Diensten nach TCP/IP-Protokollen. Mail und Netnews werden außerhalb des Internets noch viel über UUCP ausgetauscht. Im Gegensatz zu den Aufträgen an Internet-Dienste werden UUCP-Aufträge zwischengespeichert (gespoolt), erklärlich aus der Verwendung von Modemverbindungen über Telefon-Wählleitungen.

Zu dem Paket gehört ein Terminal-Emulator `cu(1)` (= call UNIX), der ein einfaches serielles Terminal emuliert (aus einem Computer ein Terminal macht). Das Programm kann benutzt werden, um einen Computer über ein serielles Kabel – gegebenenfalls verlängert durch Modem und Telefonleitung – an einen anderen Computer anzuschließen, falls man keine Netzverbindung mittels `rlogin(1)` oder `telnet(1)` hat.

Die UUCP-Programme bilden eine Hierarchie, auf deren unterster Ebene die Programme `uucico(1)` und `uuxqt(1)` die Verbindung zwischen zwei Maschinen herstellen. In der Mitte finden sich Programme wie `uucp(1)`, `uux(1)` und `uuto(1)`, die zwar Aufgaben im Auftrag eines Benutzers erledigen, normalerweise aber nicht unmittelbar von diesem aufgerufen werden, sondern in periodischen Abständen durch einen Dämon. Zuoberst liegen vom Benutzer aufgerufenen Programme wie `mail(1)` und `news(1)`. Dazu kommen Hilfsprogramme wie `uuencode(1)` oder `uustat(1)`. `uuencode(1)` wird gelegentlich auch außerhalb der UUCP-Welt benutzt, um binäre Dateien in Textdateien zum Versand per Email umzucodieren:

```
uuencode myfile | mailx -s 'Subject' wualex1@mvmhp64
```

und zurück:

```
uudecode < mymail
```

wobei die Mail-Header-Zeilen nicht stören. Da die UUCP-Programme innerhalb des Internets keine Rolle spielen, verweisen wir für Einzelheiten auf das Buch von B. ANDERSON und den Text von I. L. TAYLOR.

2.11.6 Begriffe Kommunikation

Folgende Begriffe sollten klarer geworden sein:

- Message of the Day
- News
- UUCP

Folgende Kommandos sollten beherrscht werden:

- `write` oder `talk`

2.11.7 Memo Kommunikation

- Zwischen den Benutzern derselben UNIX-Maschine bestehen seit altersher Möglichkeiten der Kommunikation. Die Kommunikation im Netz (auf verschiedenen Maschinen) erfordert zusätzliche Protokolle und Programme.
- Zwei gleichzeitig angemeldete Benutzer können mittels `write(1)` oder `talk(1)` einen Dialog per Tastatur und Bildschirm führen.
- Email ist ein zeitversetzter Nachrichtenaustausch zwischen zwei Benutzern (oder Dämonen), wie eine Postkarte.

- News sind Aushänge am Schwarzen Brett, die alle lesen können.
- Die Message of the Day ist eine Mitteilung des System-Managers, die alle lesen müssen.
- UUCP ist ein Bündel mehrerer Programme, das dem Datenaustausch zwischen UNIX-Maschinen über Wählleitungen (Modemstrecken) dient und im wesentlichen durch das Internet abgelöst worden ist.

2.11.8 Übung Kommunikation

Zur Kommunikation brauchen Sie einen Gesprächspartner, nur Mail können Sie auch an sich selbst schicken. Im Notfall steht Ihr Freund, der System-Manager (`root`), oder der Postmaster zur Verfügung.

`set`	(Umgebung ansehen)
`who`	(Partner bereit?)
`write partner`	(Bell abwarten)
Dialog führen	(nur mit RETURN, kein `control-d`)
`oo`	(over and out, Ende des Gesprächs)
`control-d`	(Ende des Gesprächs)
`mail`	(Ihr Briefkasten)
`*`	(`mail`-Kommandos ansehen)
`mail username`	(Brief an `username`)
Brief schreiben, RETURN	
`control-d`	(Ende des Briefes)
`elm`	(`elm` gibt Hinweise)
`cat > /usr/news/heute`	(News schreiben)
`Heute gibts Freibier.`	
`control-d`	(Ende News)
`chmod 644 /usr/news/heute`	
abmelden, wieder anmelden	
`news -a`	(alle News anzeigen)
`rm /usr/news/heute`	
`cat /etc/motd`	(MOTD anzeigen)

Falls es keine Message of the Day gibt, Mail an `root` schicken.

Abmelden mit `exit`.

2.11.9 Fragen Kommunikation

2.12 Echtzeit-Erweiterungen

Unter UNIX wird die Reihenfolge, in der Prozesse abgearbeitet werden, vom System selbst beeinflußt, ebenso der Verkehr mit dem Massenspeicher (Pufferung). Für einen Computer, auf dem nur gerechnet, geschrieben und gezeichnet wird, ist das eine vernünftige Lösung. Bei einem **Prozessrechner** hingegen, der Meßwerte erfaßt und eine Produktionsanlage, eine Telefonvermittlung oder ein Verkehrsleitsystem steuert, müssen bestimmte Funktionen in garantierten, kurzen Zeitspannen erledigt werden, hier darf das System keine Freiheiten haben. Ein solches System mit einem genau bestimmten Zeitverhalten nennt man **Echtzeit-System** (real time system). Um mit einem UNIX-System Echtzeit-Aufgaben zu bewältigen, hat die Firma Hewlett-Packard ihr HP-UX um folgende Fähigkeiten erweitert:

- Echtzeit-Vorrechte für bestimmte Benutzergruppen,
- Verfeinerung der Prioritäten von Prozessen,
- Blockieren des Arbeitsspeichers durch einen Prozess,
- höhere Zeitauflösung der System-Uhr,
- verbesserte Interprozess-Kommunikation,
- schnelleres und zuverlässigeres Datei-System,
- schnellere Ein- und Ausgabe,
- Vorbelegung von Platz auf dem Massenspeicher,
- Unterbrechung von Kernprozessen.

Der Preis für diese Erweiterungen ist ein erhöhter Aufwand beim Programmieren und die gelegentlich nicht so effektive Ausnutzung der Betriebsmittel. Wenn Millisekunden eine Rolle spielen, kommt es auf einige Kilobyte nicht an. Die nicht privilegierten Benutzer müssen auch schon einmal ein bißchen warten, wenn eine brandeilige Meldung bearbeitet wird.

Bestimmte Benutzergruppen erhalten das Vorrecht, ihre Programme oder Prozesse mit Echtzeitrechten laufen zu lassen. Sie dürfen wie der Super-User höhere Prioritäten ausnutzen, bleiben aber wie andere Benutzer an die Zugriffsrechte der Dateien gebunden. Würde dieses Vorrecht allen gewährt, wäre nichts gewonnen. Der System-Manager vergibt mit `setprivgrp(1M)` die Vorrechte, mit `getprivgrp(1)` kann jeder die seinigen abfragen.

Ein Benutzer-Prozess hoher **Priorität** braucht nicht nur weniger lange zu warten, bis er an die Reihe kommt, er kann sogar einen in Arbeit befindlichen Benutzer-Prozess niedriger Priorität unterbrechen (priority-based preemptive scheduling), was normalerweise nicht möglich ist. Das Vorkommando `rtprio(1)` gibt ähnlich wie `nice(1)` einem Programm eine Echtzeit-Priorität mit, die während des Laufes vom System nicht verändert wird, aber vom Benutzer geändert werden kann.

Ein Prozess kann mittels des Systemaufrufs `plock(2)` seinen Platz im Arbeitsspeicher blockieren (**memory locking**), so daß er nicht durch Swapping oder Paging ausgelagert wird. Das trägt zu kurzen, vorhersagbaren Antwortzeiten bei und geht zu Lasten der nicht privilegierten Prozesse, falls der Arbeitsspeicher knapp ist.

Die Standard-UNIX-Uhr, die vom `cron`-Dämon benutzt wird, hat eine Auflösung von einer Sekunde. Zu den Echtzeit-Erweiterungen gehören prozesseigene Uhren (**interval timer**) mit im Rahmen der Möglichkeiten der Hardware definierbarer Auflösung bis in den Mikrosekundenbereich hinunter. Bei unserer Anlage beträgt die feinste Zeitauflösung 10 ms.

Die Verbesserungen am Datei-System und an der Ein- und Ausgabe sind Einzelheiten, die wir übergehen. Die Unterbrechung von Kernprozessen durch Benutzerprozesse, die normalerweise nicht erlaubt ist, wird durch entsprechende Prioritäten der Benutzerprozesse und Soll-Bruchstellen der Kernprozesse ermöglicht (preemptable kernel). Diese weitgehenden Eingriffe in den Prozessablauf setzen strikte Regelungen für die Programme voraus, die auf einer allgemeinen UNIX-Anlage nicht durchzusetzen sind. Deshalb bemerkt und braucht der normale Benutzer, der Texte bearbeitet, Aufgaben rechnet und die Netnews liest, die Echtzeit-Erweiterungen nicht. Auf einem Prozessrechner haben sie den Vorteil, daß man in der gewohnten UNIX-Welt bleiben kann und nicht ein besonderes Echtzeit-Betriebssystem benötigt.

2.13 UNIX auf PCs

2.13.1 MINIX

Das Betriebssystem UNIX war in seinen ersten Jahren kein kommerzielles Produkt, sondern wurde gegen eine Schutzgebühr an Universitäten und andere Interessenten im Quellcode weitergegeben, die ihrerseits viel zur Weiterentwicklung beitrugen, insbesondere die University of California at Berkeley (UCB).

Also verwandte Professor ANDREW S. TANENBAUM von der Freien Universität Amsterdam UNIX zur Untermalung seiner Vorlesung über Betriebssysteme. Als AT&T mit UNIX Geld verdienen wollte und die Weitergabe des Codes einschränkte, stand er plötzlich ohne ein Beispiel für seine Vorlesung da, und nur Theorie wollte er nicht predigen.

In seinem Zorn setzte er sich hin und schrieb ein neues Betriebssystem für den IBM PC, das sich zum Benutzer wie UNIX verhielt, schön pädagogisch und übersichtlich aufgebaut und unabhängig von den Rechtsansprüchen irgendwelcher Pfeffersäcke war. Dieses Betriebssystem heißt MINIX und war von jedermann vom Verlag *Prentice-Hall* für 300 DM zu erwerben. Es lief zur Not auf einem 8088-Prozessor mit einem Floppy-Laufwerk; eine Installation auf einem PC mit 80386SX und IDE-Platte ging reibungslos vonstatten. Die zugehörige Beschreibung steht in TANENBAUMS Buch *Operating Systems*.

MINIX ist durch Urheberrecht (Copyright) geschützt; es ist nicht Public Domain und auch nicht Teil des GNU-Projektes. Der Inhaber der Rechte – der Verlag *Prentice Hall* – gestattet jedoch Universitäten, die Software für Zwecke des Unterrichts und der Forschung zu kopieren. Er gestattet weiter Besitzern von MINIX, die Software zu verändern und die Änderungen frei zu verbreiten, was auch in großem Umfang geschah. Die MINIX-Usenet-Gruppe (`comp.os.minix`) zählte etwa 25000 Mitglieder. In den letzten Jahren ist

MINIX als das UNIX des Bettelstudenten von Linux und weiteren freien UNIXen überholt worden. Ehre seinem Andenken.

2.13.2 Linux

2.13.2.1 Entstehung

Um die erweiterten Fähigkeiten des Intel-80386-Prozessors zu erkunden, begann im April 1991 der finnische Student LINUS BENEDICT TORVALDS[49], unter MINIX kleine Assembler-Programme zu schreiben. Eines seiner ersten Programme ließ zwei Prozesse die Buchstabenfolgen AAAA... und BBBB... auf dem Bildschirm ausgeben. Bald fügte er einen Treiber für die serielle Schnittstelle hinzu und erhielt so einen einfachen Terminalemulator. Zu diesem Zeitpunkt entschloß er sich, mit der Entwicklung eines neuen, freien UNIX-Betriebssystemes zu beginnen. In der Newsgruppe `comp.os.minix` veröffentlichte er seinen Plan und fand bald interessierte Mitstreiter auf der ganzen Welt, die über das Internet in Verbindung standen. Von `ftp.funet.fi` konnten sie sich die erste Kern-Version 0.01 herunterladen.

Am 5. Oktober 1991 gab LINUS die Fertigstellung des ersten offiziellen Kerns 0.02 bekannt. Er benötigte immer noch MINIX als Basissystem. Nur drei Monate vergingen, bis mit der Linux-Version 0.12 ein brauchbarer Kern verfügbar war, der stabil lief. Mit dieser Version setzte eine schnelle Verbreitung von Linux ein.

Die Entwicklung ging weiter zügig voran. Es folgte ein Versionssprung von 0.12 nach 0.95; im April 1992 konnte erstmals das X Window System benutzt werden. Im Verlauf der nächsten zwei Jahre wurde der Kern um immer mehr Features ergänzt, sodaß LINUS am 16. April 1994 die Version 1.0 herausgeben konnte. Die neue Versionsnummer sollte widerspiegeln, daß aus dem einstigen Hacker-UNIX ein für den Endanwender geeignetes System entstanden war.

Seitdem hat Linux weiter an Popularität gewonnen und dabei auch seinen Ziehvater MINIX (von dem jedoch keine einzige Zeile Code übernommen wurde) weit hinter sich gelassen. Im Jahr 1996 begann mit der Versionsnummer 2.0.0 eine neue Kern-Generation, die mit ihren Fähigkeiten selbst kommerziellen Betriebssystemen Konkurrenz macht. Die Stabilität und Leistungsfähigkeit von Linux kann man daran erkennen, daß Linux heute auch auf zentralen Servern eingesetzt wird, von deren Funktionieren ein ganzes LAN abhängt. Es sind auch schon mit Erfolg mehrere Linux-Maschinen zu einem Cluster zusammengeschaltet worden – Stichwort *Beowulf* – um parallelisierten Programmen die Rechenleistung vieler Prozessoren zur Verfügung zu stellen. Im *Guiness Book of Records* kann man den Stand der Dinge nachlesen.

An dieser Stelle eine Anmerkung zu den Versionsnummern der Linux-Kerne. Sie bestehen heutzutage aus drei Zahlen. Ist die mittlere Zahl ungerade, so handelt es sich um einen Entwickler-Kern mit einigen möglicherweise

[49]Inzwischen Ehrendoktor der Universität Stockholm.

instabilen Features. Andernfalls liegt ein Benutzerkernel vor, dessen Codebasis weitgehend stabil ist. Während wir die letzten Tippfehler in unserem Manuskript jagen, erscheint der Kern 2.2.0, also ein Benutzer-Kern.

2.13.2.2 Eigenschaften

Kein anderes Betriebssystem unterstützt so viele Dateisysteme und Netzprotokolle wie Linux, nur wenige so viele verschiedene Rechner-Architekturen. Linux gibt es für:

- PCs mit x86-kompatiblen Prozessoren ab 80386 und den Bussystemen ISA, EISA, VLB, PCI und neuerdings auch MCA
- Workstations auf der Basis von DECs Alpha-Prozessor
- Sun SPARC-Rechner
- verschiedene Plattformen, die Motorolas 680x0-Prozessoren verwenden, darunter einige Atari- und Amiga-Systeme sowie bestimmte ältere Apple Macintosh-Rechner
- PowerPCs und PowerMacs

Darüberhinaus sind Portierungen auf weitere Plattformen wie StrongARM, MIPS und PA-RISC mehr oder weniger weit gediehen.

Den Datenaustausch mit einer Vielzahl von anderen Betriebssystemen ermöglichen Kern-Treiber für die folgenden Dateisysteme:

- Extended 2 FS, das leistungsfähige Linux-eigene Dateisystem
- das Dateisystem von MINIX
- FAT, das Dateisystem von PC-DOS, einschließlich der langen Dateinamen von Microsoft Windows 95 (VFAT) und dem neuen FAT32
- HPFS, das Dateisystem von IBM OS/2 (leider nur lesend)
- NTFS, Microsofts Dateisystem für Windows NT
- das System V-Dateisystem, welches von SCO UNIX, Xenix und Coherent verwendet wird
- das BSD-Dateisystem UFS, verwendet von SunOS, FreeBSD, NetBSD und NextStep
- das Amiga-Dateisystem AFFS
- HFS, das Dateisystem von MacOS
- ADFS, verwendet auf Acorn StrongARM RISC PCs
- ein Dateisystem für ROMs und Boot-Images (ROMFS)
- NFS, das unter UNIX übliche Netz-Dateisystem
- CODA, ein möglicher Nachfolger von NFS
- SMB, Microsofts Netz-Dateisystem
- NCP, das Netz-Dateisystem von Novell Netware

Zur Zeit entstehen gerade einige Kern-Module, die die sichere Verschlüsselung von Dateisystemen erlauben.

Linux beherrscht die Internet-Protokolle TCP/IP, Novells IPX und das in der Mac-Welt übliche AppleTalk. Darüberhinaus ist ein Treiber für das im Packet Radio Netz der Funkamateure eingesetzte Protokoll AX.25 enthalten. Neben Daemonen fuer die UNIX-üblichen Protokolle ist ein Server für das von Microsoft Windows verwendete Protokoll SMB erhältlich (Samba) und ein mit Novell Netware kompatibler Datei- und Druckerserver (Mars); sogar für die Mac-Welt gibt es ein Serverpaket.

Und wie sieht es mit der Hardware-Unterstützung aus? Linux unterstützt heute die alle gängigen SCSI-Hostadapter und Netzkarten, dazu einige ISDN- und Soundkarten. Selbst exotische Hardware wie bestimmte Video-Karten und 3D-Beschleuniger wird neuerdings unterstützt.

Will man X11 verwenden (und wer will das nicht), sollte man darauf achten, daß man eine von XFree durch einen besonderen, beschleunigten Server unterstützte Graphik-Karte erwirbt.

Es dauert im allgemeinen jedoch einige Zeit, bis Treiber für neue Hardware entwickelt sind, und nicht alle Hardware kann unterstützt werden, weil einige Hersteller die technischen Daten nur zu nicht annehmbaren Konditionen (Non Disclosure Agreements) bekanntgeben. Faßt man die Installation von Linux ins Auge, sollte man daher unbedingt schon vor dem Kauf der Hardware auf Unterstützung achten. Das Hardware-HOWTO stellt hierbei eine nützliche Hilfe dar. Im Zweifelsfall im Netz fragen.

2.13.2.3 Distributionen

Da es umständlich und oft schwierig, wenn auch lehrreich ist, alle für ein vollständiges Linux/UNIX-System erforderlichen Komponenten selbst zusammenzusuchen und zu kompilieren, entstanden schon früh sogenannte **Distributionen**, die den Linux-Kern mitsamt vieler nützlicher Anwendungen in vorkompilierter Form enthalten. Dazu kommt meist ein einfach zu benutzendes Installations-Programm. Zu den bekannteren Distributionen zählen:

- Caldera (`www.caldera.com/`), kommerziell,
- Debian (`www.debian.org/`),
- Knoppix (`www.knopper.net/knoppix/`),
- Mandrake (`www.linux-mandrake.com/`), nennt sich seit einger Zeit Mandriva,
- Red Hat (`www.redhat.com/`),
- Slackware (`www.slackware.org/`),
- Stampede (`www.stampede.org/`),
- SuSE (`www.suse.de/`),
- TurboLinux (`www.turbolinux.com/`),
- Tuxtops (`www.tuxtops.com/`), für Laptops,

- MuLinux (`mulinux.nevalabs.org/`), ein minimales Linux für Diskettenbetrieb.

Heute gibt es über 200 Distributionen, aber nur wenige sind verbreitet. Die Distributionen unterscheiden sich im Umfang der mitgelieferten Anwendungen und in der Einrichtung.

Wir haben gute Erfahrungen mit **Red Hat** gemacht, aber die anderen Distributoren ziehen nach. **SuSE**, mittlerweile von Novell übernommen, bringt viele Anwendungen mit, **Debian**[50] ist vorbildlich organisiert und unterscheidet deutlich zwischen freier und bedingt freier Software, Knoppix lässt sich vollständig auf einer bootfähigen CD einrichten, Mandrake bietet einen für viele Bedürfnisse geeigneten Kompromiss aus Umfang und einfacher Einrichtung und verwendet zudem die Red Hat Programm Module (RPM). Das von Red Hat entwickelte RPM-System ermöglicht ein einfaches Einrichten, Updaten und weitgehend rückstandsfreies Entfernen von Software-Paketen. Ein ähnliches System verwendet auch Debian. Diese Systeme erleichtern dem Systemverwalter das Leben ungemein. Allerdings gibt es nicht für alle Anwendungsprogramme ein fertiges rpm- oder deb-Paket. Solche Programme müssen nach wie vor von Hand installiert werden, was Kenntnisse voraussetzt, zumindest aber das Lesen der beigefügten Dokumentation (README, INSTALL usw.).

Eine Besonderheit sind minimale Linux-Distributionen, die auf ein oder zwei Disketten Platz finden (Tiny Linux). Man darf natürlich nicht den vollen Funktionsumfang erwarten – insbesondere fehlt meist X11 – aber für manche Aufgaben ist eine solche Magerversion ausreichend. Eine Sammlung ist auf:

`http://www.tux.org/pub/distributions/tinylinux/`

zu finden, darunter HAL91, LOAF, Small Linux und Mulinux.

Die meisten Distributionen sind kostenlos über das Internet zu beziehen. Viele lassen sich sogar direkt aus dem Netz installieren. Dennoch hat der Erwerb einer CD-ROM oder DVD Vorteile: Man benötigt keine Internet-Verbindung (die im Normalfall bei Privatleuten ohnehin zu langsam für die Installation ist) und kann jederzeit Software-Pakete nachinstallieren. Der Preis, den man für eine Distribution entrichtet, deckt einerseits die Kosten für die Herstellung der CD oder DVD und des Begleitmaterials, andererseits unterstützt er die Hersteller der Distribution, die bei ihrer Arbeit auf das Geld aus dem Verkauf angewiesen sind. Die Software selbst ist frei. Für kommerzielle Erweiterungen wie Motif oder CDE gilt das natürlich nicht.

2.13.2.4 Installation

Die Einrichtung verläuft bei den meisten Distributionen dank ausgereifter Installationsskripte weitgehend einfach. Im allgemeinen müssen zunächst ein oder zwei Installations-Disketten erstellt werden, wobei darauf zu achten

[50]Zu Debian GNU/Linux finden sich ausführliche Informationen in dem Buch von PETER H. GANTEN und WULF ALEX, siehe Anhang.

ist, dass nur fehlerfreie, DOS-formatierte Disketten verwendet werden. Anschließend wird von der Boot-Diskette ein einfaches Linux-System gestartet. Nun erfolgt die Auswahl des Installationsmediums. Disketten sind beim Umfang der heutigen Distributionen selten, meist erfolgt die Installation von CD oder von einem Verzeichnis auf einer PC-DOS-Partition. Viele Distributionen erlauben aber auch die Installation von einem NFS-Volume, einer SMB-Share oder einem Anonymous-FTP-Server über das Netz.

Der nächste Schritt besteht im Anlegen von Partitionen für Linux. Viele Installationsskripte greifen hierzu auf das spartanische fdisk-Programm von Linux (nicht zu verwechseln mit dem von PC-DOS) zurück, zum Teil finden aber auch einfach zu benutzende Partitionierungstools (z. B. Disk Druid) Verwendung. Normalerweise legt man eine Partition für das Root-Datei-System und eine Swap-Partition an. Diese stellt zusätzlichen virtuellen Arbeitsspeicher zur Verfügung, falls der echte Hauptspeicher einmal nicht ausreichen sollte, ist aber nur eine Notlösung. Wie groß sie sein sollte, hängt vom beabsichtigten Einsatz des Systems ab, bei normalen Linux-Workstations sind 16-32 MB vollkommen ausreichend. Eventuell will man neben dem Root-Datei-System weitere Partitionen anlegen, zum Beispiel für die Home-Verzeichnisse der Benutzer. Die meisten Installationsskripte fragen nun, welche Partitionen wohin gemountet werden sollen; dabei können auch PC-DOS- und HPFS-Partitionen angegeben werden.

Der Hauptteil der Installation besteht im Auswählen der zu installierenden Programm-Pakete. Intelligente Skripte fragen zuerst, was installiert werden soll, und installieren dann die ausgewählten Pakete, während weniger ausgereifte Skripte vor der Installation jedes Pakets einzeln nachfragen, was während der gesamten Installationsphase Mitarbeit erfordert. Für Linux-Anfänger ist es zumeist schwierig zu entscheiden, was benötigt wird und was nicht. Dabei sind die Kurzbeschreibungen der Pakete eine gewisse Hilfestellung. Man kann aber problemlos nachinstallieren.

Schließlich fragen die meisten Installationsskripte noch einige Systemeinstellungen ab. Dazu zählen die Zeitzone, das Tastaturlayout, der Maustyp sowie die nötigen Angaben für die TCP/IP-Vernetzung. Diese lassen sich jederzeit nachträglich ändern.

Außerdem bieten die meisten Distributionen an dieser Stelle die Gelegenheit, den Linux-Loader LILO als Boot-Manager einzurichten – neuerdings zunehmend GRUB, den *Grand Unified Bootloader* – sodass man beim Booten zwischen Linux und anderen Betriebssystemen auswählen kann. Mit einigen Tricks lassen sich aber auch die Boot-Manager von IBM OS/2 und Microsoft Windows NT dazu überreden, Linux zu booten.

Mit etwas Glück kann man dann sein frischerstelltes Linux-System starten. Zu den ersten Schritten sollte das Setzen eines `root`-Passworts, das Einrichten von Benutzern und das Kompilieren eines auf die eigenen Bedürfnisse zurechtgeschnittenen Kerns sein. Der von der Distribution angebotene, universelle Kern enthält meist mehr Funktionen, als man braucht. Das kostet unnötig Arbeitsspeicher und kann auch Instabilitäten verursachen.

Um den Kern neu zu kompilieren, wechselt man zunächst in das Verzeichnis `/usr/src/linux`, in dem man mit `make mrproper` erst einmal für Ordnung sorgt. Anschließend müssen die benötigten Treiber ausgewählt werden. Seit einiger Zeit lassen sich viele Treiber auch als **Kernmodule** kompilieren; sie sind dann nicht fester Bestandteil des Kerns, sondern liegen in einer eigenen Datei vor und können je nach Bedarf mit `insmod(1)` geladen und `rmmod(1)` wieder entladen werden. Bei entsprechender Konfiguration kann Linux dies sogar automatisch tun. Durch die Modularisierung weniger häufig benötigter Treiber (z. B. für SCSI-Tapes) spart man während der meisten Zeit Arbeitsspeicher ein. Zur Treiberauswahl gibt es drei Alternativen: Die schlichte Abfrage aller möglichen Komponenten mit `make config`, die menügestützte Abfrage mit `make menuconfig` und (soweit man das X Window System und TCL/TK installiert hat) ein komfortables Konfigurationsprogramm mit `make xconfig`. Im nächsten Schritt werden die Kern-Quellen auf das Kompilieren vorbereitet: `make dep` und `make clean`. Jetzt kann der Kompilationsvorgang mit `make zImage` gestartet werden. Er dauert, je nach Systemleistung und ausgewählten Komponenten, zwischen fünf Minuten und über einer Stunde. Hat man bei der Kern-Konfiguration angegeben, einige Komponenten als Module zu kompilieren, müssen diese noch mit `make modules` erzeugt werden. Der fertige Kern findet sich im Unterverzeichnis `arch/i386/boot` als Datei `zimage`, die Module werden mit `make modules_install` in `/lib/modules` installiert. Eventuell bereits vorhandene Module sollte man vorher in ein anderes Verzeichnis verschieben.

Zur Installation des Kerns ist die Datei `/etc/lilo.conf` zu editieren und anschließend durch Aufruf des Programms `lilo(8)` der Linux-Loader neu zu installieren.

2.13.2.5 GNU und Linux

Viele der Programme, ohne die Linux kein vollständiges UNIX-System wäre, entstanden im Rahmen des GNU-Projekts der Free Software Foundation. Neben zahllosen kleinen, aber nützlichen oder sogar systemwichtigen Utilities wie GNU `tar`, GNU `awk` usw. zählen hierzu:

- `gzip`, das GNU Kompressions-Utility,
- `bash`, die Bourne-Again Shell,
- `emacs`, der große Editor,
- `gcc`, der GNU-C/C++-Compiler, ohne den Linux nie entstanden wäre,
- `glibc`, die GNU-C-Funktionsbibliothek (bekannt unter den Bezeichnungen glibc2 aka libc6), die nach und nach die alte Linux-C-Bibliothek (libc5) ersetzen wird.

Darüber hinaus unterliegen viele Programme, die unter Linux benutzt werden, der GNU Public License (GPL). Hierzu zählt auch der Linux-Kern selbst.

2.13.2.6 XFree - X11 für Linux

Linux wäre keine Alternative zu anderen modernen Betriebssystemen ohne eine grafische Benutzeroberfläche. Diese kommt in Gestalt des auf UNIX-Systemen üblichen X Window Systems (X11). Soweit man nicht einen kommerziellen X-Server vorzieht, was in den meisten Fällen nicht lohnenswert ist und oft sogar noch zusätzlichen Aufwand bei der Systemverwaltung erfordert, wird man die Implementation von XFree (`http://www.xfree86.org/`) verwenden. Diese besteht einerseits aus X-Servern, darunter verschiedene beschleunigte für bessere Grafik-Karten, andererseits aus einigen Utilities.

indexFenster-Manager Produktiv einsetzbar wird X11 erst durch einen guten **Fenster-Manager**. Hier bietet Linux eine Vielzahl von Möglichkeiten. Am weitesten verbreitet ist wahrscheinlich FVWM, der durch seinen Microsoft-Windows-95-Look Umsteigern eine vertraute Oberfläche bietet. Das kommerzielle Motif-Paket mit seinem eigenen Motif Window Manager gibt es natürlich auch für Linux; es durch eine kompatible, aber freie Widget-Bibliothek und einen Fenster-Manager zu ersetzen, ist das Ziel des LessTif-Projekts. Darüberhinaus existieren einige exotische Fenster-Manager wie Enlightenment und Afterstep, der dem Linux-Desktop das Look + Feel von Next-Step verleihen soll.

2.13.2.7 K Desktop Environment (KDE)

Eine ausgereifte und verbreitete Benutzeroberfläche für Linux und andere UNIX-Betriebssysteme stellt das K Desktop Environment (`http://www.kde.org/`) dar. KDE ist mehr als nur ein Fenster-Manager. Es ist vielmehr eine integrierte Obefläche, die dem Benutzer durch Eigenschaften wie Cut & Paste, Drag & Drop und Kontext-Menüs, aber auch durch neue Programme wie einen sehr leistungsfähigen Datei-Manager, der zugleich ein Web-Browser ist, sowie vielen kleinen graphischen Utilities, die zum Beispiel das bequeme Konfigurieren der Desktops ermöglichen, eine moderne und intuitive Arbeitsumgebung bereitstellt.

So zeigt KDE, daß Linux/UNIX nicht immer kryptische Konfigurationsdateien und für Anfänger schwierig zu benutzende Programme bedeuten muss und beseitigt damit ein Defizit, das bisher viele Benuzter vom Umstieg auf eines der freien Linux/UNIXe ohne das kommerzielle CDE abhielt.

Neben der eigentlichen Obefläche gibt es bereits eine große Menge an Programmen, die von den erweiterten Möglichkeiten von KDE Gebrauch machen. Einige davon, darunter ein einfacher Text-Editor, ein Email-Client, ein Newsreader sowie der K Configuration Manager sind in der offiziellen KDE-Distribution erhalten. Darüberhinaus existieren viele weitere nützliche Programme wie KISDN, welches die einfache Einrichtung eines Internet-Zuganges über ISDN gestattet, oder KMPG, ein Wiedergabe-Programm für das Sound-Dateiformat MPEG 1 Layer 3 (MP3).

KDE setzt auf dem von der norwegischen Firma Troll Tech AS, Oslo (`http://www.troll.no/`) entwickelten **Quasar-Toopkit** (Qt) auf. Diese ist für freie Linux/UNIX-Anwendungen frei verfügbar und enthält eine Sammlung von Widgets für Entwickler von grafischen Benutzer-Oberflächen unter X11 und Microsoft Windows. Die Einarbeitung in die Qt- und KDE-Bibliotheken stellt für einigermaßen erfahrene C++-Programmierer kein Problem dar, die Leistungsfähigkeit und das intelligente Design ermöglichen es, recht schnell graphische Benutzeroberflächen zu gestalten und bereiten dem Einsteiger somit bald Erfolgserlebnisse. Im kommerziellen Einsatz kostet die Qt-Biliothek etwas, auch norwegische Trolle müssen ihren Lebensunterhalt verdienen.

Eine weitere Arbeitsumgebung für Linux und andere UNIXe stellt das auf dem GIMP ToolKit (gtk+) basierende **GNU Network Object Model Environment** (GNOME) dar. Sowohl GNOME als auch das GTK unterliegen nur GNU-Lizenzen und sind somit im privaten wie kommerziellen Einsatz frei.

2.13.2.8 Dokumentation

Als freies Betriebssystem kommt Linux in den meisten Fällen ohne gedruckte Dokumentation daher. Viele Distributionen enthalten zwar ein einfaches Handbuch, das aber nur die Installation und die einfachsten Verwaltungsaufgaben erklärt. Dafür ist die Online-Dokumentation erheblich besser als die der meisten kommerziellen Betriebssysteme.

Neben den oft benötigten `man`-Pages, die man von einem UNIX-System erwartet, sind es vor allem die zu vielen verschiedenen Aspekten von Linux verfügbaren, sehr hilfreichen HOWTOs und Mini-HOWTOs, die für den System-Manager, aber auch den Endanwender interessant sind. Sie werden von sogenannten Maintainern gepflegt und weisen eine übersichtliche Gliederung auf. Vom Umfang her noch geeignet, eine kurze Einführung in ein bestimmtes Gebiet zu geben, fassen sie alle wesentlichen Informationen zusammen. Sie sind von `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/` zu bekommen, allerdings ist dieser Host hoch belastet, so daß man sich einen Mirror in der Nähe suchen sollte, siehe `http://sunsite.unc.edu/pub/Linux/MIRRORS.html#Europe`. Zu den wichtigeren HOWTOs gehören:

- das DOS-to-Linux-HOWTO mit Hinweisen, wie man von PC-DOS zu Linux wechselt,
- das German-HOWTO, das Tips für deutsche Benutzer gibt,
- das Hardware-HOWTO, das eine nicht unbedingt aktuelle Liste der von Linux unterstützten Hardware enthält,
- das Kernel-HOWTO bei Fragen zum Kern, insbesondere zum Kompilieren des Kerns,
- das NET-2-HOWTO mit Hilfen zur Netzkonfiguration,
- das Distribution-HOWTO mit einer Übersicht über die Linux-Distributionen,

insgesamt rund hundert HOWTOs und hundert Mini-HOWTOs. Bei Problemen sollte man also zuerst einmal einen Blick in `/usr/doc/HOWTO` werfen.

Die Chancen, daß ein anderer das Problem schon gelöst hat, stehen nicht schlecht.

Darüberhinaus entstehen im Rahmen des Linux Documentation Projects (LDP)[51] verschiedene umfangreiche Dokumentationen, die einen großen Bereich der Systemverwaltung wie die Einrichtung von Netzen, das Schreiben von Kern-Treibern usw. abdecken. Zu den Veröffentlichungen des LDP zählen:

- der Linux Programmer's Guide,
- der Network Administrator's Guide,
- der System Administrator's Guide.

Die meisten Dokumentations-Dateien sind im Verzeichnis `/usr/doc` abgelegt. Im WWW finden sie sich auf `sunsite.unc.edu/mdw/linux.html`. Von dort gelangt man auch zu FAQs und weiteren Veröffentlichungen. Auf unserer WWW-Seite `www.ciw.uni-karlsruhe.de/technik.html` ist Linux natürlich auch gut vertreten.

Aktive Unterstützung bei Problemen erhält man im Internet in den Linux-Newsgruppen (`comp.os.linux.*`, `linux.*`). Bei der Auswahl der richtigen Newsgruppe für eine Frage sollte man darauf achten, daß solche Fragen, die nicht speziell Linux betreffen, sondern ein Programm, das auch auf anderen UNIX-Systemen verfügbar ist, häufig nicht in die Linux-Hierarchien gehören.

2.13.2.9 Installations-Beispiel

Abschließend sei noch als Beispiel der Einsatz eines Linux-Rechners genannt, der unser Hausnetz (Domestic Area Network, DAN) mit dem Internet verbindet. Diese Konfiguration dürfte auf viele kleinere Netze zutreffen, beispielsweise in Schulen. Der Rechner selbst ist ein PC 486-120 mit 32 MB RAM und 1 GB Festplatte, also ein recht genügsames System. Er verfügt über eine Ethernet-Karte am hauseigenen DAN und eine ISDN-Karte für die Verbindung zum Rechenzentrum einer Universität, das den Provider spielt.

Als besonders nützlich hat sich die Fähigkeit des Linux-Kerns erwiesen, ein ganzes Subnetz hinter einer einzigen IP-Adresse zu verstecken (IP Masquerading), was neben der Schonung des knapp werdenden Adressraums auch einen Sicherheitsvorteil mit sich bringt. Die Masquerading-Funktion von Linux bietet mittlerweile sogar Unterstützung für Protokolle wie FTP, IRC und Quake, die eine besondere Umsetzung erforden.

Um den Internet-Zugang zu entlasten, laufen auf dem Linux-Rechner ein Proxy (`squid`), der WWW-Seiten zwischenspeichert für den Fall, daß sie mehrmals angefordert werden sollten, sowie ein News-Server (Leafnode), der uns das Lesen einiger ausgewählter Newsgruppen ohne Internet-Verbindung (offline) ermöglicht. Jede Nacht werden automatisch wartende Emails sowie neue News-Artikel abgeholt. Darüber hinaus dient der Linux-Rechner auch als Fax-Server, sowohl für eingehende als auch ausgehende Fax-Nachrichten, als Datei-Server, wobei neben NFS auch das Microsoft-Windows-Protokoll

[51] Siehe bespielsweise `http://www.leo.org/archiv/software/unix/linux/`.

SMB unterstützt wird, und als Druckerserver. Das System läuft bei uns seit Mitte 1997 und hat sich auch unter harten Bedingungen (was die Internet-Nutzung angeht) bewährt.

Als Clients greifen von den Arbeitsplätzen aus Computer unter Linux, NetBSD, Novell DOS und Microsoft Windows NT 4.0 auf den Linux-Server zu. Die beiden UNIX-Systeme verfügen selbstverständlich über alle UNIX-üblichen Internet-Programme, für PC-DOS gibt es ebenfalls Clients für Telnet, FTP und den Textmode-WWW-Browser Lynx, darüberhinaus sogar einen X-Server und einen Telnet-Server. Unter Microsoft Windows werden viele Internet-Programme wie Microsoft Explorer, Netscape Navigator, FTP-Clients und Real-Audio verwendet.

2.13.3 386BSD, NetBSD, FreeBSD ...

386BSD ist ein UNIX-System von WILLIAM FREDERICK JOLITZ und LYNNE GREER JOLITZ für Prozessoren ab 80386 aufwärts, ebenfalls copyrighted, für private Zwecke frei nutzbar und darf nicht mit dem kommerziellen Produkt BSD/386 verwechselt werden. Der Original Point of Distribution ist `agate.berkeley.edu`, gespiegelt von `gatekeeper.dec.com` und anderen. 386BSD entwickelt sich langsamer als Linux und unterstützt eine zum Teil andere Hardwareauswahl als dieses. Näheres in der Zeitschrift IX 1992, Nr. 5, S. 52 und Nr. 6, S. 30.

NetBSD, **OpenBSD** und **FreeBSD** sind ebenfalls UNIX-Systeme aus Berkeley, die verwandt mit 386BSD sind und darauf aufbauen; genauso für nichtkommerzielle Zwecke kostenfrei nutzbar. NetBSD ist auf eine große Anzahl von Prozessortypen portiert worden. Worin die Unterschiede liegen, auch zu Linux, wie die Zukunft aussieht und wer wo mitarbeitet, ist schwierig zu ermitteln. Archie oder das WWW fragen:

- `http://www.freebsd.org`,
- `http://www.netbsd.org`,
- `http://www.openbsd.org`.

The galaxy is a rapidly changing place, schreibt DOUGLAS ADAMS.

2.14 Systemverwaltung

Ein Betriebssystem wie UNIX lässt sich von drei Standpunkten aus betrachten, von dem

- des Benutzers,
- des System- und Netz-Managers (Verwalters),
- des System-Entwicklers.

Der **Benutzer** möchte eine möglichst komfortable und robuste Oberfläche für die Erledigung seiner Aufgaben (Anwenderprogramme, Textverarbeitung, Information Retrieval, Programmentwicklung) vorfinden. Die Benutzer könnte man noch unterteilen in solche, die nur fertige Anwendungen benutzen, und solche, die programmieren, aber das ist nebensächlich. Der **System-Manager** will sein System optimal an die vorliegenden Aufgaben anpassen und einen sicheren Betrieb erreichen. Der **System-Entwickler** muss sich mit Anpassungen an neue Bedürfnisse (Netze, Parallelrechner, Echtzeitbetrieb, neue Hardware), mit Fragen der Portabilität und der Standardisierung befassen. Während sich die bisherigen Abschnitte mit UNIX vom Standpunkt des Benutzers aus beschäftigt haben, gehen wir nun zum Standpunkt des System-Managers über. Dank Linux, FreeBSD und Kompanie hat jeder PC-Besitzer die Möglichkeit, diesen Standpunkt auch praktisch einzunehmen. Sogar kleinere Familiennetze sind technisch und finanziell in den Rahmen des Möglichen gerückt.

Zum Teil braucht auch der gewöhnliche Benutzer eine ungefähre Vorstellung von den Aufgaben des System-Managers, zum Teil muss (oder darf) er – vor allem auf kleineren Anlagen – diese Tätigkeiten selbst durchführen, in den Zeiten von Linux und BSD häufiger als früher. Ein System-Manager kommt um das gründliche Studium der Handbücher und eine ständige Weiterbildung nicht herum[52].

Die Systempflege ist die Aufgabe des **System-Managers** oder **System-Administrators**. Auf UNIX-Maschinen lautet sein Benutzername traditionell `root`[53]. In Novell-Netzen heißt der Mensch *Supervisor*. Er braucht die Vorrechte des **Superusers**, der stets die Benutzer-ID 0 trägt. Die Bezeichnungen *System-Manager* und *Superuser* werden oft synonym gebraucht, der Begriff *System-Manager* ist jedoch von der Aufgabe her definiert und daher treffender. Bei großen Anlagen findet man noch die **Operatoren**. Sie sind unmittelbar für den Betrieb zuständig, überwachen die Anlage, beseitigen Störungen, wechseln Datenträger, haben aber weniger Aufgaben in Planung, Konfiguration oder Programmierung.

2.14.1 Systemgenerierung und -update

Unter einer **Systemgenerierung** versteht man die Erstinstallation des Betriebssystems auf einer neuen Anlage oder die erneute Installation des Betriebssystems auf einer Anlage, die völlig zusammengebrochen und zu keiner brauchbaren Reaktion mehr fähig ist. Auch die Umpartitionierung der `root`-Platte erfordert eine Generierung.

Ein **System-Update** ist die Nachführung eines laufenden Systems auf eine neuere Version des Betriebssystems oder eine Erweiterung – unter Umständen auch Verkleinerung – des Betriebssystems. Die Hinzunahme weiterer Hardware oder eines Protokolles erfordert eine solche Erweiterung. Eine Erweiterung ohne Änderung der Version wird auch **System-Upgrade** genannt.

Alle drei Aufgaben sind ähnlich und im Grunde nicht schwierig. Da man aber derartige Aufgaben nicht jede Woche erledigt und sich das System zeitweilig in einem etwas empfindlichen Zustand befindet, ist die Wahrscheinlich-

[52] Experten wissen wenig, Halb-Experten alles.

[53] In seltenen Fällen `avatar`.

keit *sehr* hoch, dass etwas schiefgeht und man erst nach mehreren Versuchen Erfolg hat:

- Der erste Versuch geht völlig daneben, aber man lernt den Ablauf der Installation kennen und entwickelt Vorstellungen, was mit den Fragen und Meldungen gemeint sein könnte.
- Der zweite Versuch führt zu einem lauffähigen System, das aber noch nicht den Vorstellungen entspricht.
- Der dritte Versuch gelingt im großen Ganzen.

Deshalb soll man den Zeitpunkt für diese Arbeit so wählen, dass eine längere Sperre des Systems von den Benutzern hingenommen werden kann. Der System-Manager sollte sich vorher noch einmal gut ausschlafen und seinen Vorrat an Kaffee und Schokolade auffüllen.

Hat man ein laufendes System mit wertvollen Daten, ist der erste Schritt ein vollständiges Backup. Dabei ist es zweckmäßig, nicht das gesamte Datei-System auf einen oder eine Folge von Datenträgern zu sichern, sondern die obersten Verzeichnisse (unter `root`) jeweils für sich. Das erleichtert das gezielte Wiederherstellen. `/tmp` beispielsweise braucht überhaupt nicht gesichert zu werden, `/dev` sollte man zwar sichern, spielt es aber in der Regel nach einer Systemänderung nicht zurück, weil es entsprechend den Änderungen neu erzeugt wird. Weiterhin sollte man schon im täglichen Betrieb darauf achten, dass alle für die jeweilige Anlage spezifischen Dateien in wenigen Verzeichnissen (`/etc`, `/usr/local/bin`, `/usr/local/etc`, `/usr/local/config`, `/var`, `/opt` usw.) versammelt und erforderlichenfalls nach `/bin` oder `/etc` gelinkt sind. Nur so lässt sich nach einer Systemänderung ohne viel Aufwand entscheiden, was aus den alten und was aus den neuen Dateien übernommen wird. Gerade im `/etc`-Verzeichnis sind viele Konfigurations-Dateien zu Hause, die nach einer Systemänderung editiert werden müssen, und da ist es gut, sowohl die alte wie die neue Fassung zu haben. Es ist auch beruhigend, die obersten Verzeichnisse und die systemspezifischen Textfiles auf Papier zu besitzen.

Der nächste Schritt ist das Zurechtlegen der Handbücher und das Erkunden der Hardware, insbesondere des I/O-Subsystems. Falls man keine Handbücher hat, sondern nur mit dem `man(1)`-Kommando arbeitet, drucke man sich die Beschreibung der einschlägigen Kommandos auf Papier aus, es sei denn, man habe ein zweites System derselben Art. Wichtig sind auch die beim Booten angezeigten Hardware-Adressen für den Primary Boot Path und den **Alternate Boot Path**, bei uns 4.0.0.0.0.0 und 4.0.2.1.0.0. Ferner sollte die **Konsole** von dem Typ sein, mit dem die Anlage am liebsten zusammenarbeitet (bei uns also Hewlett-Packard). Dann wirft man alle Benutzer und Dämonen hinaus und wechselt in den Single-User-Modus. Von jetzt ab wird die Installation hardwareabhängig und herstellerspezifisch.

Falls man die neuen Dateien nicht über das Netz holt, kommen sie von einem entfernbaren Datenträger (removable medium) wie Band (Spule oder Kassette) oder CD-ROM über den Alternate Boot Path. Man legt also den Datenträger ein und bootet. Die Boot-Firmware fragt zu Beginn nach dem Boot Path, worauf man mit der Adresse des Alternate Boot Path antwortet.

Dann wird noch gefragt, ob interaktiv gebootet werden soll, was zu bejahen ist. Schließlich meldet sich ein Programm – der **Initial System Loader** ISL – das einige wenige Kommandos versteht, darunter das Kommando zum Booten:

```
hpux -a disc0(4.0.0) disc0(4.0.2.1;0x400020)
```

Eine Beschreibung des Kommandos (Secondary System Loader) findet sich unter `hpux(1M)`. Die Option `-a` bewirkt, dass die I/O-Konfiguration entsprechend der nachfolgenden Angabe geändert wird. `disc0` ist der Treiber für die Platte, `4.0.0` die Hardware-Adresse der Platte, auf der künftig der Boot-Sektor und das `root`-Verzeichnis liegen sollen. `disc0` ist ebenfalls der Treiber für das Kassetten-Bandlaufwerk, von dem das neue System installiert werden soll, `4.0.2.1` seine Hardware-Adresse. `0x400020` ist die Minor Number des Kassetten-Bandlaufwerks und sorgt für eine bestimmte Konfiguration, hat also in diesem Zusammenhang nichts mit einer Adresse zu tun. Das Kommando lädt von dem Installations-Datenträger (Kassette) ein einfaches lauffähiges System in den Arbeitsspeicher.

Dann erscheint – wenn alles gut geht – eine Halbgrafik zur **Partitionierung** der `root`-Platte. Bootsektor, Swap Area und `root` müssen auf derselben Platte liegen, da man zu Beginn des Bootens noch keine weiteren Datei-Systeme gemountet hat. Verzeichnisse wie `/usr/local`, `/opt` oder `/var` neigen während des Betriebs zum Wachsen, die zugehörigen Partitionen sollten genügend Luft aufweisen. Die Homes gehören am besten auf eine eigene Platte, die von einer Systeminstallation gar nicht betroffen ist. Falls man nach der Länge der Dateinamen gefragt wird, sollte man sich für lange Namen (maximal 255 Zeichen) entscheiden.

Im weiteren Verlauf werden viele Dateien auf die Platte kopiert, zwischendurch auch einmal gebootet und erforderlichenfalls der Datenträger gewechselt. Die Dateien werden zu Filesets gebündelt herübergezogen, wobei ein **Fileset** immer zu einer bestimmten Aufgabe wie Kernel, UNIX-Tools, Grafik, Netz, C, FORTRAN, PASCAL, COBOL, Native Language Support gehört, vergleichbar einem Debian- oder Red-Hat-Paket. Teilweise bestehen gegenseitige Abhängigkeiten, die das Installationsprogramm von sich aus berücksichtigt. Man kann sich die Filesets anzeigen lassen und entscheiden, ob sie geladen werden sollen oder nicht. Dinge, die man nicht braucht (Grafik, COBOL, NLS), kann man getrost weglassen, Dinge, für die keine Hardware im Kasten steckt (Netzadapter, bit-mapped Terminals), sind überflüssig. Nur auf den Kernel und die UNIX-Tools sollte man nicht verzichten, auch wenn der Speicherplatz noch so knapp ist. Unter Debian GNU/Linux werden die Dateien zu Paketen zusammengefasst und diese wiederum in Sachgebiete eingeordnet. Üblicherweise zusammengehörende Pakete bilden eine Task (Aufgabe).

Schließlich ist die Übertragung beendet, und man bootet vom Primary Boot Path. Das System läuft und kennt zumindest den Benutzer `root`, dem man sofort ein Passwort zuordnet. Nun beginnt die Feinarbeit mit dem Wiederherstellen der Konfiguration. Auf keinen Fall kopiere man die alten Konfigurationsfiles blindlings über die neuen, das kann zur Bootunfähigkeit und damit zu einem vierten Installationsversuch führen. Zweckmäßig vergleicht man die alten mit den neuen Dateien und editiert die neuen, wo nötig. Vorsichtshalber sollte man von den neuen Dateien vorher eine Kopie ziehen. Zeitweilig hat man also drei Versionen dieser Dateien auf dem System: die originale, die alte und die aktuelle.

2.14.2 Systemstart und -stop

Wenn das System eingeschaltet wird, steht als einziges Programm ein spezielles Test- und Leseprogramm in einem **Boot-ROM** zur Verfügung. Der Computer ist einem Neugeborenen vergleichbar, der noch nicht sprechen, schreiben, lesen und rechnen kann, aber ungeheuer lernfähig ist. Das Programm lädt den **Swapper** (Prozess Nr. 0) von der Platte in den Arbeitsspeicher. Der Swapper lädt das `/etc/init(1M)`-Programm, das die Prozess-ID 1 bekommt und der Urahne aller Benutzer-Prozesse ist.

Der `init`-Prozess liest die Datei `/etc/inittab(4)` und führt die dort aufgelisteten Tätigkeiten aus. Dazu gehören die in der Datei `/etc/rc(1M)` genannten Shell-Kommandos und die Initialisierung der Terminals. In der Datei (Shellskript) `/etc/rc(1M)` werden der Dämon `cron(1M)`, einige **Netzdämonen**, das **Accounting System** und der **Line Printer Scheduler** gestartet und die Gerätefiles für Drucker und Plotter geöffnet. In den letzten Jahren ist – vor allem infolge der Vernetzung – aus der Datei `/etc/rc` eine ganze Verzeichnisstruktur geworden, die bei Start und Stop durchlaufen wird.

Die Terminals werden initialisiert, indem ein Prozess `/etc/getty(1M)` für jedes **Terminal** erzeugt wird. Jeder `getty`-Prozess schaut in der Datei `/etc/gettydefs(4)` nach den Parametern seines Terminals, stellt die Schnittstelle ein und schreibt den login-Prompt auf den Bildschirm.

Nach Eingabe eines Benutzernamens ersetzt sich `getty` durch `/bin/login(1)`, der den Namen gegen die Datei `/etc/passwd(4)` prüft. Dann wird das Passwort geprüft. Sind Name und Passwort gültig, ersetzt sich der `login`-Prozess durch das in `/etc/passwd` angegebene Programm, üblicherweise eine Shell. Das ebenfalls in `/etc/passwd` angegebene **Home-Verzeichnis** wird zum anfänglichen Arbeits-Verzeichnis.

Die Shell führt als erstes das Skript `/etc/profile(4)` aus, das die **Umgebung** bereitstellt und einige Mitteilungen auf den Bildschirm schreibt, News zum Beispiel. Anschließend sucht die Shell im Home-Verzeichnis nach einer Datei `.profile` (der Punkt kennzeichnet die Datei als verborgen). Dieses Skript könnte für jeden Benutzer individuell gestaltet sein. Bei uns ist es jedoch zumindest gruppenweise gleich. Wir haben in dieses Skript eine Abfrage nach einem weiteren Skript namens `.autox` eingebaut, das sich jeder Benutzer selbst schreiben kann. Wir haben also eine dreifache Stufung: `/etc/profile` für alle, auch `gast`, `$HOME/.profile` für die Gruppe und `$HOME/.autox` für das Individuum. Grafische Oberflächen bringen zum Teil weitere `.profile`-Dateien mit, die zu Beginn einer Sitzung abgearbeitet werden.

Ist dies alles erledigt, wartet die Shell auf Eingaben. Wenn sie mit `exit` beendet wird, erfährt `/etc/init` davon und erzeugt einen neuen `getty`-Prozess für das Terminal. Der `getty`-Prozess wird *respawned*.

In der Datei `/etc/inittab(4)` werden **Run Levels** definiert. Das sind Systemzustände, die festlegen, welche Terminals ansprechbar sind, d. h. einen `getty`-Prozess bekommen, und welche Dämonen laufen. Der Run Level S ist der **Single-User-Modus**, in dem nur die Konsole aktiv ist. Run Level 3 ist der übliche **Multi-User-Modus**, in dem auf unserer Anlage alle Dämonen aktiv sind. Die übrigen Run Levels legt der System-Manager fest. Die Einzelheiten sind wieder von System zu System verschieden.

Beim **System-Stop** sollen zunächst alle laufenden Prozesse ordnungsgemäß beendet und alle Puffer geleert werden. Dann soll das System in den Single-User-Modus überführt werden. Das Skript `/sbin/shutdown(1M)` erledigt diese Arbeiten automatisch. Mit der Option `-r` bootet `shutdown(1M)` wieder, ansonsten dreht man anschließend den Strom ab. Dabei gilt die Regel, dass zuerst die Zentraleinheit und dann die Peripherie ausgeschaltet werden sollen. Einschalten umgekehrt.

2.14.3 Benutzerverwaltung

Die Benutzer eines UNIX-Systems lassen sich in vier Klassen einteilen:

- Programme wie `who(1)`, die als Benutzer in `/etc/passwd(4)` eingetragen sind. Auch Dämonen verhalten sich teilweise wie Benutzer, beispielsweise der Line Printer Spooler `lp`, der Dateien besitzt,
- Benutzer mit eng begrenzten Rechten wie `gast`, `ftp` oder Benutzer, die statt der Shell gleich ein bestimmtes Anwendungsprogramm bekommen, das sie nicht verlassen können,
- Normale Benutzer wie `picalz1`, im real life stud. mach. PIZZA CALZONE, mit weitgehenden, aber nicht unbegrenzten Rechten,
- Benutzer mit Superuser-Rechten wie `root`, allwissend und allmächtig.

Formal ist der Eintrag in `/etc/passwd(4)` entscheidend. Deshalb ist diese Datei so wichtig und eine potentielle Schwachstelle der System-Sicherheit. Kritisch sind die (schwach) verschlüsselten Passwörter, die heute meist in eine nur vom System lesbare Datei `/etc/shadow` ausgelagert werden.

Zur Einrichtung eines neuen Benutzers trägt der System-Manager den Benutzernamen in die Dateien `/etc/passwd(4)`:

```
picalz1:*:172:123:P. Calzone:/mnt1/homes/picalz:/bin/sh
```

und `/etc/group(4)` ein:

```
users::123:root,wualex1,gebern1,bjalex1,picalz1
```

Wir bilden den Benutzernamen aus den ersten beiden Buchstaben des Vornamens, den ersten vier Buchstaben des Nachnamens und dann einer Ziffer, die die Accounts einer Person durchnumeriert. Dieses Vorgehen ist nicht zwingend, hat sich aber bewährt. Anschließend vergibt der System-Manger mittels `passwd(1)` ein nicht zu simples Passwort. Dann richtet er ein Home-Verzeichnis ein, setzt die Zugriffsrechte (700), übereignet es dem Benutzer samt Gruppe und linkt oder kopiert schließlich ein `.profile` in das Home-Verzeichnis, ohne das der Benutzer nicht viel machen darf. Der Benutzer hat nun ein **Konto** oder einen **Account** auf dem System. Das erste Feld in der `/etc/passwd(4)`-Zeile enthält den **Benutzernamen**. Ein Stern im Passwortfeld führt dazu, daß man sich unter dem zugehörigen Namen nicht anmelden kann, das Konto ist deaktiviert. Nur `root` kann dann das Passwort ändern. Das braucht man für Dämonen wie `lp`, die als Dateibesitzer auftreten, sowie bei Maßnahmen gegen unbotmäßige oder verschollene Benutzer. Das dritte und vierte Feld speichern die Benutzer- und Gruppennummer. Fünftens folgt das GECOS-Feld (GECOS = General Electric Comprehensive Operating System, ein historisches Relikt) mit Kommentar, der von Kommandos wie `finger(1)` ausgewertet wird. Mit dem Kommando `chfn(1)` (change finger information) kann jeder Benutzer sein eigenes GECOS-Feld ändern. Schließlich das Home-Verzeichnis und die Sitzungsshell. Letztere darf kein weicher Link sein, sonst gibts Probleme. Statt der Shell, die einen Universal-Zugang vermittelt, kann für Benutzer, die nur mit einem einzigen Programm arbeiten, eine eingeschränkte Shell oder dieses Programm eingetragen werden. Das erhöht etwas die Sicherheit und dämmt Missbrauch ein. Der Eintrag in `/etc/group(4)` listet nach Gruppennamen und -nummer die zugehörigen Benutzer auf, durch Komma ohne Leerzeichen getrennt. Ein Benutzer kann mehreren Gruppen angehören. Die Anzahl der Benutzer pro Gruppe ist begrenzt. Die Grenze ist systemabhängig und liegt bei unseren Maschinen teilweise schon bei etwas über 80 Benutzern, daher keine Riesengruppen planen. Eine zu große Gruppe in `/etc/group(4)` führt dazu, dass die Datei von dieser Gruppe an nicht mehr gelesen wird. Die Dateien `/etc/passwd(4)` und `/etc/group(4)` werden vom Anfang her gelesen und beim ersten Treffer verlassen. Falls man einen Benutzer versehentlich doppelt eingetragen hat, wirkt sich nur der vordere Eintrag aus.

Auch UNIX-Kommandos wie `who(1)` oder `date(1)` lassen sich als Benutzer eintragen. Der folgende Eintrag in `/etc/passwd(4)`:

```
who::90:1:Kommando who:/:/usr/local/bin/who
```

samt dem zugehörigen Eintrag in `/etc/group(4)` ermöglicht es, sich durch Eingabe von `who` als login-Name ohne Passwort eine Übersicht über die augenblicklich angemeldeten Benutzer zu verschaffen. Das `who` aus `/usr/local/bin` ist eine Variante des ursprünglichen `/bin/who(1)` mit einer verlängerten Dauer der Anzeige:

```
/bin/who; sleep 8
```

Solche Kommandos als Benutzer haben keine Sitzungsumgebung, können also nicht auf Umgebungsvariable zugreifen. Sie gelten wegen des fehlenden Passwortes als Sicherheitslücke. Falls man sie dennoch einrichtet, soll man darauf achten, dass der aufrufende Benutzer keine Möglichkeit hat, das Kommando zu verlassen – beispielsweise eine Shell aus dem Kommando heraus zu starten (Shell-Escape) – oder abzubrechen.

Ein **Rollen-Account** gehört zunächst einmal nicht zu einer natürlichen Person, sondern zu einer Aufgabe im System, beispielsweise der Pflege der WWW-Seiten (`wwwadm`) oder einer Datenbank (`dba`). Einige Rollen werden nur als Dateibesitzer gebraucht wie etwa `bin`. Erst in einem zweiten Schritt sind einigen Rollenaccounts Personen oder Personengruppen zugeordnet. Im einfachsten Fall sind das die Personen, die das zugehörige Passwort kennen. Auch der Superuser ist eine Rolle. Hingegen ist der Postmaster keine Rolle, sondern ein Email-Alias, also eine Email-Anschrift, an die Fehlermeldungen und Problemfälle gehen. Wer diese Mails bearbeitet, ist eine andere Frage. In Datenbanken ist das Rollenkonzept stark ausgeprägt.

Der Benutzer mit **Superuser**-Rechten, üblicherweise der System-Manager unter dem Namen `root` mit der User-ID 0 (null), ist auf großen oder besonders gefährdeten Anlagen eine Schwachstelle. Ist er ein Schurke, so kann er infolge seiner Allmacht im System viel anrichten. Es gibt daher Ansätze, seine Allmacht etwas aufzuteilen, indem für bestimmte Aufgaben wie das Accounting ein eigener Benutzer namens `adm` eingerichtet wird. Die Datei `/etc/passwd(4)` sollte man von Zeit zu Zeit darauf ansehen, welche Benutzer die User-ID oder Gruppen-ID 0 haben. Dieser Kreis sollte klein sein und unbedingt ein Passwort haben. Der Eintrag für den Benutzer `root` steht meist an erster Stelle. Beschädigt man ihn durch unvorsichtigen Umgang mit dem Editor, kann guter Rat teuer werden. Deshalb haben wir einen weiteren Benutzer mit der ID 0 unter einem passenden Namen mitten in der Datei angelegt, auf den man ausweichen kann, wenn der `root`-Eintrag hinüber ist. Durch Schaden wird man klug.

Auch wäre es manchmal zweckmäßig, einzelne Aufgaben wie das Einrichten von Benutzern, die Druckerverwaltung oder das Beenden von Prozessen an Unter-Manager delegieren zu können. Man denke an Netze, in denen solche Aufgaben besser vor Ort erledigt werden. Das herkömmliche UNIX kennt jedoch nur den einzigen und allmächtigen Superuser. Microsoft Windows dagegen beschäftigt eine Schar von subalternen Managern unter dem Administrator.

2.14.4 NIS-Cluster

Netze aus zusammengehörenden Computern (Cluster, Domänen, Arbeitsgruppen) sind nach zwei Prinzipien organisiert. Sie bilden entweder:

- ein Peer-to-Peer-Netz oder
- ein server-orientiertes Netz.

In einem **Peer-to-Peer-Netz** sind alle Computer gleichberechtigt, es gibt keine zentralen Dienste. Ein Computer mag einen großen Drucker haben und im Netz zur Verfügung stellen, ein anderer vielleicht einen ISDN-Anschluss. Aber jeder Computer ist für sich allein lebensfähig. Typischerweise ist ein solches Netz nicht zu ausgedehnt, alle Benutzer kennen sich von Angesicht. In einem **server-orientierten Netz** stellen wenige zentrale Server Dienste wie Benutzerverwaltung, Datei-Systeme, Backup, Email, Datenbank und dergleichen bereit. Fällt ein Server aus, stehen alle Räder still. Solche Netze können

Kontinente umspannen, die Benutzer kennen sich nur zum Teil. Und dann gibt es noch Mischformen, die die Netz-Manager vorzeitig ergrauen lassen. Mit UNIX kann man beide Typen von Netzen verwirklichen, der Schwerpunkt liegt aber bei der Server-Orientierung. Sowie man mehr als drei Computer zu vernetzen hat und auf Sicherheit und Nachtruhe Wert legt, ist die Zentralisierung der einfachere Weg.

Ein NIS-Cluster (NIS = Network Information Service) ist eine Gruppe von UNIX-Rechnern mit gemeinsamer Benutzerverwaltung, also das, was unter Microsoft Windows NT eine Domäne ist. In der Regel wird die Benutzerverwaltung ergänzt durch gemeinsame Datei-Systeme (vor allem für die Homes), die per NFS (NFS = Network File System) auf alle Cluster-Teilnehmer gemountet werden. Jeder Cluster-Benutzer kann sich dann an irgendeine Cluster-Maschine setzen und findet dort seine Umgebung und seine Daten vor. Sowohl NIS wie NFS stammen von Sun. Die NIS-Dienste hießen anfangs Yellow Pages, dieser Name musste jedoch aus juristischen Gründen aufgegeben werden. Die zugehörigen Kommandos beginnen aber immer noch mit `yp`. Im folgenden geht es nicht um eine detaillierte Anleitung – dafür siehe die man-Seiten, vor allem zu `ypfiles(4)` oder `ypserv(8)` – sondern um das Verständnis.

In einem NIS-Cluster gibt es genau einen Master-Server und optional einige Slave-Server, in Windows-Speak einen Primary Domain Controller und einige Secondary oder Backup Domain Controller. Die restlichen Maschinen sind NIS-Clients. Auf dem Master-Server liegen im Verzeichnis `/var/yp/src` einige Dateien, die man sonst in `/etc` findet:

- passwd
- group
- hosts
- aliases
- protocols
- services

Diese Dateien werden mittels eines editierbaren Makefiles auf die Domänen-Mitglieder verteilt, nach jeder Änderung und sicherheitshalber auch noch periodisch per `cron(1)`.

Auf jedem Domänen-Mitglied findet sich eine Datei `/etc/nsswitch.conf` – am besten mittels `man -k switch` nach der Beschreibung suchen. Diese Datei enthält Zeilen folgender Art:

```
passwd:      files nis
hosts:       dns files nis
```

In obigem Beispiel soll ein Benutzer zuerst in der zuständigen lokalen Datei `/etc/passwd` gesucht werden, anschließend in der NIS-Datei `/var/yp/domaene/passwd`. Die lokale Datei hat also Vorrang. Dort finden sich außer `root` die lokalen Dämonen und wenige, rein lokale Benutzer. Hostnamen sollen zuerst über den Domain Name Service (DNS) aufgelöst werden, an zweiter Stelle mit Hilfe der lokalen Datei `/etc/host` und an dritter Stelle mit Hilfe der NIS-Datei `/var/yp/domaene/hosts`. Die NIS-Dateien in dem Verzeichnis mit dem Namen der Domäne sind ein bisschen aufbereitet, was aber für das Verständnis unerheblich ist. Damit haben wir eine zentrale Verwaltung der Benutzer und einiger weiterer Informationen.

yppasswd

NFS

ftpd?

ssh

Der Aufbau eines NIS-Clusters ist einfach und lohnt sich schon in kleinen UNIX-Netzen. Bei Linux-Distributionen sind die erforderlichen Programme dabei.

2.14.5 Geräteverwaltung

2.14.5.1 Gerätedateien

Alle Peripheriegeräte (Platten, Terminals, Drucker) werden von UNIX als Datei behandelt und erscheinen im Verzeichnis `/dev`. Dieses Verzeichnis hat einen besonderen Aufbau. Schauen Sie sich es einmal mit `ls -l /dev | more` an. Das Kommando zum Eintragen neuer Geräte lautet `/etc/mknod(1M)` oder `mksf(1M)` und erwartet als Argument Informationen über den Treiber und den Port (Steckdose) des Gerätes. Die Namen der Geräte sind der besseren Übersicht wegen standardisiert. `/dev/tty` ist beispielsweise das Kontroll-Terminal, `/dev/null` der Bit Bucket oder Papierkorb. Die ganze Sektion 7 des Referenz-Handbuches ist den Gerätefiles gewidmet.

2.14.5.2 Terminals

Moderne Bildschirm-Terminals sind anpassungsfähig. Das hat andererseits den Nachteil, dass man sie an den Computer anpassen muss. Im einfachsten und unter UNIX häufigsten Fall ist das Terminal durch eine Leitung mit minimal drei Adern an einen seriellen Ausgang (Port) des Computers angeschlossen. Die Daten werden über diese Leitung mit einer Geschwindigkeit von 9600 bit/s übertragen, das sind rund tausend Zeichen pro Sekunde. Für Text reicht das, für größere Grafiken kaum. Diese Gattung von Terminals wird als **seriell** nach ASCII-, ANSI- oder sonst einer Norm bezeichnet.

Bei PCs ebenso wie bei Workstations schreibt der Computer mit hoher Geschwindigkeit in den Bildschirmspeicher. Zu jedem Bildpunkt gehört ein Speicherplatz von ein bis vier Byte. Der Speicherinhalt wird 50- bis 100-mal pro Sekunde zum Bildschirm übertragen. Diese Gattung wird als **bitmapped** bezeichnet und ist grafikfähig. Die Leitung zwischen Computer und Terminal muss kurz sein.

Die Konfiguration erfolgt teils durch kleine Schalter (DIP-Schalter, Mäuseklaviere), teils durch Tastatureingaben und teils durch Programme vom Computer aus. Da jeder Terminaltyp in den Einzelheiten anders ist, kommt man um das Studium des zugehörigen Handbuchs nicht herum. Dieses wiegt bei dem Terminal Hewlett-Packard 2393A, das wir im folgenden vor Augen haben, runde fünf Pfund.

Das HP 2393A ist ein serielles, monochromes, grafikfähiges Terminal, das HP-Befehle versteht, aber auch auf ANSI-Befehle konfiguriert werden kann. Einige Einstellungen sind:

- Block Mode off (zeichenweise Übertragung ein)
- Remote Mode on (Local Mode off, Verbindung zum Computer ein)
- Display Functions off (keine Anzeige, sondern Ausführung von Steuerzeichen)
- Display off after 5 min (Abschalten des Bildschirms bei Ruhe)
- Language English (Statusmeldungen des Terminals)
- Term Mode HP (nicht ANSI oder VT 52)
- Columns 80 (Anzahl der Spalten im Textmodus)
- Cursor Type Line (Aussehen des Cursors)
- Graphical Resolution 512 x 390 (Punkte horizontal und vertikal)
- Baud Rate 9600 (Übertragungsgeschwindigkeit)
- Parity/Data Bits None/8 (Zeichenformat)
- Check Parity No (Paritätsprüfung)
- Stop Bits 1 (Zeichenformat)
- EnqAck Yes (Handshake)
- Local Echo Off (keine Anzeige der Eingaben durch das Terminal)

Während manche Einstellungen harmlos sind (Cursor Type), sind andere für das Funktionieren der Verbindung zum Computer lebenswichtig (Remote Mode, Baud Rate). Da viele Werte auch computerseitig eingestellt werden können, sind Missverständnissen keine Grenzen gesetzt. Der Benutzer soll die Einstellungen nicht verändern und sich bei Problemen auf das Betätigen der Reset-Taste beschränken. Der System-Manager schreibe sich die Konfiguration sorgfältig auf.

Bei der Einrichtung eines **Terminals** ist darauf zu achten, dass eine zutreffende `terminfo(4)`-Eintragung verfügbar ist. Bei neueren Terminals ist das leider eine Ausnahme, sodass der System-Manager die Terminalbeschreibung für das `terminfo`-Verzeichnis selbst in die Hände nehmen muss, was beim ersten Versuch mit Nachdenken verbunden ist.

Ein UNIX-System arbeitet mit den unterschiedlichsten Terminals zusammen. Zu diesem Zweck ist eine Beschreibung einer Vielzahl von Terminaltypen in dem Verzeichnis `/usr/lib/terminfo(4)` gespeichert (oder in `/etc/termcap`), und zwar in einer kompilierten Form. Die `curses(3)`-Funktionen zur Bildschirmsteuerung greifen darauf zurück und damit auch alle Programme, die von diesen Funktionen Gebrauch machen wie der Editor `vi(1)`.

Der Compiler heißt `tic(1M)`, der Decompiler `untic(1M)`. Um sich die Beschreibung eines Terminals auf den Bildschirm zu holen, gibt man `untic(1M)` mit dem Namen des Terminals ein, so wie er in der `terminfo` steht:

```
untic vt100
```

Die Ausgabe sieht so aus:

```
vt100|vt100-am|dec vt100,
    am, xenl,
    cols#80, it#8, lines#24, vt#3,
    bel=^G, cr=\r, csr=\E[%i%p1%d;%p2%dr, tbc=\E[3g,
    clear=\E[H\E[2J, el=\E[K, ed=\E[J,
    cup=\E[%i%p1%d;%p2%dH,
    cud1=\n, home=\E[H, cub1=\b, cuf1=\E[C,
    cuu1=\E[A, blink=\E[5m, bold=\E[1m, rev=\E[7m,
    smso=\E[7m, smul=\E[4m, sgr0=\E[m, rmso=\E[m,
    rmul=\E[m, kbs=\b, kcud1=\EOB, kcub1=\EOD,
    kcuf1=\EOC, kcuu1=\EOA, rmkx=\E[?1l\E>, smkx=\E[?1h\E=,
    cud=\E[%p1%dB, cub=\E[%p1%dD,
    cuf=\E[%p1%dC, cuu=\E[%p1%dA,
    rs2=\E>\E[?3l\E[?4l\E[?5l\E[?7h\E[?8h,
    rc=\E8, sc=\E7, ind=\n, ri=\EM,
    sgr=\E[%?%p1%t;7%;%?%p2%t;4%;%?%p3%t;7%;%?%p4%t;
                             5%;%?%p6%t;1%;m,
    hts=\EH, ht=\t,
```

Die erste Zeile enthält den gängigen Namen des Terminaltyps, dahinter durch den senkrechten Strich abgetrennt weitere Namen (Aliases), als letzten die vollständige Typbezeichnung. Die weiteren Zeilen geben die Eigenschaften (capabilities) des Typs an, eingeteilt in drei Klassen

- Boolesche Variable, das sind Eigenschaften, die entweder vorhanden sind oder nicht,
- Zahlenwerte wie die Anzahl der Zeilen und Spalten,
- Strings, das sind vielfach Steuersequenzen (Escapes).

Die Bedeutung der einzelnen Abkürzungen entnimmt man `terminfo(4)`, hier nur einige Beispiele:

- `am` Terminal has automatic margins (soll heißen: wenn man über den rechten Rand hinaus schreibt, wechselt es automatisch in die nächste Zeile),
- `xenl` Newline ignored after 80 columns (wenn man nach 80 Zeichen ein newline eintippt, wird es ignoriert, weil automatisch eines eingefügt wird, siehe oben),
- `cols#80` Number of columns in a line (80 Spalten),

- `it#8` Tabs initially every 8 spaces (Tabulatoren),
- `lines#24` Number of lines on screen or page (24 Zeilen),
- `vt#3` Virtual terminal number,
- `bel=^G` Audible signal (die Zeichenfolge, welche die Glocke erschallen lässt, `control-g`, ASCII-Zeichen Nr. 7),
- `tbc=\E[3g` Clear all tab stops (die Zeichenfolge, die alle Tabulatoren löscht, ESCAPE, linke eckige Klammer, 3, g),
- `clear=\E[H\E[2J` Clear screen and home cursor (die Zeichenfolge, die den Bildschirm putzt und den Cursor in die linke obere Ecke bringt, ESCAPE, linke eckige Klammer, H, nochmal ESCAPE, linke eckige Klammer, 2, J),
- `kcud1=\E0B` Sent by terminal down arrow key (die Zeichenfolge, die die Cursortaste Pfeil nach unten abschickt, muss nicht notwendig mit der Zeichenfolge übereinstimmen, die den Cursor zu der entsprechenden Bewegung veranlasst),
- `sgr=\E[%?....` Define the video attributes,
- `cup=\E[%i%p1%d;%p2%dH` Screen relative cursor motion row #1 column #2 (Cursorpositionierung nach Bildschirmkoordinaten)

In `termio(4)` findet man rund 200 solcher Eigenschaften erläutert; Farbe, Grafik und Maus fehlen. Der Zusammenhang zwischen den knappen Erklärungen im Referenz-Handbuch und der Beschreibung im Terminal-Handbuch ist manchmal dunkel und bedarf der Klärung durch das Experiment. Man geht am besten von der Beschreibung eines ähnlichen Terminals aus, streicht alles, was man nicht versteht und nimmt Schritt um Schritt eine Eigenschaft hinzu. Eine falsche Beschreibung macht mehr Ärger als eine unvollständige. Wenn die Kommandos `vi(1)` und `more(1)` oder `pg(1)` richtig arbeiten, stimmt wahrscheinlich auch die Terminalbeschreibung.

Die mit einem Editor verfasste Beschreibung wird mit `tic(1M)` kompiliert, anschließend werden die Zugriffsrechte in der `terminfo` auf 644 gesetzt, damit die Menschheit auch etwas davon hat.

2.14.5.3 Platten, Datei-Systeme

Einrichten einer Platte Eine fabrikneue Platte bedarf einiger Vorbereitungen, ehe sie zum Schreiben und Lesen von Daten geeignet ist. Bei älteren Platten für PCs bestand die Vorbereitung aus drei Stufen, bei neueren Platten kann der Hersteller den ersten Schritt bereits erledigt haben. Falls eine Platte bereits Daten speichert und neu eingerichtet wird, sind die Daten verloren.

Ein Plattenlaufwerk (disc drive) enthält mindestens eine, höchstens 14 Scheiben (disc) mit jeweils zwei Oberflächen. Der erste Schritt – Low-Level-Formatierung genannt – legt auf den fabrikneuen, magnetisch homogenen Scheibenoberflächen konzentrische Spuren (track) an und unterteilt diese in

Sektoren (sector). Fehlerhafte Sektoren werden ausgeblendet. Räumlich übereinanderliegende Spuren auf den Oberflächen der Scheiben (gleicher Radius) bilden einen Zylinder (cylinder). Dieser Schritt wird heute oft schon vom Hersteller getan. Die Numerierung der Spuren und Sektoren kann den tatsächlichen Verhältnissen auf den Scheiben entsprechen, aber auch durch eine Tabelle in eine logische, beliebig wählbare Numerierung umgesetzt werden. Die gesamte Anzahl der Sektoren wird durch die Umsetzung nicht größer, sonst hätte man ein preiswertes Mittel zur Erhöhung der Plattenkapazität.

SCSI-Platten werden mit einem Werkzeug formatiert, das zum Adapter gehört und vor dem Bootvorgang aufgerufen wird. Die Formatierung ist spezifisch für den Adapterhersteller. Dabei lassen sich weitere Parameter einstellen und die Platten prüfen.

Im zweiten Schritt wird die Platte in Partitionen, bestehend aus zusammenhängenden Bereichen von Zylindern, unterteilt und in jeder Partition ein Datei-System angelegt, unter UNIX mittels `mkfs(1M)` oder `newfs(1M)`, unter PC-DOS mittels `fdisk` und `format`. Das Datei-System ist vom Betriebssystem abhängig. Manche Betriebssysteme kommen mit verschiedenen Datei-Systemen zurecht, insbesondere ist Linux zu loben. Auf einem Plattenlaufwerk können mehrere Partitionen angelegt werden, eine Partition kann sich in der Regel nicht über mehrere Plattenlaufwerke erstrecken, es gibt aber Ausnahmen (spanning, RAID).

Da beim Einrichten lebenswichtige Daten auf die Platte geschrieben werden, soll sie dabei ihre Betriebslage und annähernd ihre Betriebstemperatur haben. Die Elektronik moderner Platten gleicht zwar vieles aus, aber sicher ist sicher.

Organisation Auf Platten werden große Datenmengen gespeichert, auf die oft zugegriffen wird. Eine gute Datenorganisation trägt wesentlich zur Leistung des gesamten Computers bei.

Bei der Erzeugung einer Datei ist die endgültige Größe oft unbekannt. Der Computer reserviert eine ausreichende Anzahl von Blöcken beispielsweise zu je 512 Byte. Im Lauf der Zeit wächst oder schrumpft die Datei. Der Rechner muss dann irgendwo im Datei-System weitere, freie Blöcke mit den Fortsetzungen der Datei belegen bzw. gibt Blöcke frei. Die Dateilandschaft wird zu einem Flickenteppich, die Dateien sind **fragmentiert**. Von der Software her macht die Verwaltung dieser Flicken keine Schwierigkeiten, der Benutzer merkt nichts davon. Aber die Schreibleseköpfe des Plattenlaufwerks müssen hin und her springen, um die Daten einer Datei zusammenzulesen. Das kostet Zeit und erhöht den Verschleiß. Man wird also von Zeit zu Zeit versuchen, zusammengehörende Daten wieder auf zusammengehörenden Plattenbereichen zu vereinen. Ein immer gangbarer Weg ist, die Dateien von einem Band (wo sie zusammenhängend gespeichert sind) auf die Platte zu kopieren. Während dieser Zeit ist kein Rechenbetrieb möglich. Für manche Betriebssysteme gibt es daher Dienstprogramme, die diese Arbeit während des Betriebes oder wenigstens ohne den Umweg über ein Band erledigen. In UNIX-Datei-Systemen werden die Auswirkungen der Fragmentierung durch Pufferung abgefangen, der Verwalter braucht sich um die Fragmentierung nicht zu sorgen.

Mounten eines Datei-Systems Beim Systemstart aktiviert UNIX sein root-Datei-System. In dieses müssen weitere Datei-Systeme – gleich ob von Platte, Diskette oder Band – eingehängt werden. UNIX arbeitet immer nur mit *einem* Datei-System. Das Einhängen wird nach englischem Muster als *mounten* bezeichnet. Das kann per Shellskript beim Systemstart erfolgen oder nachträglich von Hand mittels des Kommandos `mount(1M)`. Die beim Systemstart zu mountenden Datei-Systeme sind in `/etc/fstab(4)` aufgelistet. Mittels des Kommandos `mount(1M)` erfährt man Näheres über die jeweils gemounteten Datei-Systeme.

Zum Mounten braucht man Mounting Points im root-Datei-System. Das sind leere Verzeichnisse, auf die die root-Verzeichnisse der zu mountenden Datei-Systeme abgebildet werden. Mounten über das Netz funktioniert gut, es sind jedoch einige Überlegungen zur Sicherheit anzustellen.

Pflege des Datei-Systems Das **Datei-System** kann durch Stromausfälle und ähnliche Unregelmäßigkeiten fehlerhaft (korrupt) werden und wird mit Sicherheit nach einiger Zeit überflüssige Daten enthalten. Nahezu volle Datei-Systeme geben leicht Anlass zu Störungen, die nicht immer auf den ersten Blick ihre Ursache erkennen lassen. Ab 90 % Füllstand wird es kritisch, weil manche Programme Platz für temporäre Daten benötigen und hängenbleiben, wenn er fehlt. Deshalb ist eine Pflege erforderlich.

Den Füllstand der Datei-Systeme ermittelt man mit dem Kommando `df(1M)` oder `bdf(1M)`. Zum Erkennen und Beseitigen von Fehlern dient das Kommando `/etc/fsck(1M)`. Ohne Optionen oder Argumente aufgerufen überprüft es die in der Datei `/etc/checklist(4)` aufgelisteten Datei-Systeme und erfragt bei Fehlern die Rettungsmaßnahmen. Da dem durchschnittlichen System-Manager kaum etwas anderes übrig bleibt als die Vorschläge von `fsck(1M)` anzunehmen, kann man die zustimmende Antwort auch gleich als Option mitgeben und `fsck -y` eintippen. Eine Reparatur von Hand kann zwar im Prinzip mehr Daten retten als die Reparatur durch `fsck(1M)`, setzt aber eine gründliche Kenntnis des Datei-Systems und der Plattenorganisation voraus. Meistens vergrößert man den Schaden noch. Bei der Anwendung von `fsck(1M)` soll das System in Ruhe, das heißt im Single User Modus sein. In der Regel führt man die Prüfung vor einem größeren Backup und beim Booten durch.

Das Aufspüren überflüssiger Dateien erfordert eine regelmäßige, wenigstens wöchentliche Beobachtung, wobei der `cron(1M)` hilft. Dateien mit Namen wie `core(4)` oder `a.out(4)` werden üblicherweise nicht für eine längere Zeit benötigt und sollten automatisch gelöscht werden (als `root` von `cron` aufrufen lassen):

```
find /homes \( -name core -o -name a.out \)
                                  -exec /usr/bin/rm {} \;
```

Will man Dateien oder Verzeichnisse, die ihre Besitzer entgegen den Sicherheitsrichtlinien world-writable angelegt haben, automatisch auf die Zugriffsrechte 700 setzen, so erledigt das ein Shellskript mit folgenden Zeilen:

```
find /mnt \( -perm -0007 -o -perm -0003 -o -perm -0002 \) -fsonly hfs -type
find /mnt \( -perm -0007 -o -perm -0003 -o -perm -0002 \) -fsonly hfs -type
```

Von dem Verzeichnis `/mnt` an abwärts werden Verzeichnisse (`-type d`) oder Dateien (`-type f`) mit den Zugriffsrechten 0007, 0003 oder 0002 gesucht, und zwar nur in hfs-Datei-Systemen (`fsonly hfs`). Bei den Zugrifsrechten bedeutet 0 beliebig, es zählen nur die Bits für den Rest der Welt. Wird eine Datei oder ein Verzeichnis entdeckt, so werden sein Pfad in die geschweiften Klammern eingefügt und das Kommando `chmod 700 ;` ausgeführt, das heißt die Zugriffsrechte geändert. Da die runden Klammern und das Semikolon für die Shell eine Bedeutung haben, sind sie zu quoten.

Dateien, auf die seit einem Monat nicht zugegriffen wurde, gehören nicht auf die kostbare Platte; mit:

```
find /homes -atime +32 -print
```

aufspüren und die Benutzer bitten, sich ein anderes Medium zu suchen.

Dateien oder Verzeichnisse, die von aller Welt beschrieben werden dürfen, sind verdächtig. Meist sind diese laschen Zugriffsrechte aus Versehen oder Bequemlichkeit gesetzt. Mit den Kommandos:

```
find /homes \( -perm -0007 -o -perm -0003 -o -perm -0002 \) -type f -print
find /homes \( -perm -0007 -o -perm -0003 -o -perm -0002 \) -type d -print
find /homes \( -perm -0007 -o -perm -0003 -o -perm -0002 \) -type f -exec /u
find /homes \( -perm -0007 -o -perm -0003 -o -perm -0002 \) -type d -exec /u
```

kann man sie sich auflisten lassen oder gleich auf harmlosere Rechte setzen lassen. Die Optionen `-type f` oder `-type d` beschränken die Suche auf Dateien oder Verzeichnisse (directories).

Es kommt auch vor, dass Benutzer verschwinden, ohne sich beim System-Manager abzumelden. Dies lässt sich mittels `last(1)` oder des Accounting Systems feststellen.

Schließlich sollte man die Größe aller Dateien und Verzeichnisse überwachen. Einige Protokolldateien des Systems wachsen unbegrenzt und müssen von Zeit zu Zeit von Hand bereinigt werden. Auch sind sich manche Benutzer der Knappheit des Massenspeichers nicht bewußt. Mit

```
find -size +n -print
```

lassen sich alle Dateien ermitteln, deren Größe über n Blöcken liegt, mit folgendem Skript die Größe aller Home-Verzeichnisse, sortiert nach der Anzahl der Blöcke:

```
# Script Uebersicht Home-Directories

print 'Home-Directories, Groesse in Bloecken\n'
```

```
{
cd /mnt
for dir in `ls .`
do
du -s $dir
done
} | sort -nr

print '\nEnde, ' `date`
```

Quelle 2.44 : Shellskript zur Ermittlung der Größe aller Home-Verzeichnisse

Mittels `du(1)` kann man auch in dem Skript `/etc/profile`, das für jeden Benutzer beim Anmelden aufgerufen wird, eine Ermittlung und Begrenzung der Größe des Home-Verzeichnisses erzielen. Auf neueren Systemen findet man auch einen fertigen Quoten-Mechanismus, siehe `quota(1)` und `quota(5)`.

2.14.5.4 Drucker

Vorneweg: Drucken in einem heterogenen Netz ist das Härteste. Und die Benutzer ahnen nichts von den Schwierigkeiten (*Bei mir zu Hause funktioniert das doch prima* ...). Neben den lokalen man-Seiten ist das Linux-Drucker-HOWTO (deutsch oder englisch) ein guter Einstieg. Drucker können auf zwei Arten angeschlossen sein:

- an die serielle oder parallele Schnittstelle eines Computers oder Printer-Servers,
- mit einem eigenen Adapter (Ethernet-Karte) unmittelbar ans Netz.

Ein Printer-Server ist ein kleiner, spezialisierter Computer, auf dem nur das Druckprogramm läuft. Man kann auch einen ausgedienten PC dafür nehmen. Im ersten Fall ist zwischen lokalen Druckern und Druckern an einem fernen Computer oder Printer-Server im Netz zu unterscheiden. Im zweiten Fall stellen die Drucker einen Host im Netz mit eigener IP-Adresse und eigenem Namen dar. Die erste Lösung hat den Vorteil, dass man auf dem zugehörigen Computer beliebig in den Datenstrom zum Drucker eingreifen kann (bei Printer-Servern eingeschränkt), die zweite den Vorteil der höheren Übertragungsgeschwindigkeit, im Ethernet 10 oder 100 Mbit/s. Mit Netzdruckern kann man sich per Telnet unterhalten, sie sind vergleichsweise intelligent.

Beginnen wir mit dem an eine Schnittstelle des eigenen Computers angeschlossenen lokalen Drucker. Er weiß zwangsläufig nichts vom Netz und braucht ein Gerätefile `/dev/lp` oder ähnlich. Bei der Einrichtung des Gerätefiles sind Portadresse, Interrupt, Geschwindigkeit usw. zu konfigurieren; das muss man unter Kommandos wie `mknod(1m)`, `stty(1)` oder `insmod(1m)` nachlesen. Läuft die Schnittstelle einwandfrei, sollte der System-Manager mit

```
cat textfile > /dev/lp
```

ein kurzes, einfaches Textfile ausdrucken können. So darf es jedoch nicht bleiben, da sich verschiedene Druckaufträge in die Quere kommen würden. Wir brauchen einen Spooler, der die Druckaufträge in eine Warteschlange einreiht.

Außer firmenspezifischen Spoolsystemen gibt es drei verbreitete **Spooler** in der UNIX-Welt:

- das BSD-System, verwendet von Sun, DEC und Linux, Druckkommando `lpr(1)`, Spooler `lpd(1M)`,
- das System-V-System, verwendet von Sun, Siemens, HP und SGI, Druckkommando `lp(1)`, Spooler `lpsched(1m)`,
- das jüngere LPRng-System, verwendet von einigen Linux-Distributionen, Druckkommando `lpr(1)`, Spooler `lpd(1m)`.

Die drei Spooler unterscheiden sich in ihrer Struktur, ihren Kommandos und ihrem Funktionsumfang. Alle drei erlauben es, über das Netz Drucker an fremden Hosts anzusprechen.

Laserdrucker, die selbst im Netz sind und daher eine eigene IP-Adresse haben, können über das **HP Distributed Print System** HPDPS verwaltet werden. Dieses setzt Grundfunktionen des Distributed Computing Environment DCE voraus und erübrigt ein Spoolsystem obiger Art. Das System trägt auch den Namen *Palladium* und ist im Verzeichnis `/opt/pd` angesiedelt, Druckkommando `pdpr(1)`.

Die Drucksysteme bestehen aus Hard- und Software mit folgendem Aufbau (beginnend mit der Hardware):

- Physikalische (in Hardware vorhandene) Drucker,
- Filter (Programme, Skripts), durch die die Druckaufträge laufen,
- Warteschlangen für Druckaufträge,
- logische (für den Benutzer scheinbar vorhandene) Drucker,
- Software zur Administration (lpd, Spooler, Supervisor usw.), gegebenenfalls getrennt für Hard- und Software.

Zu einer Warteschlange gehört ein physikalischer Drucker oder eine Gruppe von solchen. Umgekehrt dürfen niemals zwei Warteschlangen gleichzeitig über einen physikalischen Drucker herfallen, das gibt ein Durcheinander. Einer Warteschlange können mehrere logische Drucker zugeordnet sein, mit unterschiedlichen logischen Eigenschaften wie Zugriffsrechte oder Prioritäten. Der Zweck dieses Rangierbahnhofs ist die Erhöhung der Flexiblität, sein Preis, dass man bald einen in Vollzeit beschäftigten Drucker-Verwalter braucht.

Die verschiedenen Einstellungen eines Druckers (Zeichensatz, Verhalten beim Zeilenwechsel, Schnittstelle usw.) werden entweder durch kleine Schalter (Mäuseklaviere) oder durch Tasteneingaben am Drucker vorgenommen. Zusätzlich können sie vom Computer aus per Software verändert werden. Die Einstellungsmöglichkeiten hängen vom Druckertyp ab, hier nur die wichtigsten:

- Zeichensatz (US, German ...)
- Befehlssatz (ESC/P, IBM, PCL)
- Seitenlänge (11 Zoll, 12 Zoll)
- Umsetzung von Carriage Return und Line Feed
- Sprung über die Seitenperforation
- Zeichen/Zoll, Linien/Zoll
- Druckqualität (Draft = Entwurf, NLQ, LQ)
- Puffergröße, falls Puffer eingebaut
- Auswahl der Schnittstelle (parallel, seriell, Netz)
- Parameter einer etwaigen seriellen Schnittstelle
- gegebenenfalls Netzprotokoll (TCP/IP, Appletalk ...)
- gegebenenfalls Druckersprache (PCL, Epson, PostScript ...)

Da einige Werte auch im Betriebssystem oder im Druckprogramm eingestellt oder sogar ins Dokument geschrieben werden, hilft nur, sich sorgfältig zu merken, was man wo festlegt. Am besten geht man vom Dokument zum Drucker und ändert jeweils immer nur eine Einstellung. Die Anzahl der Kombinationen strebt gegen unendlich.

2.14.6 Einrichten von Dämonen

Dämonen sind Prozesse, die im System ständig laufen oder periodisch aufgerufen werden und nicht an ein Kontrollterminal gebunden sind. In der Liste der Prozesse erscheint daher bei der Angabe des Terminals ein Fragezeichen. Die meisten werden beim Systemstart ins Leben gerufen und haben infolgedessen niedrige Prozess-IDs.

Einige Dämonen werden von der Bootprozedur gestartet, einige von `init(1M)` aufgrund von Eintragungen in der `inittab(4)` und einige durch einen Aufruf im Shellskript `/etc/rc` samt Unterskripts oder in `.profile`. Die Shellskripts kann der System-Manager editieren und so über die Bevölkerung seines Systems mit Dämonen entscheiden. Der Anschluss ans Netz bringt eine größere Anzahl von Dämonen mit sich.

In unserem System walten nach der Auskunft von `ps -e` folgende Dämonen, geordnet nach ihrer PID:

- `swapper` mit der PID 0 (keine Vorfahren) besorgt den Datenverkehr zwischen Arbeitsspeicher und Platte.
- `init(1M)` mit der PID 1 ist der Urahne aller Benutzer-Prozesse und arbeitet die `inittab(4)` ab.
- `pagedaemon` beobachtet den Pegel im Arbeitsspeicher und lagert bei Überschwemmungsgefahr Prozesse auf die Platte aus.
- `statdaemon` gehört zu den ersten Dämonen auf dem System, weshalb wir vermuten, dass er etwas mit dem Datei-System zu tun hat.

- syncer(1M) ruft periodisch – in der Regel alle 30 Sekunden – den Systemaufruf sync(2) auf und bringt das Datei-System auf den neuesten Stand.
- lpsched(1M) ist der Line Printer Spooler und verwaltet die Drucker- und Plotter-Warteschlangen.
- rlbdaemon(1M) gehört in die LAN/9000-Familie und wird für Remote Loopback Diagnostics mittels rlb(1M) benötigt.
- sockregd dient der Network Interprocess Communication.
- syslogd(1M) schreibt Mitteilungen des Systemkerns auf die Konsole, in bestimmte Dateien oder zu einer anderen Maschine.
- rwhod(1M) beantwortet Anfragen der Kommandos rwho(1) und ruptime(1).
- inetd(1M) ist der ARPA-Oberdämon, der an dem Tor zum Netz wacht und eine Schar von Unterdämonen wie ftpd(1M) befehligt, siehe /etc/inetd.conf und /etc/services(4).
- sendmail(1M) ist der Simple Mail Transfer Protocol Dämon – auch aus der ARPA-Familie – und Voraussetzung für Email im Netz.
- portmap(1M), nfsd(1M), biod(1M) usw. sind die Dämonen, die das Network File System betreiben, sodass Datei-Systeme über das Netz gemountet werden können (in beiden Richtungen).
- cron(1M) ist der Dämon mit der Armbanduhr, der pünktlich die Aufträge aus der crontab und die mit at(1) versehenen Programmaufrufe erledigt.
- ptydaemon stellt Pseudo-Terminals für Prozesse bereit.
- delog(1M) ist der Diagnostic Event Logger für das I/O-Subsystem.

Wenn Sie dies lesen, sind es vermutlich schon wieder ein paar mehr geworden. Die Netzdienste und das X Window System bringen ganze Scharen von Dämonen mit. Unser jüngster Zugang ist der Festplatten-Bestell-Dämon fbd(1M), der automatisch bei unserem Lieferanten eine weitere Festplatte per Email bestellt, wenn das Kommando df(1) anzeigt, dass eine Platte überzulaufen droht.

2.14.7 Überwachung, Systemprotokolle, Accounting System

In einem Netz, in dem zahlreiche Benutzer vom Funktionieren zentraler Maschinen (Server) abhängen, ist es zweckmäßig, diese halbautomatisch zu überwachen. Wir rufen auf jedem Server per cron(1M) frühmorgens ein Shellskript auf, das uns wichtige Systemdaten per Email übermittelt.

```
# Shellscript watch zur Ueberwachung von Servern
# /usr/local/bin/gestern erforderlich
```

```
# fuer HP-Maschinen, ksh:
MAILLOG=/var/adm/syslog/mail.log
FTPLOG=/var/adm/syslog/*syslog.log
ECHO="echo"
NOLF="\c"
PS="ps -e"
SORT="sort -bnr +2 -3"
AWK="awk"
DF="bdf -t hfs"
FST="fsonly hfs"
USER="/usr/sbin/pwck"
GROUP="/usr/sbin/grpck"

# fuer Linux-Maschinen, bash:
# MAILLOG=/var/log/maillog*
# FTPLOG=/var/log/xferlog*
# ECHO="echo -n"
# NOLF=""
# PS="ps -ax"
# SORT="sort -bnr +3 -4"
# AWK="gawk"
# DF="df -t ext2"
# FST="fstype ext2"
# USER="/usr/sbin/pwck -r"
# GROUP="/usr/sbin/grpck -r"

# Anzahl der legalen suid/gid-Files (anzupassen)
SUID=235

# Liste der Daemonen
LISTE="sendmail inetd sshd lpd"

echo
echo `hostname` `date`

# Mail

echo
$ECHO "Mail:           $NOLF"
fgrep   "`/usr/local/bin/gestern`" $MAILLOG | wc -l
$ECHO "Mail from:      $NOLF"
grep -F "`/usr/local/bin/gestern`" $MAILLOG | grep -F ' from=' | wc -l
$ECHO "Mail to:        $NOLF"
grep -F "`/usr/local/bin/gestern`" $MAILLOG | grep -F ' to=' | wc -l

# FTP

echo
$ECHO "FTP:            $NOLF"
fgrep "`/usr/local/bin/gestern`" $FTPLOG | fgrep ftpd | wc -l
$ECHO "AFTP:           $NOLF"
fgrep "`/usr/local/bin/gestern`" $FTPLOG | fgrep ANON | wc -l
$ECHO "AFTP-retrieve:  $NOLF"
fgrep "`/usr/local/bin/gestern`" $FTPLOG | fgrep ANON | fgrep retri | wc
```

```
$ECHO "AFTP-store (0): $NOLF"
fgrep "`/usr/local/bin/gestern`" $FTPLOG | fgrep ANON | fgrep store | wc -l

# Daemonen

echo
echo 'Daemonen:'

for DAEMON in $LISTE
do
X=`$PS | fgrep $DAEMON | fgrep -v fgrep`
if test -n "$X"
then
echo "$DAEMON ok"
else
echo "$DAEMON NICHT OK"
fi
done

# Zombies

echo
echo 'Zombies (0):'
$PS | fgrep zombie | fgrep -v fgrep

# CPU-Zeit

echo
echo CPU-Zeit:
$PS | sed 's/://g' | $SORT | head -4

# Filesysteme

echo
echo 'Filesysteme (max. 90%):'
$DF | $AWK '{printf("%-16s %4s\n", $6, $5)}'

# SUID-Files etc.

if test `date +%a` = "Mon"
then

echo
$ECHO "SUID/SGID-Files ($SUID): $NOLF"
find / \( -perm -4000 -o -perm -2000 \) -$FST -type f -print | wc -l
find /mnt2 \( -perm -4000 -o -perm -2000 \) -$FST -type f -print
echo
echo 'World-writable Files (0): '
find /mnt2 \( -perm -0007 -o -perm -0003 -o -perm -0002 \) -$FST -type f -pr
echo
echo 'World-writable Verzeichnisse (0): '
find /mnt2 \( -perm -0007 -o -perm -0003 -o -perm -0002 \) -$FST -type d -pr

fi
```

```
# Benutzer

echo
echo 'Benutzercheck: '
$USER
$GROUP

echo
echo watch beendet `date`
```

Quelle 2.45 : Shellskript zum Überwachen von Servern

Die Auswertung der Daten haben wir nicht automatisiert, ein System-Manager sollte seine Maschinen ein bisschen kennen und von Hand pflegen, insbesondere regelmäßig die Protokolldateien (`*.log`) lesen. Da das Skript sowohl auf HP-UX-Maschinen wie auf Linux-Maschinen eingesetzt wird, die sich in Kleinigkeiten unterscheiden, finden sich in den ersten Zeilen einige Variable, die entsprechend auszukommentieren sind. Zu Beginn wird der Email-Verkehr des vorherigen Tages dargestellt, dann das FTP-Aufkommen. Nach den wichtigsten Dämonen folgen Zombies, die nicht auftreten sollten, und die vier CPU-Zeit-intensivsten Prozesse. Die lokalen Datei-Systeme drohen oft überzulaufen, daher werden auch sie überwacht. Schließlich sind einige Dateien und Verzeichnisse mit bestimmten Zugriffsrechten verdächtig. Da das `find`-Kommando erheblich auf den Platten herumschrappt, wird dieser Teil des Überwachung nur montags durchgeführt. Zuletzt werden noch die Dateien mit den Benutzern und ihren Gruppen geprüft. Es sollte leicht fallen, das Skript zu ändern.

Die meisten cron-Jobs dienen administrativen Zwecken und laufen daher in der ruhigen Zeit zwischen drei und sieben Uhr morgens. Es empfiehlt sich, Zeitfenster nach einem Muster folgender Art vorzugeben, damit sich die Jobs nicht ins Gehege kommen:

- 02.00 bis 02.55 benutzereigene cron-Jobs
- 03.00 bis 03.55 Netzadministration
- 04.00 bis 04.55 Serveradministration
- 05.00 bis 05.55 Administration der Arbeitsplatzrechner
- 06.00 bis 06.55 benutzereigene cron-Jobs

Manche cron-Jobs hängen voneinander ab; obige Reihenfolge funktioniert fast immer. Die Stunden fallen in das Minimum der Belastung unseres WWW-Servers; das Minimum dürfte bei anderen Maschinen ähnlich liegen.

Das Kommando `ping(1M)`, das von jedem Benutzer aufgerufen werden kann, schickt an den Empfänger einen ICMP Echo Request, also eine Aufforderung, mit einem ICMP Echo Response zu antworten. Wie die Datenpakete aussehen, interessiert uns nicht. Kommt keine Antwort, ist etwas faul. Bei einer Störungssuche geht das erste `ping(1M)` immer an die eigene Adresse (127.0.0.1 oder localhost):

```
ping 127.0.0.1
```

Das Kommando wird mit control-c abgebrochen. Dann kann man die IP-Adresse eines nahen Knotens ausprobieren, anschließend dessen Namen usw. Auf diese Weise versucht man, die Störung zu finden.

Ein Protokoll ist im Zusammenhang dieses Abschnittes eine Datei, in welches Prozesse – oft Dämonen – Informationen über ihre Tätigkeit schreiben. Das brauchen nicht immer Fehlermeldungen zu sein, auch ganz normale Abläufe werden protokolliert.

Es gibt einen zentralen Mechanismus zum Protokollieren, den `syslogd(1M)`, aber jedem Programm steht es frei, ein eigenes Protokoll zu führen. Der `syslogd(1M)` wird beim Systemstart ziemlich früh gestartet. Seine Konfiguration steht in `/etc/syslog.conf`:

```
mail.debug /var/adm/syslog/mail.log
local0.info /var/adm/syslog/pop.log
*.info;mail,local0.none /var/adm/syslog/syslog.log
*.alert /dev/console
*.alert root
*.emerg *
```

Zum Verständnis lese man die man-Seiten zu `logger(1)`, `syslogd(1M)` und `syslog(3)`. Jede Zeile besteht aus einem Selektor und einer Aktion, getrennt durch mindestens einen TAB. Ein Selektor besteht aus einem Paar *Facility.Level*. Mehrere Selektoren dürfen durch Komma getrennt aneinandergereiht werden. Eine Facility ist eine vorgegebene Bezeichnung für ein Subsystem wie `kern`, `mail` oder `local0`. Der Stern bedeutet alle. Nach dem Punkt folgt eine vorgegebene Dringlichkeit (level) wie `info`, `alert` oder `emerg`. Zu einem Selektor gehörende Meldungen gehen gemäß der angegebenen Aktion auf die Systemkonsole, die Systemkonsole eines anderen Hosts im Netz, das Terminal ausgewählter Benutzer oder in ein Protokollfile. In obigem Beispiel gehen Meldungen des `mail`-Subsystems der Dringlichkeit `debug` in das Protokollfile `/var/adm/syslog/mail.log`, Meldungen des pop-Dämons, der die Facility `local0` verwendet, in das Protokollfile `pop.log` im selben Verzeichnis, alle Meldungen der Dringlichkeit `info` mit Ausnahme der Meldungen der Subsysteme `mail` oder `local0` in das Protokollfile `syslog.log`. Alle Meldungen der Dringlichkeit `emerg` gehen an alle möglichen Empfänger.

Benutzer veranlassen mittels einer Kommandozeile oder einer Zeile in einem Shellskript wie:

```
logger Alex hat Durst.
```

einen Eintrag in eine Protokolldatei. Das Kommando kennt Optionen. Programmierer verwenden die C-Standardfunktion `syslog(3)`. Bei Missbrauch verärgert man seinen Systemverwalter, der die Protokolldateien auswertet.

Das **Accounting System** ist die Buchhaltung des Systems, der Buchhalter ist der Benutzer `adm`, oft aber nicht notwendig derselbe Mensch wie `root`. Die zugehörigen Prozesse werden durch das Kommando `/usr/lib/acct/startup(1M)`, zu finden unter `acctsh(1M)`, in der Datei `/etc/rc` in Gang gesetzt. Das Gegenstück `/usr/lib/acct/shutacct(1M)` ist Teil des Shellskripts shutdown(1M). Das Accounting System erfüllt drei Aufgaben:

- Es liefert eine Statistik, deren Auswertung die Leistung des Systems verbessert.
- Es ermöglicht das Aufspüren von Bösewichtern, die die Anlage missbrauchen.
- Es erstellt Abrechnungen bei Anlagen, die gegen Entgelt rechnen.

Zu diesem Zweck werden bei der Beendigung eines jeden Prozesses die zugehörigen statistischen Daten wie Zeitpunkt von Start und Ende, Besitzer, CPU- und Plattennutzung usw. in eine Datei geschrieben, siehe acct(4). Das ist eine Unmenge von Daten, die man sich selten unmittelbar ansieht. Zu dem Accounting System gehören daher Werkzeuge, die diese Daten auswerten und komprimieren, siehe acct(1M). Man kann sich dann eine tägliche, wöchentliche oder monatliche Übersicht, geordnet nach verschiedenen Kriterien, ausgeben lassen. Für das Gröbste nehmen wir den cron(1M); die crontab-Datei des Benutzers adm sieht so aus:

```
30 23 * * 0-6 /usr/lib/acct/runacct 2>
                    /usr/adm/acct/nite/fd2log &
10 01 * * 0,3 /usr/lib/acct/dodisk
20  * * *   * /usr/lib/ckpacct
30 01 1 *   * /usr/lib/acct/monacct
45 23 * *   * /usr/lib/acct/acctcom -l tty2p4 | mail root
```

Die Datei wird mit dem Kommando crontab(1) kompiliert. Im einzelnen bewirken die Zeilen

- runacct(1M) ist ein Shellskript, das täglich ausgeführt wird und den Tagesverlauf in verschiedenen Dateien zusammenfasst.
- dodisk(1M), beschrieben unter acctsh(1M), ermittelt die Plattenbelegung zu den angegebenen Zeitpunkten (sonntags, mittwochs), um Ausgangsdaten für die mittlere Belegung bei der Monatsabrechnung zu haben.
- ckpacct(1M), beschrieben unter acctsh(1M), überprüft stündlich die Größe des Protokollfiles /usr/adm/pacct, in das jeder Prozess eingetragen wird, und ergreift bei Überschreiten einer Grenze geeignete Vorkehrungen.
- monacct(1M), beschrieben unter acctsh(1M), stellt die Monatsabrechnung in einer Datei /usr/adm/acct/fiscal/fiscrpt zusammen, das gelesen oder gedruckt werden kann.
- acctcom(1M) protokolliert die Prozesse des Terminals /dev/tty2p4, wohinter sich unser Modem verbirgt. Eine Sicherheitsmaßnahme.

Weiteres kann man im Referenz-Handbuch und vor allem im System Administrator's Manual nachlesen. Nicht mehr aufzeichnen, als man auch auswertet, Altpapier gibt es schon genug.

2.14.8 Weitere Pflichten

2.14.8.1 Softwarepflege

Zu den Pflichten der System-Manager gehören weiterhin die Beschaffung und Pflege der Handbücher auf Papier oder als CD-ROM, das Herausdestillieren von benutzerfreundlichen Kurzfassungen (Quick Guides, Reference Cards), das Schreiben oder Herbeischaffen (GNU!) von Werkzeugen in `/usr/local/bin` oder `/opt`, Konvertierungen aller denkbaren Datenformate ineinander, das Beobachten der technischen Entwicklung und des Marktes, der Kontakt zum Hersteller der Anlage, das Einrichten von Software wie Webservern, Datenbanken oder LaTeX usw. sowie das Beraten, Ermahnen und Trösten der Benutzer.

2.14.8.2 Konfiguration des Systems

Ein System wird mit einer durchschnittlichen Konfiguration in Betrieb genommen. Es kann sich im Lauf der Zeit herausstellen, dass die Aufgaben grundsätzlich oder zu gewissen Zeiten mit einer angepassten Konfiguration – unter Umständen nach einer Ergänzung der Hard- oder Software – schneller zu bewältigen sind. Beispielsweise überwiegt tags der Dialog mit vielen kurzen Prozessoranforderungen, nachts der Batch-Betrieb mit rechenintensiven Prozessen. Die Auslastung der Maschine zeigt das Werkzeug `top(1)` (CPU, Speicher, Prozesse) oder ein eigenes Programm unter Verwendung des Systemaufrufs `pstat(2)` an.

Im Lauf der Jahrzehnte habe ich einige Programme oder Systeme konfiguriert. Am liebsten sind mir Vorlagen (Template) in Form von Textfiles mit viel Kommentar, die mit einem Editor bearbeitet werden. Grafische Front-Ends sind mir immer verdächtig, und wehe, sie funktionieren nicht richtig. Es ist auch nicht hilfreich, wenn man – wie in der Linux-Welt häufig – für diesselbe Aufgabe verschiedene grafische Front-Ends zur Auswahl hat. Konfigurationsaufgaben, an denen mehrere voneinander abhängige Dateien beteiligt sind, lassen sich allerdings ohne Front-End kaum noch bewältigen, und wenn es ein Shellskript ist.

2.14.8.3 Störungen und Fehler

Alles Leben ist Problemlösen, schrieb 1991 der aus Wien stammende Philosoph Sir Karl Raimund Popper. Durch die Computer und deren Vernetzung sind die Probleme in diesem Leben nicht weniger geworden. Das Beheben von Störungen und Beseitigen von Fehlern ist das tägliche Brot eines System-Managers. Ein so komplexes Gebilde wie ein heterogenes Netz, das ständig im Wandel begriffen ist, erfordert (noch) die dauernde Betreuung durch hochintelligente Lebensformen, die mit unvorhergesehenen Situationen fertig werden. Wir können nur einige allgemeine Hinweise geben, die für Hard- und Software gleichermaßen gelten, zum Teil auch für Autopannen:

- Die beobachteten Fehler aufschreiben und dabei *keine* Annahmen, Vermutungen oder Hypothesen über Ursachen oder Zusammenhänge treffen.

- Dann per Logik zunächst die Menge der möglichen Ursachen einengen (Hypothesen aufstellen), anschließend die Fehler untersuchen, die mit wenig Aufwand getestet werden können, schließlich die Fehler abchecken, die häufig vorkommen. Dieses Vorgehen minimiert im Mittel den Aufwand, kann im Einzelfall jedoch falsch sein.
- Oft gibt es nicht nur eine Erklärung für einen Fehler. Über der nächstliegenden nicht die anderen vergessen. Das kann man aus Kriminalromanen lernen.
- Die Hardware oder das Programm auf ein Minimum reduzieren, zum Laufen bringen und dann eine Komponente nach der anderen hinzufügen.
- Immer nur *eine* Änderung vornehmen und dann testen, niemals mehrere zugleich.
- Fehler machen sich nicht immer dort bemerkbar, wo die Ursache liegt, sondern oft an anderer Stelle bzw. später.
- Gleichzeitigkeit begründet keinen Kausalzusammenhang.
- Fehlermeldungen sind entweder nichtssagend[54] oder irreführend, von ein paar Ausnahmen abgesehen. Also nicht wörtlich nehmen, ebensowenig wie die Aufforderung, die CD in das Disketten-Laufwerk zu schieben.
- Viel lesen, auch die Protokollfiles im System. Manchmal findet sich ein Hinweis in einer Datei oder unter einer Überschrift, die dem Anschein nach nichts mit dem Fehler zu tun hat. Insbesondere der RFC 2321: *RITA The Reliable Internetwork Troubleshooting Agent* enthält wertvolle Hinweise.
- Nachdem man gelesen hat, darf man auch fragen. Vielleicht hat ein Leidensgenosse aus dem wirklichen oder virtuellen Leben schon mit dem gleichen Fehler gerungen.
- Mit einem Freund oder Kollegen das Problem diskutieren[55]

Besonders übel sind Fehler, die auf eine Kombination von Ursachen zurückgehen, sowie solche, die nur gelegentlich auftreten.

Handbücher sind – vor allem wenn sie einen guten Index haben – eine Hilfe. Die erste Hürde ist jedoch, das richtige Stichwort zu finden. Manche Hersteller haben eine eigene Ausdrucksweise. Ein Server ist andernorts

[54]Besonders schätzen wir als System-Manager die Meldung *Fragen Sie Ihren System-Manager.*

[55]*Wenn Du etwas wissen willst und es durch Meditation nicht finden kannst, so rate ich dir, mein lieber, sinnreicher Freund, mit dem nächsten Bekannten, der dir aufstößt, darüber zu sprechen.* HEINRICH VON KLEIST, *Über die allmähliche Verfertigung der Gedanken beim Reden.*

das, was in UNIX ein Dämon ist, und eine Bezugszahl ist das, was ein C-Programmierer als Flag bezeichnet. Hat man sich das Vokabular erst einmal angeeignet, wird einiges leichter. Manche Aufgaben erscheinen nur so lange schwierig, bis man sie zum ersten Mal gelöst hat. Das Einrichten unserer ersten Mailing-Liste hat einen Tag beansprucht, die folgenden liefen nach jeweils einer Stunde. Also nicht am eigenen Verstand zweifeln und aufgeben, sondern weiterbohren. Der Gerechtigkeit halber muss gesagt werden, dass das Schreiben von verständlichen Handbüchern ebenfalls keine einfache Aufgabe ist. Ehe man zur Flasche oder zum Strick greift, kann man auch die Netnews befragen oder in Mailing-Listen sein Leid klagen.

Ein kleiner Tip noch. Wenn wir uns in ein neues Gebiet einarbeiten, legen wir ein Sudelheft, eine Kladde an, in das wir unsere Erleuchtungen und Erfahrungen formlos eintragen. Auch was nicht funktioniert hat, kommt hinein. Das Heft unterstützt das Gedächtnis und hilft dem Kollegen.

2.14.9 Begriffe Systemverwaltung

Folgende Begriffe sollten klarer geworden sein:

- Backup
- Benutzer, System-Manager, System-Entwickler
- Benutzerkonto (Account), Benutzername, Passwort
- Betriebssicherheit, Datensicherheit
- Prokolldateien (log-Dateien)
- Superuser (root)

Folgende Kommandos sollten beherrscht werden:

- `du`, `df` oder `bdf`
- `stty`
- `crontab`
- `find`
- `ping`
- `rpm` (auf Red-Hat- oder Mandrake-Distributionen)
- `shutdown`

2.14.10 Memo Systemverwaltung

- Ein UNIX-System darf nicht einfach ausgeschaltet werden, sondern muss vorher mittels `shutdown(1M)` heruntergefahren werden.
- Ein Benutzer muss in `/etc/passwd(4)` und `/etc/group(4)` eingetragen werden und sich ein nicht zu einfaches Passwort wählen. Er braucht ferner ein Home-Verzeichnis, in der Regel mit den Zugriffsrechten 700.

- Die Verwaltung logischer und physikalischer Geräte in heterogenen Netzen ist ein ständiger Elchtest für die Administratoren.
- Man unterscheidet Betriebssicherheit (Verfügbarkeit) und Datensicherheit (Datenintegrität).
- Datenschutz ist der Schutz auf individuelle Personen bezogener Daten vor Missbrauch, per Gesetz geregelt.
- Viren im weiteren Sinne (malicious software, malware) sind unerwünschte Programme oder Programmteile, die absichtlich Schaden anrichten. Unter UNIX selten, aber nicht unmöglich.
- Das Ziehen von Backup-Kopien ist lästig, ihr Besitz aber ungemein beruhigend. Anfangs unbedingt prüfen, ob auch das Zurückspielen klappt.

2.14.11 Übung Systemverwaltung

Viele Tätigkeiten in der Systemverwaltung setzen aus gutem Grund die Rechte eines Superusers voraus. Auf diese verzichten wir hier. Vielleicht dürfen Sie Ihrem System-Manager einmal bei seiner verantwortungsvollen Tätigkeit helfen. Wir schauen uns ein bisschen im Datei-System um:

`cd`	(ins Home-Verzeichnis wechseln)
`du`	(Plattenbelegung)
`df`	(dito, nur anders)
`bdf`	(dito, noch anders)
`find . -atime +30 -print`	(suche Ladenhüter)
`find . -size +100 -print`	(suche Speicherfresser)

Dann sehen wir uns die Datei `/etc/passwd(4)` an:

`pwget`	(Benutzereinträge, HP-UX)
`grget`	(Gruppeneinträge, HP-UX)
`less /etc/passwd`	(Benutzereinträge, Linux)
`less /etc/group`	(Gruppeneinträge, Linux)

und schließlich versuchen wir, die Konfiguration unserer Terminalschnittstelle zu verstehen:

`stty -a`	(in Sektion 1 nachlesen)

Mit diesem Kommando lassen sich die Einstellungen auch ändern, aber Vorsicht, das hat mit dem Roulettespiel einiges gemeinsam. Die Kommandos `tset(1)` oder `reset(1)` – sofern sie noch eingegeben werden können – setzen die Schnittstelle auf vernünftige Werte zurück.

Es wäre auch kein Fehler, wenn Sie mit Unterstützung durch Ihren System-Manager ein Backup Ihres Home-Verzeichnisses ziehen und wieder einspielen würden. Mit einem ausgetesteten Backup schläft sich's ruhiger.

2.14.12 Fragen zur Systemverwaltung

- Warum soll man ein UNIX-System nicht einfach ausschalten?
- Was gehört zu einem Benutzer-Account?
- Wie soll ein Passwort aussehen oder nicht aussehen?
- Was bedeuten Betriebssicherheit, Datensicherheit, Datenschutz?
- Was ist ein Backup? Welche Möglichkeiten gibt es, auch für den Normalbenutzer?

2.15 Exkurs über Informationen

Die im Text verstreuten Exkurse – Abschweifungen vom Thema UNIX, C und Internet – braucht man nicht sogleich zu lesen. Sie gehören zum Allgemeinwissen in der Informatik und runden das Spezialwissen über UNIX und C/C++ ab.

Im Abschnitt 1.1 *Was macht ein Computer?* auf Seite 2 sprachen wir von Informationen, Nachrichten oder Daten. Es gibt noch mehr Begriffe in diesem Wortfeld:

- Signal,
- Datum, Plural: Daten,
- Nachricht,
- Information,
- Wissen,
- Verstand, Vernunft, Intelligenz, Weisheit ...

Zur Frage des PILATUS versteigt sich die Informatik nicht, obwohl sie viel von *true* und *false* spricht. Wir lassen auch die Intelligenz natürlichen oder künstlichen Ursprungs außer Betracht – das heißt wir empfehlen das Nachdenken darüber als Übungsaufgabe – und beschränken uns auf die genauer, wenn auch nicht immer einheitlich bestimmten Begriffe von Signal bis Wissen.

Ein **Signal** ist die zeitliche Änderung einer physikalischen Größe, die von einer Signalquelle hervorgerufen wird mit dem Zweck, einem Signalempfänger eine Nachricht zu übermitteln. Nicht jede zeitliche Änderung einer physikalischen Größe ist ein Signal, der Zweck fehlt: ein Steigen der Lufttemperatur infolge Erwärmung durch die Sonne ist keines. Auch hängt das Signal vom Empfänger ab, der Warnruf eines Eichelhähers ist kein Signal für einen Menschen, wohl aber für seine Artgenossen. Die Zeit gehört mit zum

Signal, sie ist manchmal wichtiger (informationsträchtiger) als die sich ändernde physikalische Größe, denken Sie an die Haustürklingel. In der UNIX-Welt hat der Begriff *Signal* darüber hinaus eine besondere Bedeutung, siehe `signal(2)`.

Nimmt die Signalgröße nur Werte aus einem endlichen Vorrat deutlich voneinander unterschiedener (diskreter) Werte an, so haben wir ein **digitales** Signal im Gegensatz zu einem **analogen**. Die Signale der Verkehrsampeln sind digital, auch wenn sie nichts mit Zahlen zu tun haben. Die Zeigerstellung einer herkömmlichen Uhr hingegen ist ein analoges Signal.

Ein Element aus einer zur Darstellung von Informationen zwischen Quelle und Empfänger vereinbarten Menge digitaler Signale (Zeichenvorrat) ist ein **Zeichen** (character). Signale, die aus Zeichen bestehen, werden **Daten** genannt. *Daten* ist die Pluralform zu *Datum* = lat. das Gegebene. Einige Autoren verstehen unter Daten anders als obige Definition aus dem Informatik-Duden Nachrichten samt den ihnen zugeordneten Informationen. So oder so füllen die Daten den Massenspeicher immer schneller als erwartet.

Eine bestimmte (konkrete) **Nachricht**, getragen von einem Signal, überbringt eine von der Nachricht lösbare (abstrakte) **Information**. Ein Beispiel: die Information vom Untergang eines Fährschiffes läßt sich durch eine Nachricht in deutscher, englischer, französischer usw. Sprache übertragen. Die Information bleibt dieselbe, die Nachricht lautet jedesmal anders. Darüber hinaus kann jede dieser Nachrichten nochmals durch verschiedene Signale dargestellt werden. Der eine erfährt sie im Fernsehen, der andere im Radio, der dritte per Email oder Brief. Inwieweit die Nachricht die Information beeinflußt, also *nicht* von ihr zu lösen ist, macht Elend und Glanz der Übersetzung aus.

Eine Nachricht kann für mich belanglos sein (keine Information enthalten), falls ich sie entweder schon früher einmal empfangen habe oder sie aus anderen Gründen keinen Einfluß auf mein Verhalten hat. Dieselbe Nachricht kann für einen anderen Empfänger äußerst wichtig sein (viel Information enthalten), wenn beispielsweise auf der havarierten Fähre Angehörige waren.

Verschlüsselte Nachrichten ermöglichen nur dem Besitzer des Schlüssels die Entnahme der Information. Schließlich kommen auch gar nicht selten mehrdeutige Nachrichten vor. Ein Bauer versteht unter *Schönem Wetter* je nach dem Wetter der vergangenen Wochen etwas anderes als ein Tourist. Selbst ein Bergsteiger zieht auf manchen Hüttenanstiegen einen leichten Nieselregen einer gnadenlos strahlenden Sonne vor. Die kaum zu vermeidende Mehrdeutigkeit von Klausuraufgaben hat schon Gerichte beschäftigt. In ähnlicher Weise, wie hier zwischen Nachricht und Information unterschieden wird, trennt die Semantik (Bedeutungslehre, ein Zweig der Linguistik, die Lehre nicht von den Sprachen, sondern von der Sprache schlechthin) die Bezeichnung von der Bedeutung.

In einem Handwörterbuch von 1854 wird unter *Nachricht* die mündliche oder schriftliche Bekanntmachung einer in der Ferne geschehenen Sache verstanden, womit die Ortsabhängigkeit des Informationsgehaltes einer Nachricht angesprochen wird. Ein Blick in das Duden-Herkunftswörterbuch be-

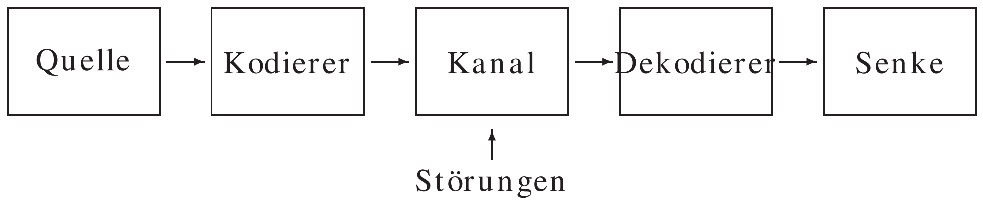

Abb. 2.13: Übertragung einer Information, Modell nach C. E. Shannon

lehrt uns, daß eine Nachricht ursprünglich etwas war, wonach man sich zu richten hatte, eine Anweisung. Und ein gewisser General CARL VON CLAUSEWITZ bezeichnet mit dem Worte *Nachrichten* etwas einseitig *die ganze Kenntnis, welche man von dem Feinde und seinem Lande hat, also die Grundlage aller eigenen Ideen und Handlungen*. Wir stimmen jedoch vollinhaltlich seiner Meinung zu, daß ein großer Teil der Nachrichten widersprechend, ein noch größerer falsch und bei weitem der größte einer ziemlichen Ungewißheit unterworfen sei. Deshalb hat der Mensch eine Fehlertoleranz entwickelt, um den ihn die Computer noch lange beneiden.

Im antiken Rom bedeutete *informieren* jemanden durch Unterweisung bilden oder formen, daher *informatio* = Begriff, Vorstellung, Darlegung, Unterweisung, Belehrung. Einen genaueren und daher nur begrenzt verwendbaren Begriff der Information gebraucht CLAUDE ELWOOD SHANNON in der von ihm begründeten **Informationstheorie**. Wir betrachten den Weg einer Nachricht von einer Nachrichtenquelle (source) durch einen Codierer (coder), einen Übertragungskanal (channel) und einen Decodierer (decoder) zum Empfänger oder zur Nachrichtensenke (sink), siehe Abbildung 2.13 auf Seite 284. Die Quelle sind Menschen, Meßgeräte oder ihrerseits zusammengesetzte Systeme. Der Codierer paßt die Quelle an den Kanal an, der Decodierer den Kanal an die Senke. Stellen Sie sich als Quelle einen Nachrichtensprecher vor, als Codierer die Technik vom Mikrofon bis zur Sendeantenne, als Kanal den Weg der elektromagnetischen Wellen, als Decodierer Ihr Radio von der Antenne bis zum Lautsprecher, und als Senke dürfen Sie selbst auftreten. Oder Sie sind die Quelle, Ihre Tastatur ist der Codierer, der Speicher ist der Kanal, die Hard- und Software für die Ausgabe (Bildschirm) bilden den Decodierer, und schließlich sind Sie oder ein anderer Benutzer die Senke. Die Quelle macht aus einer Information eine Nachricht und gibt formal betrachtet Zeichen mit einer zugehörigen Wahrscheinlichkeit von sich. Was die Zeichen bedeuten, interessiert SHANNON nicht, er kennt nur die trockene Statistik. Der Codierer setzt mittels einer Tabelle oder eines Regelwerks die Nachricht in eine für den Kanal geeignete Form um, beispielsweise Buchstaben in Folgen von 0 und 1. Der Decodierer macht das gleiche in umgekehrter Weise, wobei die Nachricht nicht notwendig die ursprüngliche Form annimmt: Die Ausgabe einer über die Tastatur eingegebenen Nachricht geschieht praktisch nie durch Tastenbewegungen. Der Kanal ist dadurch gekennzeichnet, daß er Signale verliert und auch hinzufügt. Die Senke zieht aus der Nachricht die Information heraus, möglichst die richtige. Das Ganze läßt sich zu einer stark mathematisch ausgerichteten Wissenschaft vertiefen, der man die Verbindung zum Computer nicht mehr ansieht.

Im Codieren und Decodieren steckt eine Menge Intelligenz. Eine Nachricht kann nämlich zweckmäßig codiert werden, das heißt so, daß sie wenig Ansprüche an den Kanal stellt. Ansprüche sind Zeit bei der Übertragung und Platzbedarf beim Speichern. Damit sind wir wieder bei UNIX: es gibt Programme wie `gzip(1)` zum Umcodieren von Daten, die ohne Informationsverlust die Anzahl der Bytes verringern, so daß weniger Speicher und bei der Übertragung weniger Zeit benötigt werden. Umgekehrt läßt sich die Sicherheit der Aufbewahrung und Übertragung durch eine Vermehrung der Bits oder Bytes verbessern. In vielen PCs wird daher 1 Byte in 9 Bits gespeichert. Bei der Bildverarbeitung wachsen die Datenmengen so gewaltig, daß man bei der Codierung Verluste hinnimmt, die die Senke gerade noch nicht bemerkt, genau so bei der Musik-CD.

Wissen auf Knopfdruck? Manches, was im Computer gespeichert ist, bezeichnen wir als **Wissen**, sobald es in unserem Kopf gelandet ist. Trotzdem zögern wir, beim Computer von Wissen zu sprechen (und schon gar nicht von Bewußtsein). Fragen wir ein Lexikon der Informatik: Wissen (knowledge) *ist eine geheimnisvolle Mischung aus Intuition, Erfahrung, Informiertheit, Bildung und Urteilskraft*. Ähnlich dunkel sind auch unsere eigenen Vorstellungen; befragen wir ein anderes Buch: *Wissen ist Information, die aufgeteilt, geformt, interpretiert, ausgewählt und umgewandelt wurde. Tatsachen allein sind noch kein Wissen. Damit Information zu Wissen wird, müssen auch die wechselseitigen ideellen Beziehungen klar sein*. Aus der Broschüre einer angesehenen Computerfirma: *Wissen ist die Gesamtheit der Kenntnisse, die durch Erfahrung oder Lernen erworben wurden*. Ende der Zitate, nichts ist klar.

Wissen wird eingeteilt in *Explizites Wissen* (explicit knowledge) und *Verborgenes Wissen* (unausgesprochenes, stilles, echtes Wissen; tacit knowledge). Ein Kind kann Radfahren lernen, es weiß dann, wie man radfährt. Aber selbst einem Ingenieur fällt es schwer, in Worten und von mir aus auch Zeichnungen zu beschreiben, wie man radfährt. Die Bemühungen des **Wissensmanagements** gehen dahin, möglichst viel Wissen in eine explizite und damit per Computer handhabbare Form zu bringen und gewinnträchtig anzuwenden[56].

Jemand, der alle Geschichtszahlen aus dem Großen Ploetz oder aus einer Enzyklopädie auswendig kann, wird zwar bestaunt, aber nicht als kenntnisreich oder klug angesehen, eher im Gegenteil. Erst wenn er die Tatsachen zu verknüpfen und auf neue Situationen anzuwenden weiß, beginnen wir, ihm Wissen zuzubilligen. Andererseits kommt das Denken nicht ohne Tatsachen aus, Geschichtswissen ohne die Kenntnis einiger Jahreszahlen ist kaum vorstellbar. Die Strukturierung der Tatsachen in Hierarchien (denken Sie an die Einordnung der Taschenratte alias Gopher) oder verwickelteren Ordnungen (semantischen Netzen) mit Kreuz-und-Quer-Beziehungen, das Verbinden von Tatsachen nach Regeln, die ihrerseits wieder geordnet sind, und die Anwen-

[56]Zur Einführung sehen Sie sich `http://www.bus.utexas.edu/kman/` an.

dung des Wissens auf noch nicht Gewußtes scheinen einen wesentlichen Teil des Wissens auszumachen. Das den Computern beizubringen, ist ein Ziel der Bemühungen um die **Künstliche Intelligenz** (KI, englisch AI). Datenbanken, Expertensysteme und Hypermedia sind erste Schritte auf einem vermutlich langen Weg. Ehe wir uns weiter auf dessen Glatteis begeben, verweisen wir auf die Literatur.

Wenden wir uns zum Abschluß wieder den bodennäheren Schichten der Information zu. Viele Pannen im Berufs-, Vereins-, Partei- und Familienleben rühren einfach daher, daß die Informationen nicht richtig flossen. Dabei lassen sich solche Pannen mit relativ wenig Aufwand vermeiden, indem man frühzeitig dem Informationswesen etwas Aufmerksamkeit widmet. Einige Erfahrungen eines ergrauten Post- und Webmasters:

- Falsche Informationen sind gefährlicher als fehlende (Das wissen die Geheimdienste schon lange).
- Der Zeitpunkt der Übermittlung einer Information kann wichtiger sein als der Inhalt.
- Viele Informationen haben außer ihrem Sachinhalt auch eine emotionelle Seite, der nicht mit Sachargumenten beizukommen ist. Beispielsweise spielt die Reihenfolge, in der Empfänger benachrichtigt werden, eine Rolle.
- Guten Informationen muß man hinterherlaufen, überflüssige kommen von allein.
- Informationen altern.
- Eine Information zusammenstellen, ist eine Sache, sie auf dem laufenden zu halten, eine andere. Die zweite Aufgabe ist mühsamer, da sie kein Ende nimmt. Gilt insbesondere für die Einrichtung von WWW-Servern und -Seiten.

... aber die Daten fehlen, um den ganzen
Nonsens richtig zu überblicken –
G. BENN, Drei alte Männer

A Zahlensysteme

Außer dem **Dezimalsystem** sind das **Dual-**, das **Oktal-** und das **Hexadezimalsystem** gebräuchlich. Ferner spielt das **Binär codierte Dezimalsystem (BCD)** bei manchen Anwendungen eine Rolle. Bei diesem sind die einzelnen Dezimalstellen für sich dual dargestellt. Die folgende Tabelle enthält die Werte von 0 bis dezimal 255. Bequemlichkeitshalber sind auch die zugeordneten ASCII-Zeichen aufgeführt.

dezimal	dual	oktal	hex	BCD	ASCII
0	0	0	0	0	nul
1	1	1	1	1	soh
2	10	2	2	10	stx
3	11	3	3	11	etx
4	100	4	4	100	eot
5	101	5	5	101	enq
6	110	6	6	110	ack
7	111	7	7	111	bel
8	1000	10	8	1000	bs
9	1001	11	9	1001	ht
10	1010	12	a	1.0	lf
11	101	13	b	1.1	vt
12	1100	14	c	1.10	ff
13	1101	15	d	1.11	cr
14	1110	16	e	1.100	so
15	1111	17	f	1.101	si
16	10000	20	10	1.110	dle
17	10001	21	11	1.111	dc1
18	10010	22	12	1.1000	dc2
19	10011	23	13	1.1001	dc3
20	10100	24	14	10.0	dc4
21	10101	25	15	10.1	nak
22	10110	26	16	10.10	syn
23	10111	27	17	10.11	etb
24	11000	30	18	10.100	can
25	11001	31	19	10.101	em
26	11010	32	1a	10.110	sub
27	11011	33	1b	10.111	esc
28	11100	34	1c	10.1000	fs
29	11101	35	1d	10.1001	gs

30	11110	36	1e	11.0	rs
31	11111	37	1f	11.1	us
32	100000	40	20	11.10	space
33	100001	41	21	11.11	!
34	100010	42	22	11.100	”
35	100011	43	23	11.101	#
36	100100	44	24	11.110	$
37	100101	45	25	11.111	%
38	100110	46	26	11.1000	&
39	100111	47	27	11.1001	’
40	101000	50	28	100.0	(
41	101001	51	29	100.1	)
42	101010	52	2a	100.10	*
43	101011	53	2b	100.11	+
44	101100	54	2c	100.100	,
45	101101	55	2d	100.101	-
46	101110	56	2e	100.110	.
47	101111	57	2f	100.111	/
48	110000	60	30	100.1000	0
49	110001	61	31	100.1001	1
50	110010	62	32	101.0	2
51	110011	63	33	101.1	3
52	110100	64	34	101.10	4
53	110101	65	35	101.11	5
54	110110	66	36	101.100	6
55	110111	67	37	101.101	7
56	111000	70	38	101.110	8
57	111001	71	39	101.111	9
58	111010	72	3a	101.1000	:
59	111011	73	3b	101.1001	;
60	111100	74	3c	110.0	<
61	111101	75	3d	110.1	=
62	111110	76	3e	110.10	>
63	111111	77	3f	110.11	?
64	1000000	100	40	110.100	@
65	1000001	101	41	110.101	A
66	1000010	102	42	110.110	B
67	1000011	103	43	110.111	C
68	1000100	104	44	110.1000	D
69	1000101	105	45	110.1001	E
70	1000110	106	46	111.0	F
71	1000111	107	47	111.1	G
72	1001000	110	48	111.10	H
73	1001001	111	49	111.11	I
74	1001010	112	4a	111.100	J
75	1001011	113	4b	111.101	K
76	1001100	114	4c	111.110	L

77	1001101	115	4d	111.111	M
78	1001110	116	4e	111.1000	N
79	1001111	117	4f	111.1001	O
80	1010000	120	50	1000.0	P
81	1010001	121	51	1000.1	Q
82	1010010	122	52	1000.10	R
83	1010011	123	53	1000.11	S
84	1010100	124	54	1000.100	T
85	1010101	125	55	1000.101	U
86	1010110	126	56	1000.110	V
87	1010111	127	57	1000.111	W
88	1011000	130	58	1000.1000	X
89	1011001	131	59	1000.1001	Y
90	1011010	132	5a	1001.0	Z
91	1011011	133	5b	1001.1	[
92	1011100	134	5c	1001.10	\
93	1011101	135	5d	1001.11	]
94	1011110	136	5e	1001.100	^
95	1011111	137	5f	1001.101	_
96	1100000	140	60	1001.110	`
97	1100001	141	61	1001.111	a
98	1100010	142	62	1001.1000	b
99	1100011	143	63	1001.1001	c
100	1100100	144	64	1.0.0	d
101	1100101	145	65	1.0.1	e
102	1100110	146	66	1.0.10	f
103	1100111	147	67	1.0.11	g
104	1101000	150	68	1.0.100	h
105	1101001	151	69	1.0.101	i
106	1101010	152	6a	1.0.110	j
107	1101011	153	6b	1.0.111	k
108	1101100	154	6c	1.0.1000	l
109	1101101	155	6d	1.0.1001	m
110	1101110	156	6e	1.1.0	n
111	1101111	157	6f	1.1.1	o
112	1110000	160	70	1.1.10	p
113	1110001	161	71	1.1.11	q
114	1110010	162	72	1.1.100	r
115	1110011	163	73	1.1.101	s
116	1110100	164	74	1.1.110	t
117	1110101	165	75	1.1.111	u
118	1110110	166	76	1.1.1000	v
119	1110111	167	77	1.1.1001	w
120	1111000	170	78	1.10.0	x
121	1111001	171	79	1.10.1	y
122	1111010	172	7a	1.10.10	z
123	1111011	173	7b	1.10.11	{

124	1111100	174	7c	1.10.100	|
125	1111101	175	7d	1.10.101	}
126	1111110	176	7e	1.10.110	~
127	1111111	177	7f	1.10.111	del
128	10000000	200	80	1.10.1000	
129	10000001	201	81	1.10.1001	
130	10000010	202	82	1.11.0	
131	10000011	203	83	1.11.1	
132	10000100	204	84	1.11.10	
133	10000101	205	85	1.11.11	
134	10000110	206	86	1.11.100	
135	10000111	207	87	1.11.101	
136	10001000	210	88	1.11.110	
137	10001001	211	89	1.11.111	
138	10001010	212	8a	1.11.1000	
139	10001011	213	8b	1.11.1001	
140	10001100	214	8c	1.100.0	
141	10001101	215	8d	1.100.1	
142	10001110	216	8e	1.100.10	
143	10001111	217	8f	1.100.11	
144	10010000	220	90	1.100.100	
145	10010001	221	91	1.100.101	
146	10010010	222	92	1.100.110	
147	10010011	223	93	1.100.111	
148	10010100	224	94	1.100.1000	
149	10010101	225	95	1.100.1001	
150	10010110	226	96	1.101.0	
151	10010111	227	97	1.101.1	
152	10011000	230	98	1.101.10	
153	10011001	231	99	1.101.11	
154	10011010	232	9a	1.101.100	
155	10011011	233	9b	1.101.101	
156	10011100	234	9c	1.101.110	
157	10011101	235	9d	1.101.111	
158	10011110	236	9e	1.101.1000	
159	10011111	237	9f	1.101.1001	
160	10100000	240	a0	1.110.0	
161	10100001	241	a1	1.110.1	
162	10100010	242	a2	1.110.10	
163	10100011	243	a3	1.110.11	
164	10100100	244	a4	1.110.100	
165	10100101	245	a5	1.110.101	
166	10100110	246	a6	1.110.110	
167	10100111	247	a7	1.110.111	
168	10101000	250	a8	1.110.1000	
169	10101001	251	a9	1.110.1001	
170	10101010	252	aa	1.111.0	

171	10101011	253	ab	1.111.1
172	10101100	254	ac	1.111.10
173	10101101	255	ad	1.111.11
174	10101110	256	ae	1.111.100
175	10101111	257	af	1.111.101
176	10110000	260	b0	1.111.110
177	10110001	261	b1	1.111.111
178	10110010	262	b2	1.111.1000
179	10110011	263	b3	1.111.1001
180	10110100	264	b4	1.1000.0
181	10110101	265	b5	1.1000.1
182	10110110	266	b6	1.1000.10
183	10110111	267	b7	1.1000.11
184	10111000	270	b8	1.1000.100
185	10111001	271	b9	1.1000.101
186	10111010	272	ba	1.1000.110
187	10111011	273	bb	1.1000.111
188	10111100	274	bc	1.1000.1000
189	10111101	275	bd	1.1000.1001
190	10111110	276	be	1.1001.0
191	10111111	277	bf	1.1001.1
192	11000000	300	c0	1.1001.10
193	11000001	301	c1	1.1001.11
194	11000010	302	c2	1.1001.100
195	11000011	303	c3	1.1001.101
196	11000100	304	c4	1.1001.110
197	11000101	305	c5	1.1001.111
198	11000110	306	c6	1.1001.1000
199	11000111	307	c7	1.1001.1001
200	11001000	310	c8	10.0.0
201	11001001	311	c9	10.0.1
202	11001010	312	ca	10.0.10
203	11001011	313	cb	10.0.11
204	11001100	314	cc	10.0.100
205	11001101	315	cd	10.0.101
206	11001110	316	ce	10.0.110
207	11001111	317	cf	10.0.111
208	11010000	320	d0	10.0.1000
209	11010001	321	d1	10.0.1001
210	11010010	322	d2	10.1.0
211	11010011	323	d3	10.1.1
212	11010100	324	d4	10.1.10
213	11010101	325	d5	10.1.11
214	11010110	326	d6	10.1.100
215	11010111	327	d7	10.1.101
216	11011000	330	d8	10.1.110
217	11011001	331	d9	10.1.111

218	11011010	332	da	10.1.1000
219	11011011	333	db	10.1.1001
220	11011100	334	dc	10.10.0
221	11011101	335	dd	10.10.1
222	11011110	336	de	10.10.10
223	11011111	337	df	10.10.11
224	11100000	340	e0	10.10.100
225	11100001	341	e1	10.10.101
226	11100010	342	e2	10.10.110
227	11100011	343	e3	10.10.111
228	11100100	344	e4	10.10.1000
229	11100101	345	e5	10.10.1001
230	11100110	346	e6	10.11.0
231	11100111	347	e7	10.11.1
232	11101000	350	e8	10.11.10
233	11101001	351	e9	10.11.11
234	11101010	352	ea	10.11.100
235	11101011	353	eb	10.11.101
236	11101100	354	ec	10.11.110
237	11101101	355	ed	10.11.111
238	11101110	356	ee	10.11.1000
239	11101111	357	ef	10.11.1001
240	11110000	360	f0	10.100.0
241	11110001	361	f1	10.100.1
242	11110010	362	f2	10.100.10
243	11110011	363	f3	10.100.11
244	11110100	364	f4	10.100.100
245	11110101	365	f5	10.100.101
246	11110110	366	f6	10.100.110
247	11110111	367	f7	10.100.111
248	11111000	370	f8	10.100.1000
249	11111001	371	f9	10.100.1001
250	11111010	372	fa	10.101.0
251	11111011	373	fb	10.101.1
252	11111100	374	fc	10.101.10
253	11111101	375	fd	10.101.11
254	11111110	376	fe	10.101.100
255	11111111	377	ff	10.101.101

B Zeichensätze und Sondertasten

B.1 EBCDIC, ASCII, Roman8, IBM-PC

Die Zeichensätze sind in den Ein- und Ausgabegeräten (Terminal, Drucker) gespeicherte Tabellen, die die Zeichen in Zahlen und zurück umsetzen.

dezimal	oktal	EBCDIC	ASCII-7	Roman8	IBM-PC
0	0	nul	nul	nul	nul
1	1	soh	soh	soh	Grafik
2	2	stx	stx	stx	Grafik
3	3	etx	etx	etx	Grafik
4	4	pf	eot	eot	Grafik
5	5	ht	enq	enq	Grafik
6	6	lc	ack	ack	Grafik
7	7	del	bel	bel	bel
8	10		bs	bs	Grafik
9	11	rlf	ht	ht	ht
10	12	smm	lf	lf	lf
11	13	vt	vt	vt	home
12	14	ff	ff	ff	ff
13	15	cr	cr	cr	cr
14	16	so	so	so	Grafik
15	17	si	si	si	Grafik
16	20	dle	dle	dle	Grafik
17	21	dc1	dc1	dc1	Grafik
18	22	dc2	dc2	dc2	Grafik
19	23	dc3	dc3	dc3	Grafik
20	24	res	dc4	dc4	Grafik
21	25	nl	nak	nak	Grafik
22	26	bs	syn	syn	Grafik
23	27	il	etb	etb	Grafik
24	30	can	can	can	Grafik
25	31	em	em	em	Grafik
26	32	cc	sub	sub	Grafik
27	33		esc	esc	Grafik
28	34	ifs	fs	fs	cur right
29	35	igs	gs	gs	cur left
30	36	irs	rs	rs	cur up
31	37	ius	us	us	cur down
32	40	ds	space	space	space
33	41	sos	!	!	!
34	42	fs	”	”	”

35	43		#	#	#
36	44	byp	$	$	$
37	45	lf	%	%	%
38	46	etb	&	&	&
39	47	esc	'	'	'
40	50		(	(	(
41	51		)	)	)
42	52	sm	*	*	*
43	53		+	+	+
44	54		,	,	,
45	55	enq	-	-	-
46	56	ack	.	.	.
47	57	bel	/	/	/
48	60		0	0	0
49	61		1	1	1
50	62	syn	2	2	2
51	63		3	3	3
52	64	pn	4	4	4
53	65	rs	5	5	5
54	66	uc	6	6	6
55	67	eot	7	7	7
56	70		8	8	8
57	71		9	9	9
58	72		:	:	:
59	73		;	;	;
60	74	dc4	<	<	<
61	75	nak	=	=	=
62	76		>	>	>
63	77	sub	?	?	?
64	100	space	@	@	@
65	101		A	A	A
66	102	â	B	B	B
67	103	ä	C	C	C
68	104	à	D	D	D
69	105	á	E	E	E
70	106	ã	F	F	F
71	107	å	G	G	G
72	110	ç	H	H	H
73	111	ñ	I	I	I
74	112	[	J	J	J
75	113	.	K	K	K
76	114	<	L	L	L
77	115	(	M	M	M
78	116	+	N	N	N
79	117	!	O	O	O
80	120	&	P	P	P
81	121	é	Q	Q	Q

82	122	ê	R	R	R
83	123	ë	S	S	S
84	124	è	T	T	T
85	125	í	U	U	U
86	126	î	V	V	V
87	127	ï	W	W	W
88	130	ì	X	X	X
89	131	ß	Y	Y	Y
90	132	]	Z	Z	Z
91	133	$	[	[	[
92	134	*	\	\	\
93	135	)	]	]	]
94	136	;	^	^	^
95	137	^	_	_	_
96	140	–	`	`	`
97	141	/	a	a	a
98	142	Â	b	b	b
99	143	Ä	c	c	c
100	144	À	d	d	d
101	145	Á	e	e	e
102	146	Ã	f	f	f
103	147	Å	g	g	g
104	150	Ç	h	h	h
105	151	Ñ	i	i	i
106	152	|	j	j	j
107	153	,	k	k	k
108	154	%	l	l	l
109	155	_	m	m	m
110	156	>	n	n	n
111	157	?	o	o	o
112	160	ø	p	p	p
113	161	É	q	q	q
114	162	Ê	r	r	r
115	163	Ë	s	s	s
116	164	È	t	t	t
117	165	Í	u	u	u
118	166	Î	v	v	v
119	167	Ï	w	w	w
120	170	Ì	x	x	x
121	171	‘	y	y	y
122	172	:	z	z	z
123	173	#	{	{	{
124	174	@	|	|	|
125	175	’	}	}	}
126	176	=	~	~	~
127	177	”	del	del	Grafik

128	200	Ø		Ç
129	201	a		ü
130	202	b		é
131	203	c		â
132	204	d		ä
133	205	e		à
134	206	f		å
135	207	g		ç
136	210	h		ê
137	211	i		ë
138	212	$\ll$		è
139	213	$\gg$		ı
140	214			î
141	215	ý		ì
142	216			Ä
143	217	$\pm$		Å
144	220			É
145	221	j		œ
146	222	k		Æ
147	223	l		ô
148	224	m		ö
149	225	n		ò
150	226	o		û
151	227	p		ù
152	230	q		y
153	231	r		Ö
154	232	ª		Ü
155	233	º		
156	234	æ		£
157	235	–		Yen
158	236	Æ		Pt
159	237			*f*
160	240	μ		á
161	241	$\sim$	À	í
162	242	s	Â	ó
163	243	t	È	ú
164	244	u	Ê	ñ
165	245	v	Ë	Ñ
166	246	w	Î	ª
167	247	x	Ï	º
168	250	y	’	¿
169	251	z	‘	Grafik
170	252	¡	^	Grafik
171	253	¿		1/2
172	254		~	1/4
173	255	Ý	Ù	¡

174	256		Û	≪
175	257			≫
176	260			Grafik
177	261	£		Grafik
178	262	Yen		Grafik
179	263		◦	Grafik
180	264	*f*	Ç	Grafik
181	265	§	ç	Grafik
182	266	¶	Ñ	Grafik
183	267		ñ	Grafik
184	270		¡	Grafik
185	271		¿	Grafik
186	272			Grafik
187	273	|	£	Grafik
188	274	–	Yen	Grafik
189	275		§	Grafik
190	276		*f*	Grafik
191	277	=		Grafik
192	300	{	â	Grafik
193	301	A	ê	Grafik
194	302	B	ô	Grafik
195	303	C	û	Grafik
196	304	D	á	Grafik
197	305	E	é	Grafik
198	306	F	ó	Grafik
199	307	G	ú	Grafik
200	310	H	à	Grafik
201	311	I	è	Grafik
202	312		ò	Grafik
203	313	ô	ù	Grafik
204	314	ö	ä	Grafik
205	315	ò	ë	Grafik
206	316	ó	ö	Grafik
207	317	õ	ü	Grafik
208	320	}	Å	Grafik
209	321	J	î	Grafik
210	322	K	Ø	Grafik
211	323	L	Æ	Grafik
212	324	M	å	Grafik
213	325	N	í	Grafik
214	326	O	ø	Grafik
215	327	P	æ	Grafik
216	330	Q	Ä	Grafik
217	331	R	ì	Grafik
218	332		Ö	Grafik
219	333	û	Ü	Grafik

220	334	ü	É	Grafik
221	335	ù	ı	Grafik
222	336	ú	ß	Grafik
223	337	y	Ô	Grafik
224	340	\	Á	α
225	341		Ã	β
226	342	S	ã	Γ
227	343	T		π
228	344	U		Σ
229	345	V	Í	σ
230	346	W	Ì	μ
231	347	X	Ó	τ
232	350	Y	Ò	Φ
233	351	Z	Õ	θ
234	352		õ	Ω
235	353	Ô	Š	δ
236	354	Ö	š	∞
237	355	Ò	Ú	Ø
238	356	Ó	Y	$\in$
239	357	Õ	y	$\cap$
240	360	0	thorn	$\equiv$
241	361	1	Thorn	$\pm$
242	362	2		$\geq$
243	363	3		$\leq$
244	364	4		Haken
245	365	5		Haken
246	366	6	–	$\div$
247	367	7	1/4	$\approx$
248	370	8	1/2	$\circ$
249	371	9	a̲	$\bullet$
250	372		o̲	$\cdot$
251	373	Û	$\ll$	$\surd$
252	374	Ü	$\sqcup$	n
253	375	Ù	$\gg$	2
254	376	Ú	$\pm$	$\sqcup$
255	377			(FF)

B.2 German-ASCII

Falls das Ein- oder Ausgabegerät einen deutschen 7-Bit-ASCII-Zeichensatz enthält, sind folgende Ersetzungen der amerikanischen Zeichen durch deutsche Sonderzeichen üblich:

Nr.	US-Zeichen	US-ASCII	German ASCII

91	linke eckige Klammer	[	Ä
92	Backslash	\	Ö
93	rechte eckige Klammer	]	Ü
123	linke geschweifte Klammer	{	ä
124	senkrechter Strich	|	ö
125	rechte geschweifte Klammer	}	ü
126	Tilde	~	ß

Achtung: Der IBM-PC und Ausgabegeräte von Hewlett-Packard verwenden keinen 7-Bit-ASCII-Zeichensatz, sondern eigene 8-Bit-Zeichensätze, die die Sonderzeichen unter Nummern höher 127 enthalten, siehe vorhergehende Tabelle.

B.3 ASCII-Steuerzeichen

Die Steuerzeichen der Zeichensätze dienen der Übermittlung von Befehlen und Informationen an das empfangende Gerät und nicht der Ausgabe eines sicht- oder druckbaren Zeichens. Die Ausgabegeräte kennen in der Regel jedoch einen Modus (transparent, Monitor, Display Functions), in der die Steuerzeichen nicht ausgeführt, sondern angezeigt werden. Die meisten Steuerzeichen belegen keine eigene Taste auf der Tastatur, sondern werden als Kombination aus der control-Taste und einer Zeichentaste eingegeben. In C/C++ läßt sich jedes Zeichen durch seine oktale Nummer in der Form `\123` oder durch seine hexadezimale Nummer in der Form `\x53` eingeben (hier das `S`).

dezimal	C-Konst.	ASCII	Bedeutung	Tasten
0	`\x00`	nul	ASCII-Null	control @
1		soh	Start of heading	control a
2		stx	Start of text	control b
3		etx	End of text	control c
4		eot	End of transmission	control d
5		enq	Enquiry	control e
6		ack	Acknowledge	control f
7	`\a`	bel	Bell	control g
8	`\b`	bs	Backspace	control h, BS
9	`\t`	ht	Horizontal tab	control i, TAB
10	`\n`	lf	Line feed	control j, LF
11	`\v`	vt	Vertical tab	control k
12	`\f`	ff	Form feed	control l
13	`\r`	cr	Carriage return	control m, RETURN
14		so	Shift out	control n
15		si	Shift in	control o
16		dle	Data link escape	control p
17		dc1	Device control 1, xon	control q
18		dc2	Device control 2, tape	control r
19		dc3	Device control 3, xoff	control s

20		dc4	Device control 4, tape	control t
21		nak	Negative acknowledge	control u
22		syn	Synchronous idle	control v
23		etb	End transmission block	control w
24		can	Cancel	control x
25		em	End of medium	control y
26		sub	Substitute	control z
27	`\x1b`	esc	Escape	control [, ESC
28		fs	File separator	control \
29		gs	Group separator	control]
30		rs	Record separator	control ^
31		us	Unit separator	control _
127		del	Delete	DEL, RUBOUT

B.4 Latin-1 (ISO 8859-1)

Die internationale Norm ISO 8859 beschreibt gegenwärtig zehn Zeichensätze, die jedes Zeichen durch jeweils ein Byte darstellen. Jeder Zeichensatz umfaßt also maximal 256 druckbare Zeichen und Steuerzeichen. Der erste – Latin-1 genannt – ist für west- und mitteleuropäische Sprachen – darunter Deutsch – vorgesehen. Latin-2 deckt Mittel- und Osteuropa ab, soweit das lateinische Alphabet verwendet wird. Wer einen polnisch-deutschen Text schreiben will, braucht Latin 2. Die deutschen Sonderzeichen liegen in Latin 1 bis 6 an denselben Stellen. Weiteres siehe in der ISO-Norm und im RFC 1345 *Character Mnemonics and Character Sets* vom Juni 1992. Auch `http://wwwwbs.cs.tu-berlin.de/~czyborra/charsets/` hilft weiter.

Die erste Hälfte (0 – 127) aller Latin-Zeichensätze stimmt mit US-ASCII überein, die zweite mit keinem der anderen Zeichensätze. Zu jedem Zeichen gehört eine standardisierte verbale Bezeichnung. Einige Zeichen wie das isländische Thorn oder das Cent-Zeichen konnten hier mit LaTeX nicht dargestellt werden.

dezimal	oktal	hex	Zeichen	Bezeichnung
000	000	00	nu	Null (nul)
001	001	01	sh	Start of heading (soh)
002	002	02	sx	Start of text (stx)
003	003	03	ex	End of text (etx)
004	004	04	et	End of transmission (eot)
005	005	05	eq	Enquiry (enq)
006	006	06	ak	Acknowledge (ack)
007	007	07	bl	Bell (bel)
008	010	08	bs	Backspace (bs)
009	011	09	ht	Character tabulation (ht)
010	012	0a	lf	Line feed (lf)
011	013	0b	vt	Line tabulation (vt)

012	014	0c	ff	Form feed (ff)
013	015	0d	cr	Carriage return (cr)
014	016	0e	so	Shift out (so)
015	017	0f	si	Shift in (si)
016	020	10	dl	Datalink escape (dle)
017	021	11	d1	Device control one (dc1)
018	022	12	d2	Device control two (dc2)
019	023	13	d3	Device control three (dc3)
020	024	14	d4	Device control four (dc4)
021	025	15	nk	Negative acknowledge (nak)
022	026	16	sy	Synchronous idle (syn)
023	027	17	eb	End of transmission block (etb)
024	030	18	cn	Cancel (can)
025	031	19	em	End of medium (em)
026	032	1a	sb	Substitute (sub)
027	033	1b	ec	Escape (esc)
028	034	1c	fs	File separator (is4)
029	035	1d	gs	Group separator (is3)
030	036	1e	rs	Record separator (is2)
031	037	1f	us	Unit separator (is1)
032	040	20	sp	Space
033	041	21	!	Exclamation mark
034	042	22	”	Quotation mark
035	043	23	#	Number sign
036	044	24	$	Dollar sign
037	045	25	%	Percent sign
038	046	26	&	Ampersand
039	047	27	’	Apostrophe
040	050	28	(	Left parenthesis
041	051	29	)	Right parenthesis
042	052	2a	*	Asterisk
043	053	2b	+	Plus sign
044	054	2c	,	Comma
045	055	2d	-	Hyphen-Minus
046	056	2e	.	Full stop
047	057	2f	/	Solidus
048	060	30	0	Digit zero
049	061	31	1	Digit one
050	062	32	2	Digit two
051	063	33	3	Digit three
052	064	34	4	Digit four
053	065	35	5	Digit five
054	066	36	6	Digit six
055	067	37	7	Digit seven
056	070	38	8	Digit eight
057	071	39	9	Digit nine
058	072	3a	:	Colon

059	073	3b	;	Semicolon
060	074	3c	<	Less-than sign
061	075	3d	=	Equals sign
062	076	3e	>	Greater-than sign
063	077	3f	?	Question mark
064	100	40	@	Commercial at
065	101	41	A	Latin capital letter a
066	102	42	B	Latin capital letter b
067	103	43	C	Latin capital letter c
068	104	44	D	Latin capital letter d
069	105	45	E	Latin capital letter e
070	106	46	F	Latin capital letter f
071	107	47	G	Latin capital letter g
072	110	48	H	Latin capital letter h
073	111	49	I	Latin capital letter i
074	112	4a	J	Latin capital letter j
075	113	4b	K	Latin capital letter k
076	114	4c	L	Latin capital letter l
077	115	4d	M	Latin capital letter m
078	116	4e	N	Latin capital letter n
079	117	4f	O	Latin capital letter o
080	120	50	P	Latin capital letter p
081	121	51	Q	Latin capital letter q
082	122	52	R	Latin capital letter r
083	123	53	S	Latin capital letter s
084	124	54	T	Latin capital letter t
085	125	55	U	Latin capital letter u
086	126	56	V	Latin capital letter v
087	127	57	W	Latin capital letter w
088	130	58	X	Latin capital letter x
089	131	59	Y	Latin capital letter y
090	132	5a	Z	Latin capital letter z
091	133	5b	[	Left square bracket
092	134	5c	\	Reverse solidus
093	135	5d	]	Right square bracket
094	136	5e	^	Circumflex accent
095	137	5f	_	Low line
096	140	60	‘	Grave accent
097	141	61	a	Latin small letter a
098	142	62	b	Latin small letter b
099	143	63	c	Latin small letter c
100	144	64	d	Latin small letter d
101	145	65	e	Latin small letter e
102	146	66	f	Latin small letter f
103	147	67	g	Latin small letter g
104	150	68	h	Latin small letter h
105	151	69	i	Latin small letter i

106	152	6a	j	Latin small letter j
107	153	6b	k	Latin small letter k
108	154	6c	l	Latin small letter l
109	155	6d	m	Latin small letter m
110	156	6e	n	Latin small letter n
111	157	6f	o	Latin small letter o
112	160	70	p	Latin small letter p
113	161	71	q	Latin small letter q
114	162	72	r	Latin small letter r
115	163	73	s	Latin small letter s
116	164	74	t	Latin small letter t
117	165	75	u	Latin small letter u
118	166	76	v	Latin small letter v
119	167	77	w	Latin small letter w
120	170	78	x	Latin small letter x
121	171	79	y	Latin small letter y
122	172	7a	z	Latin small letter z
123	173	7b	{	Left curly bracket
124	174	7c	|	Vertical line
125	175	7d	}	Right curly bracket
126	176	7e	~	Tilde
127	177	7f	dt	Delete (del)
128	200	80	pa	Padding character (pad)
129	201	81	ho	High octet preset (hop)
130	202	82	bh	Break permitted here (bph)
131	203	83	nh	No break here (nbh)
132	204	84	in	Index (ind)
133	205	85	nl	Next line (nel)
134	206	86	sa	Start of selected area (ssa)
135	207	87	es	End of selected area (esa)
136	210	88	hs	Character tabulation set (hts)
137	211	89	hj	Character tabulation with justification (htj)
138	212	8a	vs	Line tabulation set (vts)
139	213	8b	pd	Partial line forward (pld)
140	214	8c	pu	Partial line backward (plu)
141	215	8d	ri	Reverse line feed (ri)
142	216	8e	s2	Single-shift two (ss2)
143	217	8f	s3	Single-shift three (ss3)
144	220	90	dc	Device control string (dcs)
145	221	91	p1	Private use one (pu1)
146	222	92	p2	Private use two (pu2)
147	223	93	ts	Set transmit state (sts)
148	224	94	cc	Cancel character (cch)
149	225	95	mw	Message waiting (mw)
150	226	96	sg	Start of guarded area (spa)
151	227	97	eg	End of guarded area (epa)
152	230	98	ss	Start of string (sos)

153	231	99	gc	Single graphic character introducer (sgci)
154	232	9a	sc	Single character introducer (sci)
155	233	9b	ci	Control sequence introducer (csi)
156	234	9c	st	String terminator (st)
157	235	9d	oc	Operating system command (osc)
158	236	9e	pm	Privacy message (pm)
159	237	9f	ac	Application program command (apc)
160	240	a0	ns	No-break space
161	241	a1	¡	Inverted exclamation mark
162	242	a2		Cent sign
163	243	a3	£	Pound sign
164	244	a4		Currency sign (künftig Euro?)
165	245	a5		Yen sign
166	246	a6		Broken bar
167	247	a7	§	Section sign
168	250	a8		Diaresis
169	251	a9	©	Copyright sign
170	252	aa	^{a}	Feminine ordinal indicator
171	253	ab	$\ll$	Left-pointing double angle quotation mark
172	254	ac	$\neg$	Not sign
173	255	ad	-	Soft hyphen
174	256	ae		Registered sign
175	257	af	$^-$	Overline
176	260	b0	$^\circ$	Degree sign
177	261	b1	$\pm$	Plus-minus sign
178	262	b2	^{2}	Superscript two
179	263	b3	^{3}	Superscript three
180	264	b4	’	Acute accent
181	265	b5	μ	Micro sign
182	266	b6	¶	Pilcrow sign
183	267	b7	·	Middle dot
184	270	b8	¸	Cedilla
185	271	b9	^{1}	Superscript one
186	272	ba	$^\circ$	Masculine ordinal indicator
187	273	bb	$\gg$	Right-pointing double angle quotation mark
188	274	bc	1/4	Vulgar fraction one quarter
189	275	bd	1/2	Vulgar fraction one half
190	276	be	3/4	Vulgar fraction three quarters
191	277	bf	¿	Inverted question mark
192	300	c0	À	Latin capital letter a with grave
193	301	c1	Á	Latin capital letter a with acute
194	302	c2	Â	Latin capital letter a with circumflex
195	303	c3	Ã	Latin capital letter a with tilde
196	304	c4	Ä	Latin capital letter a with diaresis
197	305	c5	Å	Latin capital letter a with ring above
198	306	c6	Æ	Latin capital letter ae

199	307	c7	Ç	Latin capital letter c with cedilla
200	310	c8	È	Latin capital letter e with grave
201	311	c9	É	Latin capital letter e with acute
202	312	ca	Ê	Latin capital letter e with circumflex
203	313	cb	Ë	Latin capital letter e with diaresis
204	314	cc	Ì	Latin capital letter i with grave
205	315	cd	Í	Latin capital letter i with acute
206	316	ce	Î	Latin capital letter i with circumflex
207	317	cf	Ï	Latin capital letter i with diaresis
208	320	d0		Latin capital letter eth (Icelandic)
209	321	d1	Ñ	Latin capital letter n with tilde
210	322	d2	Ò	Latin capital letter o with grave
211	323	d3	Ó	Latin capital letter o with acute
212	324	d4	Ô	Latin capital letter o with circumflex
213	325	d5	Õ	Latin capital letter o with tilde
214	326	d6	Ö	Latin capital letter o with diaresis
215	327	d7	×	Multiplication sign
216	330	d8	Ø	Latin capital letter o with stroke
217	331	d9	Ù	Latin capital letter u with grave
218	332	da	Ú	Latin capital letter u with acute
219	333	db	Û	Latin capital letter u with circumflex
220	334	dc	Ü	Latin capital letter u with diaresis
221	335	dd	Ý	Latin capital letter y with acute
222	336	de		Latin capital letter thorn (Icelandic)
223	337	df	ß	Latin small letter sharp s (German)
224	340	e0	à	Latin small letter a with grave
225	341	e1	á	Latin small letter a with acute
226	342	e2	â	Latin small letter a with circumflex
227	343	e3	ã	Latin small letter a with tilde
228	344	e4	ä	Latin small letter a with diaresis
229	345	e5	å	Latin small letter a with ring above
230	346	e6	æ	Latin small letter ae
231	347	e7	ç	Latin small letter c with cedilla
232	350	e8	è	Latin small letter e with grave
233	351	e9	é	Latin small letter e with acute
234	352	ea	ê	Latin small letter e with circumflex
235	353	eb	ë	Latin small letter e with diaresis
236	354	ec	ì	Latin small letter i with grave
237	355	ed	í	Latin small letter i with acute
238	356	ee	î	Latin small letter i with circumflex
239	357	ef	ï	Latin small letter i with diaresis
240	360	f0		Latin small letter eth (Icelandic)
241	361	f1	ñ	Latin small letter n with tilde
242	362	f2	ò	Latin small letter o with grave
243	363	f3	ó	Latin small letter o with acute

244	364	f4	ô	Latin small letter o with circumflex
245	365	f5	õ	Latin small letter o with tilde
246	366	f6	ö	Latin small letter o with diaresis
247	367	f7	÷	Division sign
248	370	f8	ø	Latin small letter o with stroke
249	371	f9	ù	Latin small letter u with grave
250	372	fa	ú	Latin small letter u with acute
251	373	fb	û	Latin small letter u with circumflex
252	374	fc	ü	Latin small letter u with diaresis
253	375	fd	ý	Latin small letter y with acute
254	376	fe		Latin small letter thorn (Icelandic)
255	377	ff	ÿ	Latin small letter y with diaresis

B.5 Latin-2 (ISO 8859-2)

Der Zeichensatz Latin-2 deckt folgende Sprachen ab: Albanisch, Bosnisch, Deutsch, Englisch, Finnisch, Irisch, Kroatisch, Polnisch, Rumänisch, Serbisch (in lateinischer Transskription), Serbokroatisch, Slowakisch, Slowenisch, Sorbisch, Tschechisch und Ungarisch. Samisch wird in Latin-9 berücksichtigt. Auf:

`http://sizif.mf.uni-lj.si/linux/cee/iso8859-2.html`

finden sich Einzelheiten und weitere URLs. Hier nur die Zeichen, die von Latin-1 abweichen:

dezimal	oktal	hex	Zeichen	Bezeichnung
161	241	a1		Latin capital letter a with ogonek
162	242	a2		Breve
163	243	a3	Ł	Latin capital letter l with stroke
165	245	a5	L’	Latin capital letter l with caron
166	246	a6	Ś	Latin capital letter s with acute
169	251	a9	Š	Latin capital letter s with caron
170	252	aa	Ş	Latin capital letter s with cedilla
171	253	ab	Ť	Latin capital letter t with caron
172	254	ac	Ź	Latin capital letter z with acute
174	256	ae	Ž	Latin capital letter z with caron
175	257	af	Ż	Latin capital letter z with dot above
177	261	b1		Latin small letter a with ogonek
178	262	b2		Ogonek (Schwänzchen)
179	263	b3	ł	Latin small letter l with stroke
181	265	b5		Latin small letter l with caron
182	266	b6		Latin small letter s with acute
183	267	b7		Caron
185	271	b9		Latin small letter s with caron
186	272	ba		Latin small letter s with cedilla

187	273	bb	Latin small letter t with caron
188	274	bc	Latin small letter z with acute
189	275	bd	Double acute accent
190	276	be	Latin small letter z with caron
191	277	bf	Latin small letter z with dot above
192	300	c0	Latin capital letter r with acute
195	303	c3	Latin capital letter a with breve
197	305	c5	Latin capital letter l with acute
198	306	c6	Latin capital letter c with acute
200	310	c8	Latin capital letter c with caron
202	312	ca	Latin capital letter e with ogonek
204	314	cc	Latin capital letter e with caron
207	317	cf	Latin capital letter d with caron
208	320	d0	Latin capital letter d with stroke
209	321	d1	Latin capital letter n with acute
210	322	d2	Latin capital letter n with caron
213	325	d5	Latin capital letter o with double acute
216	330	d8	Latin capital letter r with caron
217	331	d9	Latin capital letter u with ring above
219	333	db	Latin capital letter u with double acute
222	336	de	Latin capital letter t with cedilla
224	340	e0	Latin small letter r with acute
227	343	e3	Latin small letter a with breve
229	345	e5	Latin small letter l with acute
230	346	e6	Latin small letter c with acute
232	350	e8	Latin small letter c with caron
234	352	ea	Latin small letter e with ogonek
236	354	ec	Latin small letter e with caron
239	357	ef	Latin small letter d with caron
240	360	f0	Latin small letter d with stroke
241	361	f1	Latin small letter n with acute
242	362	f2	Latin small letter n with caron
245	365	f5	Latin small letter o with double acute
248	370	f8	Latin small letter r with caron
249	371	f9	Latin small letter u with ring above
251	373	fb	Latin small letter u with double acute
254	376	fe	Latin small letter t with cedilla
255	377	ff	Dot above

B.6 HTML-Entities

HTML-Entities sind eine Ersatzschreibweise für Zeichen, die nicht direkt in HTML-Text eingegeben werden können. Zu diesen Zeichen gehören:

- Sonderzeichen außerhalb des US-ASCII-Zeichensatzes (Umlaute),
- Zeichen, die eine besondere Bedeutung in HTML haben (&, <),

- mathematische und andere Symbole ($\pm$, ©).

Für den Ersatz gibt es zwei Möglichkeiten:

- die dezimale oder hexadezimale Nummer des Zeichens im Zeichensatz,
- eine Umschreibung mit ASCII-Zeichen.

Soweit die Zeichen im Latin-1-Zeichensatz enthalten sind, können die dort angegebenen Nummern verwendet werden. Die vollständige Tabelle entnimmt man am einfachsten der HTML-Spezifikation. Hier nur die häufigsten Zeichen:

dezimal	hex	char ent	Zeichen	Bezeichnung
&	&	&	&	ampersand
<	L	<	$<$	less-than sign
>	N	>	$>$	greater-than sign
				non-breaking space
¡	¡	¡	¡	inverted exclamation mark
£	£	£	£	pound sign
©	©	©	©	copyright sign
«	«	«	$\ll$	left pointing guillemet
				soft or dicretionary hyphen
®	®	®		registered sign
°	°	°	$^\circ$	degree sign
±	±	±	$\pm$	plus-minus sign
²	²	²	^{2}	superscript two
³	³	³	^{3}	superscript three
µ	µ	µ	μ	micro sign
»	»	»	$\gg$	right pointing guillemet
½	½	½	1/2	fraction one half
À	À	À	À	latin capital letter A with grave
Á	Á	Á	Á	latin capital letter A with acute
Â	Â	Â	Â	latin capital letter A with circumflex
Ã	Ã	Ã	Ã	latin capital letter A with tilde
Ä	Ä	Ä	Ä	latin capital letter A with diaresis
Å	Å	Å	Å	latin capital letter A with ring above
Æ	Æ	Æ	Æ	latin capital ligature AE
Ç	Ç	Ç	Ç	latin capital letter C with cedilla
Ñ	Ñ	Ñ	Ñ	latin capital letter N with tilde
Ø	Ø	Ø	Ø	latin capital letter O with stroke
ß	ß	ß	ß	latin small letter sharp s
ë	ë	ë	ë	latin small letter e with diaresis
ñ	ð	ñ	ñ	latin small letter n with tilde

B.7 Sondertasten

Computertastaturen weisen im Vergleich zu den Tastaturen herkömmlicher Schreibmaschinen einige Tasten zusätzlich auf – mit besonderen Bedeutungen. Obwohl die PC-Tastatur (MF-2) weit verbreitet ist, gibt es eine Vielzahl abweichender Tastaturen. Die Wirkung, die von einer Taste ausgelöst wird, hängt in den meisten Fällen von der Software ab, lässt sich daher nicht allgemein angeben und kann von Programm zu Programm variieren. Typische Beispiele sind die Funktionstasten, aber auch fast alle anderen Tasten lassen sich umprogrammieren. In der folgenden Tabelle werden die wichtigsten Sondertasten aufgeführt:

	DÜZ	Datenübertragung zeilenweise
ACK		Acknowledge
AlphaLock		wie Caps Lock
Alt	Alt	Alternate
AltGr	AltGr	Alternate graphic(s)
BackTab		
BS, Backspace	Rücktaste	einen Schritt zurück
BEL		Glocke, ASCII-Nr. 7
Break	Untbr	Signal SIGINT, Signal Nr. 2
CAN		Cancel
Caps Lock	Feststelltaste	Umschaltung feststellen
CBT		Cursor back tab
CE, ClearEntry		
CharDel		Lösche Zeichen
CharInsert		Füge Zeichen vor Cursor ein
ClearSpace		
CR		Carriage return, Wagenrücklauf, ASCII-Nr. 13
Ctrl	Strg	Control, Steuerung
CUB		Cursor back
CUD		Cursor down
CUF		Cursor forward
CUU		Cursor up
DCH		Delete character
DC1		Device control 1, ASCII-Nr. 17
DC2		Device control 2, ASCII-Nr. 18
DC3		Device control 3, ASCII-Nr. 19
DC4		Device control 4, ASCII-Nr. 20
Dead Keys	Tod-Tasten, Akzent-Tasten	T. für diakritische Zeichen, Akzente *ohne* Zeichenvorschub
DEL, Delete	Entf	Zeichen entfernen, löschen
DL		Delete line
DLE		Data link enable
Down	Pfeil abwärts	Cursor abwärts
EM		End of media

End	Ende	Cursor zum Seitenende
ENQ		Enquiry
Enter	Zeilenwechsel	Eingabe vollziehen
EOT		End of transmission
Esc	Escape	Escape, Fluchtsymbol, ASCII-Nr. 27
ETB		End of transaction block
ETX		End of text
ExtChar		
F1	F1	Funktionstaste 1 usw.
FF		Form feed, Blattvorschub, ASCII-Nr. 12
Funct		ähnlich wie Ctrl
Home	Pos1	Cursor zum Seitenanfang
HT		Horizontal tab
INS, Insert	Einfg	Zeichen vor Cursor einfügen
IS1		Information separator 1
IS2		Information separator 2
IS3		Information separator 3
IS4		Information separator 4
Left	Pfeil links	Cursor nach links
LF		Line feed, ASCII-Nr. 10
LineDel		Lösche Zeile
LineErase		Lösche Zeile ab Cursor
LineFeed		wie Down
LineIns		Füge Zeile oberhalb Cursor ein
LocEsc		Local Escape
NAK		Not acknowledge
NoScroll		Hält Bildschirm an
NumLock	Num	Number lock key
Page		Nächste Seite des Speichers
Page Down	Bild abwärts	Nächste Seite des Speichers
PageErase		Seite löschen
Page Up	Bild aufwärts	Vorige Seite des Speichers
Pause	Pause	Pause
PFK		Program Function Key
PrevWin		Previous window
PrintScreen	Druck	Hardcopy des Bildschirms
Reset		Setzt Terminal zurück
Return	Zeilenwechsel	Zeilenwechsel, Eingabe vollziehen
Right	Pfeil rechts	Cursor nach rechts
Rubout		Zeichen löschen, wie DEL
ScrollLock	Rollen	Hält Bildschirm an
Send		Sende Seite zum Computer
Setup		Zeigt Konfiguration an
Shift	Umschaltung	Umschaltung groß-klein
Space	Leertaste	ASCII-Nr. 32, Leerschritt
SOH		Start of header
SUB		Substitute character, ASCII-Nr. 26

SYN		Synch idle
SysRq	S-Abf	System request, System-Anfrage
Tab	Tab	(Horizontal-)Tabulator
Up	Pfeil aufwärts	Cursor nach oben

Daneben finden sich auf den Tastaturen einzelner Hersteller noch Tasten mit angebissenen Äpfeln, fliegenden Fenstern und dergleichen.

C Papier- und Schriftgrößen

C.1 Papierformate

In der EDV sind folgende Papierformate gebräuchlich (Ein Zoll (inch) sind 25,4 mm oder 72 Punkte):

- **DIN A3** Blattgröße 297 mm x 420 mm entsprechend 842 p x 1190 p.
- **DIN A4** Blattgröße 210 mm x 297 mm entsprechend 595 p x 842 p, näherungsweise 8 Zoll x 12 Zoll.
- **Briefumschlag DIN lang** 110 mm x 220 mm
- **Tabellierpapier schmal** Blattgröße einschließlich Lochrand 240 mm x 305 mm, ohne Lochrand 210 mm x 305 mm, also annähernd DIN A4. 305 mm sind 12 Zoll.
- **Tabellierpapier breit** Blattgröße einschließlich Lochrand 375 mm x 305 mm.

  Bei 10 Zeichen/Zoll faßt die Zeile 132 Zeichen, bei 12 Zeichen/Zoll 158 Zeichen, bei 15 Zeichen/Zoll 198 Zeichen.
- **Executive** amerikanisch, Blattgröße 7.25 Zoll x 10.5 Zoll (184,2 mm x 266,7 mm) entsprechend 540 p x 720 p.
- **Ledger** amerikanisch, Blattgröße 17 Zoll x 11 Zoll (431,8 mm x 279,4 mm) entsprechend 1224 p x 792 p, Querformat.
- **Legal** amerikanisch, Blattgröße 8.5 Zoll x 14 Zoll (215,9 mm x 355,6 mm) entsprechend 612 p x 1008 p.
- **Letter** amerikanisch, Blattgröße 8.5 Zoll x 11 Zoll (215,9 mm x 279,4 mm) entsprechend 612 p x 792 p, Schreibmaschinenpapier.
- **Tabloid** amerikanisch, Blattgröße 11 Zoll x 17 Zoll (279,4 mm x 431,8 mm) entsprechend 792 p x 1224 p.

Es ist zwischen der Blattgröße, der Größe des adressierbaren Bereichs (logische Seite) und der Größe des beschreibbaren Bereichs zu unterscheiden. Näheres beispielsweise im *HP PCL 5 Printer Language Technical Reference Manual* (das leider nicht zum Lieferumfang der HP Laserdrucker gehört, für die System-Manager aber äußerst wichtig ist).

C.2 Schriftgrößen

Bei den Schriftgrößen oder -graden finden sich mehrere Maßsysteme. In Mitteleuropa ist traditionell das aus dem Bleisatz stammende Punktsystem nach François Ambroise Didot gebräuchlich, das die Höhe der Schrifttype (nicht des Buchstabens) angibt. Es geht auf den französischen Fuß zurück; ein Punkt war früher 0,376 mm, heute ist er in Angleichung an das metrische System 0,375 mm. Die wichtigsten Größen werden auch mit eigenen Namen bezeichnet (12 pt = Cicero). Die untere Grenze der Lesbarkeit liegt bei 9 pt, optimal sind 12 pt.

Auf englische Füße geht das Pica-System zurück, in dem 1 Pica gleich 12 Points oder 4,216 mm (1/6 Zoll) ist. Hiervon abgeleitet ist das IBM-Pica-System, in dem Pica eine Schrift mit 6 Zeilen pro Zoll und 10 Zeichen pro Zoll ist. Elite bedeutet 12 Zeichen pro Zoll bei gleichem Zeilenabstand.

Auf deutschen Schreibmaschinen war Pica eine Schriftart mit 2,6 mm Schrittweite (etwa 10 Zeichen pro Zoll) und Perl eine Schriftart mit 2,3 mm Schrittweite.

D Farben

D.1 RGB-Farbwerte

D.2 X11-Farben

Eine Auswahl der von X11 verwendeten Farben. Die vollständige Liste ist in `/etc/X11/rgb.txt`, `/usr/X11R6/lib/X11/rgb.txt` oder `/usr/lib/x11/rgb.txt` zu finden. Mittels `showrgb` werden die Farben mit ihren Zahlenwerten (RGB-Tripletts) aufgelistet. Das Kommando `xcolors` stellt in einem großen Fenster alle Farben gleichzeitig samt ihren Bezeichnungen dar, das Kommando `xcolorsel` in einem kleinen Fenster einen Ausschnitt samt Hexwerten und Bezeichnungen. Die tatsächliche Farbdarstellung hängt vom Bildschirm bzw. Drucker ab, deshalb sind die Farbmuster nur als Orientierungshilfe zu verstehen.

X11 stellt auch Farben dar, die nicht in dieser Liste genannt sind. Darüber hinaus kann sich ein Benutzer mit einem Farbeditor eigene Farben für X11 mischen. Nach dem RGB-Modell arbeiten `xcolmix` (Debian) und `xcoloredit`, zu holen bei `http://freebsd.home4u.ch/de/` oder `ftp://export.lcs.mit.edu/R5contrib/`. Ein durchschnittlicher Benutzer kommt jedoch mit den vorgefertigten Farben aus.

Farbe	Colour	RGB-Vektor	Hex, 8bittig
weiß	white	255,255,255	#ffffff
gelb	yellow	255,255,0	#ffff00
fuchsienrot	magenta1	255,0,255	#ff00ff
cyan	cyan1	0,255,255	#00ffff
rot	red	255,0,0	#ff0000
maigrün	green	0,255,0	#00ff00
blau	blue1	0,0,255	#0000ff
grau	grey52	133,133,133	#868686
olivgrün	olive drab	107,142,35	#698e20
purpur	purple	160,32,240	#9e20ef
dunkelgrün	green4	0,139,0	#008a00
kastanienbraun	maroon	176,48,96	#ae3061
grün	green	0,255,0	#00ff00
marineblau	navy blue	0,0,128	#000086
lachsrot	salmon	250,128,114	#f78271
dunkelrosa	hot pink	255,105,180	#ff69b6
dunkelorange	dark orange	255,140,0	#ff8e00
königsblau	royal blue	65,105,225	#4169df
altweiß	antique white	250,235,215	#f7ebd7

ocker	navajo white	255,222,173	#ffdfae
hellgrau	light grey	211,211,211	#cfd3cf
schwarz	black	0,0,0	#000000

D.3 HTML-Farben

Eine Auswahl von Farben nach dem RGB-Farbmodell, ihren Bezeichnungen und ihren Hex-Codes, wie sie in HTML-Seiten verwendet werden. Die ersten 16 Farben können auf Webseiten über ihre englische Bezeichnung angesprochen werden; die Hex-Codes sollten immer richtig verstanden werden. Weiteres unter `http:\\www.htmlhelp.com/cgi-bin/color.cgi` oder `http://www.seifert-web.de/info/bgclick.htm`.

Farbe	Colour	Hex Code
weiß	white	#ffffff
schwarz	black	#000000
gelb	yellow	#ffff00
fuchsienrot	magenta, fuchsia	#ff00ff
cyan	cyan, aqua	#00ffff
rot	red	#ff0000
maigrün	lime	#00ff00
blau	blue	#0000ff
grau	gray	#808080
olivgrün	olive	#808000
purpur	purple	#800080
dunkelgrün	teal	#008080
kastanienbraun	maroon	#800000
grün	green	#008000
marineblau	navy	#000080
silbern	silver	#c0c0c0
lachsrot	salmon	#fa8072
dunkelrosa	hot pink	#ff69b4
dunkel orange	dark orange	#ff8c00
königsblau	royal blue	#4169e1
ocker	navajo white	#ffdead

E Die wichtigsten Linux/UNIX-Kommandos

Einzelheiten siehe Online-Referenz-Handbuch. Das wichtigste Kommando zuerst, die übrigen nach Sachgebiet und dann alphabetisch geordnet:

`man man`	Beschreibung zum Kommando `man(1)` ausgeben
`man -k time`	man-Seiten zum Stichwort `time` auflisten
`apropos time`	man-Seiten zum Stichwort `time` auflisten

Allgemeines

`alias`	Alias in der Shell einrichten
	`alias r='fc -e -'`
`at`	Programm zu einem beliebigen Zeitpunkt starten
	`at 0815 Jan 24`
	`program`
	`EOF` (meist `control-d`)
`bdf, df, du`	Plattenbelegung ermitteln
	`df`
	`du .`
`calendar`	Terminverwaltung (Reminder Service)
	(Datei `$HOME/calendar` muß existieren)
	`calendar`
`crontab`	Tabelle für `cron` erzeugen
	`crontab crontabfile`
`date`	Datum und Zeit anzeigen
	`date`
`echo, print`	Argument auf `stdout` schreiben
	`echo 'Hallo, wie gehts?'`
`exit`	Shell beenden
	`exit`
`kill`	Signal an Prozess senden
	`kill process_id`
	`kill -s SIGHUP process_id`
`leave`	an Feierabend erinnern
	`leave 2215`
`lock`	Terminal sperren
	`lock`
`newgrp`	Benutzergruppe wechseln
	`newgrp student`
`nice`	Priorität eines Programmes herabsetzen
	`nice program`

`nohup`	Programm von Sitzung abkoppeln `nohup program &`
`passwd`	Passwort ändern `passwd` `yppasswd` (in NIS-Clustern)
`ps`	laufende Prozesse anzeigen `ps -ef`
`script`	Sitzung mitschreiben `script` (beenden mit exit)
`set`	Umgebung anzeigen `set`
`sh, ksh, bash`	Shells (Bourne, Korn, Bash) `bash`
`stty`	Terminal-Schnittstelle anzeigen `stty`
`su`	Usernamen wechseln (substituieren) `su bjalex1`
`tset, reset`	Terminal initialisieren `tset vt100`
`tty`	Terminalnamen (`/dev/tty*`) anzeigen `tty`
`who`	eingeloggte Benutzer auflisten `who -H`
`whoami, id`	meinen Namen anzeigen `id`
`xargs`	Argumentliste aufbauen und Kommando ausführen `ls | xargs -i -t mv {} subdir/{}`

Dateien, Verzeichnisse

`cd`	Arbeitsverzeichnis wechseln
	`cd`
	`cd /usr/local/bin`

`chgrp`	Gruppe einer Datei wechseln `chgrp students file`
`chmod`	Zugriffsrechte einel Datei ändern `chmod 755 file`
`chown`	Besitzer einer Datei wechseln `chown aralex1 file`
`cmp, diff`	zwei Dateien vergleichen `cmp file1 file2`

`compress`	Datei komprimieren
	`compress file`
	`uncompress file.Z`

`cp`	Datei kopieren `cp original kopie`
`file`	Dateityp ermitteln

	`file file`
`find, whereis`	Dateien suchen
	`find . -name file -print`
`gzip`	Datei komprimieren (GNU)
	`gzip file`
	`gunzip file.gz`
`ln`	Datei linken
	`ln file hardlinkname`
	`ln -s file softlinkname`
`ls`	Verzeichnisse auflisten
	`ls -al`
	`ls -l /usr/local`
`mkdir`	Verzeichnis anlegen
	`mkdir newdir`
`mv`	Datei verschieben oder umbenennen
	`mv oldfilename newfilename`
`od`	(oktalen) Dump einer Datei ausgeben
	`od -c file`
`pwd`	Arbeitsverzeichnis anzeigen
	`pwd`
`rm, rmdir`	Datei oder leeres Verzeichnis löschen
	`rm file`
	`rm -r dir` (rekursiv)
	`rmdir dir`
`tar`	Datei-Archiv schreiben oder lesen
	`tar -cf /dev/st0 ./dir &`
`touch`	leere Datei erzeugen, Zeitstempel ändern
	`touch file`

Kommunikation, Netz

`ftp`	Datei Transfer
	`ftp ftp.ciw.uni-karlsruhe.de`
`hostname`	Hostnamen anzeigen
	`hostname`
`irc`	Netzgeschwätz
	`irc` (beenden mit `/quit`)
`kermit`	Datei übertragen, auch von/zu Nicht-UNIX-Anlagen
	`kermit` (beenden mit `exit`)
`mail,elm,`	Mail lesen und versenden
`mutt`	`mail wulf.alex@mvm.uni-karlsruhe.de < file`
	`elm`
	`mutt`
`netscape`	WWW-Browser (einer unter vielen)
	`netscape &`
`news`	Neuigkeiten anzeigen
	`news -a`

`nslookup`	Auskunft über Host
	`nslookup mvmpc100.ciw.uni-kalrsuhe.de`
	`nslookup 129.13.118.100`

`ping`	Verbindung prüfen `ping 129.13.118.100`
`ssh`	verschlüsselte Verbindung zu Host im Netz `ssh mvmpc100.ciw.uni-karlsruhe.de`
`rlogin`	Dialog mit UNIX-Host im Netz (unverschlüsselt) `rlogin mvmpc100`
`telnet`	Dialog mit Host im Netz (unverschlüsselt) `telent mvmpc100`
`tin, xn`	Newsreader `rtin`
`uucp`	Programmpaket mit UNIX-Netzdiensten
`whois`	Auskunft über Netzknoten `whois -h whois.internic.net gatekeeper.dec.com`
`write, talk`	Dialog mit eingeloggtem Benutzer `talk wualex1@mvmpc100`

Programmieren

`ar`	Gruppe von files archivieren `ar -r archiv.a file1 file2`
`cb`	C-Beautifier, Quelle verschönern `cb prog.c > prog.b`
`cc`	C-Compiler mit Linker `cc -o prog prog.c`
`ci`	einchecken in das RCS-System `ci mysource.c`
`co`	auschecken aus dem RCS-System `co -l mysource.c`
`lint`	C-Syntax-Prüfer `lint prog.c`
`make`	Compileraufruf vereinfachen `make` (Makefile erforderlich)
`sdb, xdb`	symbolischer Debugger `xdb` (einige Dateien erforderlich)

Textverarbeitung

`adjust`	Text formatieren (einfachst) `adjust -j -m60 textfile`
`awk`	Listengenerator `awk -f awkscript textfile` (awk-Script erforderlich)
`cancel`	Druckauftrag löschen `cancel lp-4711`
`cat`	von `stdin` lesen, nach `stdout` schreiben

	`cat textfile`
	`cat file1 file2 > file.all`
	`cat textfile`
`cut`	Spalten aus Tabellen auswählen
	`cut -f1 tablefile > newfile`
`ed`	Zeileneditor, mit `diff(1)` nützlich
	`ed textfile`
`emacs`	Editor, alternativ zum `vi(1)` (GNU)
	`emacs textfile`
`expand`	Tabs ins Spaces umwandeln
	`expand textfile > newfile`
`grep, fgrep`	Muster in Dateien suchen
	`grep -i Unix textfile`
	`fgrep UNIX textfile`
`head, tail`	Anfang bzw. Ende eines Textfiles anzeigen
	`head textfile`
`lp, lpr`	Datei über Spooler ausdrucken
	`lp -dlp2 textfile`
	`lpr -Plp2 textfile`
`lpstat, lpq`	Spoolerstatus anzeigen
	`lpstat -t`
`more, less`	Textfile schirmweise anzeigen
	`more textfile`
	`ls -l | more`
`nroff`	Textformatierer
	`nroff nrofffile | lp`
`recode`	Filter zur Umwandlung von Zeichensätzen (GNU)
	`recode --help`
	`recode -l`
	`recode -v ascii-bs:EBCDIC-IBM textfile`
`sed`	filternder Editor
	`sed 's/[A-Z]/[a-z]/g' textfile > newfile`
`sort`	Textfile zeilenweise sortieren
	`sort liste | uniq > newfile`
`spell`	Rechtschreibung prüfen
	`spell` spelling textfile+
`tee`	`stdout` zugleich in eine Datei schreiben
	`who | tee whofile`
`tr`	Zeichen in Textfile ersetzen
	`tr -d "\015" < textfile1 > textfile2`
`uniq`	sortiertes Textfile nach doppelten Zeilen durchsuchen
	`sort liste | uniq > newfile`
`vi`	Bildschirm-Editor, alternativ zum `emacs(1)`
	`vi textfile`
`view`	`vi(1)` nur zum Lesen aufrufen
	`view textfile`
`vis`	Dateien mit Steuersequenzen anzeigen

		`vis textfile`
`wc`	Zeichen, Wörter und Zeilen zählen	
		`wc textfile`

F Besondere Linux/UNIX-Kommandos

F.1 vi(1)

Kommando	Wirkung
esc	schaltet in Kommando-Modus um
h	Cursor nach links
j	Cursor nach unten
k	Cursor nach oben
l	Cursor nach rechts
0	Cursor an Zeilenanfang
$	Cursor an Zeilenende
nG	Cursor in Zeile Nr. n
G	Cursor in letzte Zeile des Textes
+n	Cursor n Zeilen vorwärts
-n	Cursor n Zeilen rückwärts
a	schreibe anschließend an Cursor
i	schreibe vor Cursor
o	öffne neue Zeile unterhalb Cursor
O	öffne neue Zeile oberhalb Cursor
r	ersetze das Zeichen auf Cursor
R	ersetze Text ab Cursor
x	lösche das Zeichen auf Cursor
dd	lösche Cursorzeile
ndd	lösche n Zeilen ab und samt Cursorzeile
nY	übernimm n Zeilen in Zwischenspeicher
p	schreibe Zwischenpuffer unterhalb Cursorzeile
P	schreibe Zwischenpuffer oberhalb Cursorzeile
J	hänge nächste Zeile an laufende an
/abc	suche String `abc` nach Cursor
?abc	suche String `abc` vor Cursor
n	wiederhole Stringsuche
u	mache letztes Kommando ungültig (undo)
%	suche Gegenklammer (in Programmen)
:read file	lies `file` ein
:w	schreibe Text zurück (write, save)
:q	verlasse Editor ohne write (quit)
:wq	write und quit

Weitere `vi`-Kommandos im Referenz-Handbuch unter `vi(1)`, in verschiedenen Hilfen im Netz oder in dem Buch von MORRIS I. BOLSKY.

F.2 emacs(1)

Der `emacs(1)` kennt keine Modi, die Kommandos werden durch besondere Tastenkombinationen erzeugt. `control h, t` bedeutet: zuerst die Tastenkombination `control h` eintippen, loslassen, dann `t` eintippen. Die Mächtigkeit des Emacs zeigt sich daran, daß nicht ein Hilfefenster angeboten wird, sondern ein ganzes Tutorial.

Kommando	Wirkung
emacs filename	Aufruf zum Editieren des Files `filename`
control h, t	Tutorial
control h, i	Manuals
control x, control s	Text speichern
control x, control x	Text speichern, Editor verlassen

Weitere `emacs`-Kommandos im Tutorial, im Referenz-Handbuch unter `emacs(1)`, in verschiedenen Hilfen im Netz oder in dem Buch von D. CAMERON und B. ROSENBLATT.

F.3 joe(1)

Der `joe(1)` kennt keine Modi, die Kommandos werden durch besondere Tastenkombinationen erzeugt. `control k, h` bedeutet: zuerst die Tastenkombination `control k` eintippen, loslassen, dann `h` eintippen.

Kommando	Wirkung
joe filename	Aufruf zum Editieren des Files `filename`
control k, h	Hilfe anzeigen
control k, x	Text speichern, Editor verlassen
control c	Editor ohne Speichern verlassen

Weitere `joe`-Kommandos im Hilfefenster, im Referenz-Handbuch unter `joe(1)` oder in verschiedenen Hilfen im Netz.

F.4 Drucken

In der UNIX-Welt gibt es mehrere Systeme aus Druckdämonen und -kommandos. Man muß sich beim System-Manager erkundigen (oder experimentell ermitteln), welches Drucksystem verwendet wird. In ihrer Funktionalität ähneln sich die Systeme, in Einzelheiten und der Kommandosyntax unterscheiden sie sich.

Aufgabe	System V	BSD	LPRng	HPDPS

Druckauftrag geben	lp	lpr	lpr	pdpr
Druckauftrag löschen	cancel	lprm	lprm	pdrm, pdclean
Status abfragen	lpstat	lpq	lpq	pdq
Spoolerdämon	lpsched	lp	lp	(Spooler)

G Zugriffsrechte (`ls -l`)

- `d` Verzeichnis (directory),
- `l` weicher (symbolischer) Link,
- `c` Character Device File,
- `b` Block Device File,
- `p` Named Pipe (FIFO),
- `n` Network Device File,
- `s` an vorderster Stelle: Socket,
- `-` an vorderster Stelle: reguläre Datei, sonst gleichbedeutend mit nicht gesetztem Recht, Zahlenwert 0,
- `r` lesen (read), kopieren, bei Verzeichnissen: mit `ls(1)` auflisten, Zahlenwert 4,
- `w` schreiben (write), ändern, bei Verzeichnissen Dateien anlegen oder löschen, Zahlenwert 2,
- `x` ausführen (execute, was nur bei Programmen sinnvoll ist), bei Verzeichnissen mit `cd(1)` hinein- oder hindurchgehen (search), Zahlenwert 1,
- `S` bei den Besitzerrechten: Set-User-ID-Bit gesetzt, Zahlenwert 4, `x` nicht gesetzt,
- `s` bei den Besitzerrechten: Set-User-ID-Bit gesetzt, Zahlenwert 4, `x` gesetzt,
- `S` bei den Gruppenrechten: Set-Group-ID-Bit gesetzt, Zahlenwert 2, `x` nicht gesetzt,
- `s` bei den Gruppenrechten: Set-Group-ID-Bit gesetzt, Zahlenwert 2, `x` gesetzt,
- `T` Sticky Bit gesetzt, Zahlenwert 1, `x` nicht gesetzt,
- `t` Sticky Bit, Zahlenwert 1, und `x` gesetzt,

Tab. G.1: Zugriffsrechte von Dateien und Verzeichnissen, Beispiele

Rechte	Rechte	Besitzer	Gruppe	Rest der Welt
000	---------	Rechte ändern	nichts	nichts
700	rwx------	alles	nichts	nichts
600	rw-------	lesen + schreiben	nichts	nichts
500	r-x------	lesen + ausführen	nichts	nichts
400	r--------	lesen	nichts	nichts
300	-wx------	schreiben + ausführen	nichts	nichts
200	-w-------	schreiben	nichts	nichts
100	--x------	ausführen	nichts	nichts
750	rwxr-x---	alles	lesen + ausführen	nichts
640	rw-------	lesen + schreiben	lesen	nichts
755	rwxr-xr-x	alles	lesen + ausführen	lesen + ausführen
644	rw-r--r--	lesen + schreiben	lesen	lesen
664	rw-rw-r--	lesen + schreiben	lesen + schreiben	lesen
775	rwxrwxr-x	alles	alles	lesen + ausführen
777	rwxrwxrwx	alles	alles	alles
1000	--------T	nichts	nichts	nichts, Sticky Bit
2000	-----S---	nichts	nichts, sgid-Bit	nichts
4000	--S------	nichts, suid-Bit	nichts	nichts
4600	rwS------	lesen + schreiben, suid-Bit	nichts	nichts
4700	rws------	alles, suid-Bit	nichts	nichts
6750	rwsr-x---	alles, suid-Bit	lesen + ausführen, sgid-Bit	nichts
4555	r-sr-xr-x	lesen + ausführen, suid-Bit	lesen + ausführen	lesen + ausführen
7555	r-sr-sr-t	lesen + ausführen, suid-Bit	lesen + ausführen, sgid-Bit	lesen + ausführen, Sticky Bit

H Linux/UNIX-Signale

Die Default-Reaktion eines Prozesses auf die meisten Signale ist seine Beendigung; sie können aber abgefangen und umdefiniert werden. Die Signale 09, 24 und 26 können nicht abgefangen werden. Die Bezeichnungen sind nicht ganz einheitlich. Weiteres unter `signal(2)`, `signal(5)` oder `signal(7)`.

Name	Nr.	Bedeutung
SIGHUP	01	hangup
SIGINT	02	interrupt (meist Break-Taste)
SIGQUIT	03	quit
SIGILL	04	illegal instruction
SIGTRAP	05	trace trap
SIGABRT	06	software generated abort
SIGIOT	06	software generated signal
SIGEMT	07	software generated signal
SIGFPE	08	floating point exception
SIGKILL	09	kill (sofortiger Selbstmord)
SIGBUS	10	bus error
SIGSEGV	11	segmentation violation
SIGSYS	12	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination (bitte Schluß machen)
SIGUSR1	16	user defined signal 1
SIGSTKFLT	16	stack fault on coprocessor
SIGUSR2	17	user defined signal 2
SIGCHLD	18	death of a child
SIGCLD	18	= SIGCHLD
SIGPWR	19	power fail
SIGINFO	19	= SIGPWR
SIGVTALRM	20	virtual timer alarm
SIGPROF	21	profiling timer alarm
SIGIO	22	asynchronous I/O signal
SIGPOLL	22	= SIGIO
SIGWINDOW	23	window change or mouse signal
SIGSTOP	24	stop
SIGTSTP	25	stop signal generated from keyboard
SIGCONT	26	continue after stop
SIGTTIN	27	background read attempt from control terminal
SIGTTOU	28	background write attempt from control terminal
SIGURG	29	urgent data arrived on an I/O channel
SIGLOST	30	NFS file lock lost

SIGXCPU	30	CPU time limit exceeded
SIGXFSZ	31	file size limit exceeded

Die Nr. 0 wird nicht für ein Signal verwendet. Wird sie dem Shell-Kommando `trap` übergeben, so ist damit das Ende der Shell (`exit`) gemeint.

I Beispiele LaTeX

I.1 Textbeispiel

```
% (Kommentar) Erste Hilfe zum Arbeiten mit LaTeX
%
% Das Prozentzeichen leitet Kommentar ein, gilt
% bis Zeilenende. Um ein Prozentzeichen in den Text
% einzugeben, muss man folglich einen Backslash
% davor setzen. Dasselbe gilt fuer Dollar, Ampersand,
% Doppelkreuz, Unterstrich und geschweifte
% Klammern. Diese Zeichen kommen in Befehlen vor.
%
% Format: LaTeX

\NeedsTeXFormat{LaTeX2e}

% Die meisten Befehle beginnen mit einem Backslash.
%
% Es gibt die Klassen Buch, Bericht (Report), Artikel,
% Brief, Folien und weitere. Bericht und Artikel
% sind die wichtigsten.
% Je nach Font und Schriftgroesse Zeilen auskommentieren!
%
% 12 Punkte, doppelseitig, Times Roman (wie Buch)
% \documentclass[12pt,twoside,a4paper]{article}
% \usepackage{german}
% \unitlength1.0mm
%
% 12 Punkte, einseitig, New Century Schoolbook
\documentclass[12pt,a4paper]{article}
\usepackage{german,newcent,verbatim}
\unitlength1.0mm

% Der Befehl \sloppy weist LaTeX an, mit den
% Wortabstaenden nicht zu kleinlich zu sein.

\sloppy

% Trennhilfe (Liste von Ausnahmen)

\input{hyphen}
```

```
% Satzspiegel vergroessern:

\topmargin-24mm
\textwidth180mm
\textheight240mm

% zusaetzlicher Seitenrand fuer beidseitigen Druck:

\oddsidemargin-10mm
\evensidemargin-10mm

% Jetzt kommt der erste Text: Titel, Autor und Datum

\title{Erste Hilfe beim Arbeiten mit LaTeX}
\author{W. Alex}
\date{\today}

% Hier beginnt das eigentliche Dokument

\begin{document}

% Befehl zum Erzeugen des Titels

\maketitle

% Nun kommt der erste Abschnitt (section).
% Die Ueberschrift steht in geschweiften Klammern.
% Es gibt auch Unterabschnitte.

\section{Zweck}

LATeX ist ein {\em Satzsystem}. Mit seiner Hilfe
lassen sich Texte zur Ausgabe auf Papier formatieren,
vorzugsweise zur Ausgabe mittels eines Laserdruckers.
Es gibt Hilfsprogramme zur Anfertigung von Zeichnungen.
Ferner ist es m"oglich, beliebige Postscript-Abbildungen
einzubinden, also auch Fotos.

LaTeX ist eine Makro-Sammlung von {\sc Leslie Lamport},
die auf dem TeX-System von {\sc Donald Knuth} aufbaut.
TeX und LaTex sind frei verf"ugbar, in Deutschland zum
Beispiel bei der Universit"at Heidelberg (Dante e. V.).

\section{Aufbau des Textfiles}

Schreiben Sie Ihr Textfile mit einem beliebigen Editor,
beispielsweise mit dem \verb+vi+, \verb+emacs+ oder
```

```
\verb+joe+. Ein Textfile hat folgenden Aufbau:

% Hier ein woertlich auszugebender Text

\begin{verbatim}
% Kommentar (optional)
\documentclass[12pt,a4paper]{article}
\usepackage{german,verbatim}
\begin{document}
Text
\end{document}
\end{verbatim}

Schreiben Sie den Text ohne Worttrennungen und ohne
Numerierungen. Zeilenwechsel haben keine Bedeutung
f"ur LaTeX, bevorzugen Sie kurze Zeilen. Abs"atze
werden durch Leerzeilen markiert.

Die deutschen Umlaute werden als G"ansef"u\3chen mit
folgendem Grundlaut geschrieben, das \3 als Backslash
mit folgender Ziffer 3. Es gibt weitere M"oglichkeiten,
auch f"ur die Sonderzeichen anderer Sprachen.

\section{Mathematische Formeln}

Eine St"arke von LaTeX ist die Darstellung
mathematischer Formeln. Hier drei nicht ganz
einfache Beispiele:

\boldmath

\begin{equation}
c = \sqrt{a^{2} + {b}^2}
\end{equation}

\begin{equation}
\oint \vec E\:d\vec s = - \frac {\,\partial}{\partial t}
\int \vec B \:d\vec A
\end{equation}

\begin{equation}
\frac{1}{2 \pi j} \int\limits_{x-j\infty}^{x+j\infty}
e^{ts} \:f(s)\:ds =
\left\{\begin{array}{r@{\quad \mbox{f"ur} \quad}l}
0 & t < 0\\
F(t) & t > 0
\end{array} \right.
```

```
\end{equation}

\unboldmath

\section{Schriftgr"o\3en}

Ausgehend von der Grundgr"o\3e lassen sich f"ur
einzelne Zeilen oder Abschnitte die Schriftgr"o\3en
verkleinern oder vergr"o\3ern:

{\tiny Diese Schrift ist winzig (tiny).}

{\scriptsize Diese ist sehr klein (scriptsize).}

{\footnotesize F"ur Fu\3noten (footnotesize).}

{\small Diese Schrift ist klein (small).}

{\normalsize Diese Schrift ist normal.}

{\large Jetzt wirds gr"o\3er (large).}

{\Large Jetzt wirds noch gr"o\3er (Large).}

{\LARGE Noch gr"o\3er (LARGE).}

{\huge Riesige Schrift (huge).}

{\Huge Gigantisch (Huge).} \\

\noindent
Formeln lassen sich nicht vergr"o\3ern oder
verkleinern. Zur"uck zur Normalgr"o\3e. Diese
betr"agt auf DIN A4 am besten 12 Punkte.
10 oder 11 Punkte sind auch m"oglich, aber schon
etwas unbequem zu lesen. Weiterhin kann man "uber
die Schriftart die Lesbarkeit beeinflussen.
Standard ist die Times Roman, eine gut lesbare,
platzsparende Schrift. Soll es etwas deutlicher
sein, so ist die New Century Schoolbook zu
empfehlen, die rund 10 \% mehr Platz beansprucht.

\section{Arbeitsg"ange}

Die Reihenfolge der einzelnen Schritte ist folgende:

% Aufzaehlungen sind kein Problem, wie man sieht.
```

```
\begin{itemize}
\item mit einem Editor das File \verb+abc.tex+ schreiben,
\item mit \verb+latex abc+ das File formatieren,
\item mit \verb+latex abc+ Referenzen einsetzen,
\item mit \verb+xdvi abc+ den formatierten Text anschauen,
\item mit \verb+dvips -o abc.ps abc+ das Postscript-File
      erzeugen,
\item mit \verb+lp -dlp4 abc.ps+ das Postscript-File
      zum Drucker schicken.
\end{itemize}
LaTeX erzeugt dabei einige Zwischen- und Hilfsfiles.

\section{Literatur}

\begin{enumerate}
\item L. Lamport, LaTeX: A Document Preparation System\\
Addison-Wesley, 1986
\item H. Kopka, LaTeX - Eine Einf"uhrung\\
Addison-Wesley, 1988
\item H. Partl, E. Schlegl, I. Hyna:
      LaTeX-Kurz\-be\-schrei\-bung\\
      EDV-Zentrum der TU Wien
\end{enumerate}

% Nun kommt das Ende des Dokuments:

\end{document}
```

I.2 Gelatexte Formeln

$$c = \sqrt{a^2 + b^2} \tag{I.1}$$

$$\sqrt[3]{1+x} \approx 1 + \frac{x}{3} \qquad \text{für} \quad x \ll 1 \tag{I.2}$$

$$r = \sqrt[3]{\frac{3}{4\pi} V} \tag{I.3}$$

$$\lim_{x \to 0} \frac{\sin x}{x} = 1 \tag{I.4}$$

$$a = \frac{F_0}{k} \frac{1}{\sqrt{(1 - \frac{\Omega^2}{\omega_0^2})^2 + (\frac{\Omega c}{k})^2}} \tag{I.5}$$

$$m\ddot{x} + c\dot{x} + kx = \sum_k F_k \cos \Omega_k t \tag{I.6}$$

$$\Theta \frac{d^2\varphi}{dt^2} + k^* \frac{d\varphi}{dt} + D^* \varphi = |\vec{D}| \tag{I.7}$$

$$\bar{Y} \approx f(\bar{x}) + \frac{1}{2} \frac{N-1}{N} f''(\bar{x})\, s_x^2 \tag{I.8}$$

$$\vec{F_\Gamma} = -\frac{\Gamma m M}{r^2} \vec{e_r} = -\frac{\Gamma m M}{r^3} \vec{r} \tag{I.9}$$

$$\begin{aligned} \text{Arbeit} &= \lim_{\Delta r_i \to 0} \sum \vec{F_i} \Delta \vec{r_i} \\ &= \int\limits_{\vec{r}_0}^{\vec{r}(t)} \vec{F}(\vec{r})\, d\vec{r} \end{aligned} \tag{I.10}$$

$$\prod_{j \geq 0} \left(\sum_{k \geq 0} a_{jk} z^k \right) = \sum_{n \geq 0} z^n \left(\sum_{\substack{h_0, k_1 \ldots \geq 0 \\ k_0 + k_1 + \cdots = 0}} a_{0k_o} a_{1k_1} \ldots \right) \tag{I.11}$$

$$\vec{x} = \begin{pmatrix} x - x_s \\ y - y_s \\ z - z_s \end{pmatrix} \tag{I.12}$$

$$\mathbf{\Psi} = \begin{pmatrix} \binom{ab}{cd} & \frac{e+f}{g-h} \\ \Re z & \left| \begin{matrix} ij \\ kl \end{matrix} \right| \end{pmatrix} \tag{I.13}$$ [1]

$$dE_\omega = V \frac{\hbar}{\pi^2 c^3} \omega^3 \cdot e^{-\frac{\hbar \omega}{T}} \cdot d\omega \tag{I.14}$$

$$\oint \vec{E}\, d\vec{s} = -\frac{\partial}{\partial t} \int \vec{B}\, d\vec{A} \tag{I.15}$$

$$\frac{1}{2\pi j} \int\limits_{x-j\infty}^{x+j\infty} e^{ts} f(s)\, ds = \left\{ \begin{array}{rcl} 0 & \text{für} & t < 0 \\ F(t) & \text{für} & t > 0 \end{array} \right. \tag{I.16}$$

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1} \tag{I.17}$$

$$\boxed{\forall x \in \mathrm{R}: \qquad x^2 \geq 0} \tag{I.18}$$

[1]Fette Griechen gibt es nur als Großbuchstaben. Die Erzeugung dieser Fußnote war übrigens nicht einfach.

$$\forall x, y, z \in \mathrm{M}: \qquad (xRy \wedge xRz) \Rightarrow y = z \tag{I.19}$$

$$A \cdot B = \overline{\overline{A} + \overline{B}} \tag{I.20}$$

$$\rho \cdot \bar{v}_k \cdot \frac{\partial \bar{v}_j}{\partial x_k} = -\frac{\partial \bar{p}}{\partial x_j} + \frac{\partial}{\partial x_k}\left(\mu \frac{\partial \bar{v}_j}{\partial x_k} - \rho\, \overline{v_k' v_j'}\right) \tag{I.21}$$

$$r' = \frac{\overline{v_1' v_2'}}{\sqrt{\overline{v_1'^2}}\,\sqrt{\overline{v_2'^2}}} \tag{I.22}$$

$$\tau_{tur} = \rho l^2 \, |\frac{\partial \bar{v}_1}{\partial x_2}| \, \frac{\partial \bar{v}_1}{\partial x_2} \tag{I.23}$$

$$\begin{aligned} \ddot{R} &= \frac{1}{\Psi}(V_r - \dot{R}) + R\dot{\Phi}^2 \\ \ddot{\Phi} &= [\frac{1}{\Psi}(V_\varphi - R\dot{\Phi}) - 2\dot{R}\dot{\Phi}]\frac{1}{R} \\ \ddot{Z} &= \frac{1}{\Psi}(V_Z - \dot{Z}) \end{aligned} \tag{I.24}$$

$$\hat{\chi}^2 = \frac{n(n-1)}{B(n-B)} \sum_{i=1}^{k} \frac{(B_i - E_i)^2}{n_i} > \chi^2_{k-1;\alpha} \tag{I.25}$$

$$V(r, \vartheta, \varphi) = \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} q_{l,m} \frac{Y_{l,m}\vartheta, \varphi)}{r^{l+1}} \tag{I.26}$$

$$\vec{q} = \begin{pmatrix} q_{0,0} \\ q_{1,1} \\ q_{1,0} \\ q_{1,-1} \\ q_{2,2} \\ q_{2,1} \\ q_{2,0} \\ q_{2,-1} \\ q_{2,-2} \end{pmatrix} \tag{I.27}$$

$$q'_{l',m'} = \sum_{o=0}^{\infty} \sum_{p=-o}^{o} n_{o,p}\; q_{o,p}\; (\nabla)_{l'+o,m'-p;o,p} \frac{Y_{l'+o,m'-p}(\vartheta_a, \varphi_a)}{a^{l'+o+1}} \tag{I.28}$$

$$q_{l,m} = -q'_{l,m} R^{2l+1} \frac{l(\epsilon_i - \epsilon_a)}{l(\epsilon_a + \epsilon_i) + \epsilon_a} \tag{I.29}$$

$$\vec{q'}_{1,1} = \vec{q'}_{1,0} + I_{2,1}\; \vec{q'}_{2,0} \tag{I.30}$$

$$\vec{q'}_{2,1} = \vec{q'}_{2,0} + I_{1,2}\; \vec{q'}_{1,0} \tag{I.31}$$

$$\begin{aligned}
&\nabla_{\pm 1} \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} q_{l,m} \, \frac{Y_{l,m}(\vartheta,\varphi)}{r^{l+1}} \\
&\quad = \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} (\tilde{\nabla}_{\pm 1})_{l,m} \, q_{l,m} \, \frac{Y_{l+1,m\pm 1}(\vartheta,\varphi)}{r^{l+2}}
\end{aligned} \tag{I.32}$$

$$\underbrace{a + \overbrace{b + \cdots + y}^{123} + z}_{\alpha\beta\gamma}$$

Lange Formeln muß man selbst in Zeilen auflösen:

$$\begin{aligned}
w + x + y + z = {} & \\
& a + b + c + d + e + f + \\
& g + h + i + j + k + l
\end{aligned} \tag{I.33}$$

$$N_{+1} = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(\tilde{\nabla}_{+})_{0,0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (\tilde{\nabla}_{+})_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (\tilde{\nabla}_{+})_{1,0} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (\tilde{\nabla}_{+})_{1,-1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,1} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,0} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,-1} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,-2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Wie man an Gleichung (I.24) auf Seite 335 sieht, muß man erforderlichenfalls bei einem Gleichungssystem (`eqnarray`) mittels des Kommandos `\nonumber` dafür sorgen, daß es nur eine einzige, gemeinsame Nummer erhält. Zugleich ist dies hier ein Beispiel für einen Bezug auf eine Gleichung.

I.3 Formeln im Quelltext

```
\begin{equation}
```

```
c = \sqrt{a^{2} + b^{2}}
\end{equation}

\begin{equation}
\sqrt[3]{1 + x} \approx 1 + \frac{x}{3}
\qquad \mbox{f"ur} \quad x \ll 1
\end{equation}

\begin{equation}
r = \root 3 \of {\frac{3}{4\pi} V}
\end{equation}

\begin{equation}
\lim_{x \to 0} \frac{\sin x}{x} = 1
\end{equation}

\begin{equation}
a = \frac{F_0}{k}\:\frac{1}{\sqrt{(1 - \frac{\Omega^2}
{\omega_0^2})^2 + (\frac{\Omega_c}{k})^2}}
\end{equation}

\begin{equation}
m\ddot x + c\dot x + kx = \sum_k F_k \cos \Omega_k t
\end{equation}

\begin{equation}
\Theta \frac{d^2 \varphi}{dt^2} + k^\ast \frac{d \varphi}{dt}
+ D^\ast \varphi = | \vec D |
\end{equation}

\begin{equation}
\bar{Y} \approx f(\bar x) + \frac{1}{2}\, \frac{N - 1}{N}
\,f’’(\bar{x})\,s_x^2
\end{equation}

\begin{equation}
\vec{F_\Gamma} = - \frac{\Gamma m M}{r^2} \vec{e_r} =
 - \frac{\Gamma m M}{r^3} \vec{r}
\end{equation}

\begin{eqnarray}
\mbox{Arbeit} & = & \lim_{\Delta r_i \to 0} \sum
{\vec F_i \Delta \vec r_i}\nonumber \\
& = & \int\limits_{\vec r_0}^{\vec r(t)}
{\vec F(\vec r\,)\:d\vec r}
\end{eqnarray}
```

```
\begin{equation}
\prod_{j\ge0}\left( \sum_{k\ge0} a_{jk}z^k \right) =
\sum_{n \ge0} z^n \left(\sum_{h_0,k_1\ldots\ge0
\atop k_0+k_1+\cdots=0}
a_{0k_o} a_{1k_1}\ldots \right)
\end{equation}

\begin{equation}
\vec x =
\left( \begin{array}{c}
x - x_s \\
y - y_s \\
z - z_s \\
\end{array} \right)
\end{equation}

\begin{minipage}{120mm}
\begin{displaymath}
{\bf\Psi} = \left( \begin{array}{cc}
\displaystyle{ab \choose cd}
& \displaystyle \frac{e+f}{g-h} \\
\Re z & \displaystyle \left| {ij \atop kl} \right|
\end{array} \right)
\end{displaymath}
\end{minipage}
\stepcounter{equation}
\hspace*{\fill} (\theequation)\makebox[0pt][l]{\footnotemark}
\footnotetext{Fette Griechen gibt es nur als Gro\3buchstaben.
Die Erzeugung dieser Fu\3note war "ubrigens nicht einfach.}
\vspace{1mm}

\begin{equation}
dE_\omega = V \frac{\hbar}{\pi^2 c^3}\omega^3 \cdot
e ^{-\frac{\hbar \omega} {T}} \cdot d\omega
\end{equation}

\begin{equation}
\oint \vec E\:d\vec s = - \frac {\,\partial}{\partial t}
\int \vec B \:d\vec A
\end{equation}

\begin{equation}
\frac{1}{2 \pi j} \int\limits_{x-j\infty}^{x+j\infty}
e^{ts} \:f(s)\:ds =
\left\{\begin{array}{r@{\quad \mbox{f"ur} \quad}l}
0 & t < 0\\
F(t) & t > 0
```

```
\end{array} \right.
\end{equation}

\begin{equation}
{n+1 \choose k} = {n \choose k} + {n \choose k-1}
\end{equation}

\begin{equation}
\mbox{
\fbox{\parbox{60mm}{\begin{displaymath}
\forall x \in {\rm R}: \qquad x^2 \geq 0
\end{displaymath}}}}
\end{equation}
\vspace{1mm}

\begin{equation}
\forall x,y,z \in {\rm M}: \qquad (xRy \wedge xRz)
\Rightarrow y = z
\end{equation}

\begin{equation}
A \cdot B = \overline{\bar A + \bar B}
\end{equation}

\begin{equation}
\rho \cdot \bar{v}_k \cdot \frac{\partial \bar{v}_j}
{\partial{x}_k} = - \frac{\partial \bar{p}}
{\partial x_j} + \frac{\partial}{\partial x_k}
\left( \mu \frac{\partial \bar{v}_j}{\partial x_k}
- \rho\:\overline{v'_k v'_j} \right)
\end{equation}

\begin{equation}
r' = \frac{\overline{v'_1 v'_2}}{\sqrt{\bar{v'^2_1}}
\: \sqrt{\bar{v'^2_2}}}
\end{equation}

\begin{equation}
\tau_{tur} = \rho l^2 \; | \frac{\partial \bar{v_1}}
{\partial x_2} | \; \frac{\partial \bar{v_1}}{\partial x_2}
\end{equation}

\begin{eqnarray}
\label{formel}
\ddot R & = & \frac{1}{\Psi} ( V_r - \dot R)
+ R {\dot \Phi}^2 \nonumber\\
\ddot \Phi & = & \big\lbrack \frac{1}{\Psi}
```

```
(V_\varphi - R \dot \Phi )
- 2 \dot R \dot \Phi \big\rbrack \frac{1}{R} \\
\ddot Z & = & \frac{1}{\Psi}(V_Z-\dot Z) \nonumber\\
\nonumber
\end{eqnarray}

\begin{equation}
{\hat\chi}^2 = \frac{n(n-1)}{B(n-B)} \sum_{i=1}^k
\frac{(B_i - E_i)^2}{n_i} > \chi_{k-1;\alpha}^2
\end{equation}

\begin{equation}
V(r,\vartheta,\varphi)=\sum\limits_{l=0}^\infty
\frac{4\pi}{2l+1}
\sum\limits_{m=-l}^l \, q_{l,m} \frac{Y_{l,m}
\vartheta,\varphi)}{r^{l+1}}
\end{equation}

\begin{eqnarray}
\vec{q}= \left(\begin{array}{c}
q_{0,0}\\
q_{1,1}\\q_{1,0}\\q_{1,-1}\\
q_{2,2}\\q_{2,1}\\q_{2,0}\\q_{2,-1}\\q_{2,-2}\\
\end{array} \right)
\end{eqnarray}

\begin{equation}
q'_{l',m'}=\sum\limits_{o=0}^\infty\sum\limits_{p=-o}^o
n_{o,p} \; q_{o,p} \; (\nabla)_{l'+o,m'-p;o,p} \,
\frac{Y_{l'+o,m'-p}(\vartheta_a,\varphi_a)}{a^{l'+o+1}}
\end{equation}

\begin{equation}
q_{l,m}= -q'_{l,m} R^{2l+1} \,
\frac{l(\epsilon_i-\epsilon_a)}
{l(\epsilon_a+\epsilon_i)+\epsilon_a}
\end{equation}

\begin{eqnarray}
\vec{q'}_{1,1}=\vec{q'}_{1,0}+I_{2,1} \; \vec{q'}_{2,0}\\[0.3cm]
\vec{q'}_{2,1}=\vec{q'}_{2,0}+I_{1,2} \; \vec{q'}_{1,0}
\end{eqnarray}

\begin{eqnarray}
\lefteqn{\nabla_{\pm 1} \sum\limits_{l=0}^{\infty}
\frac{4\pi}{2l+1} \,
\sum\limits_{m=-l}^{l} \: q_{l,m} \: \frac{Y_{l,m}
```

```
(\vartheta,\varphi)}{r^{l+1}}}
\nonumber \\[0.3cm]
&&=\sum\limits_{l=0}^{\infty}\frac{4\pi}{2l+1} \,
\sum\limits_{m=-l}^{l} \: (\tilde{\nabla_{\pm 1}})_{l,m}
\, q_{l,m} \:
\frac{Y_{l+1,m \pm 1}(\vartheta,\varphi)}{r^{l+2}}
\end{eqnarray}

\[ \underbrace{a + \overbrace{b + \cdots + y}^{123}
                 + z}_{\alpha\beta\gamma}         \]

Lange Formeln mu\3 man selbst in Zeilen aufl"osen:
\begin{eqnarray}
\lefteqn{w + x + y + z = } \hspace{1cm} \nonumber\\
   & & a + b + c + d + e + f + \\
   & & g + h + i + j + k + l \nonumber
\end{eqnarray}

\begin{eqnarray*}
N_{+1}=\left(\begin{array}{*{8}{c@{\;}}c}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
(\tilde{\nabla}_{+})_{0,0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & (\tilde{\nabla}_{+})_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & (\tilde{\nabla}_{+})_{1,0} & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & (\tilde{\nabla}_{+})_{1,-1} & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,2} & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,1} & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,0} & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,-1} & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_{+})_{2,-2}\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
\end{array}\right)
\end{eqnarray*}

\vspace{8mm}
Wie man an Gleichung (\ref{formel}) auf Seite \pageref{formel}
sieht, mu\3 man erforderlichenfalls bei einem Gleichungssystem
({\tt eqnarray}) mittels des Kommandos \verb+\nonumber+ daf"ur
sorgen, da\3 es nur eine einzige, gemeinsame Nummer erh"alt.
Zugleich ist dies hier ein Beispiel f"ur einen Bezug auf eine
Gleichung.
```

J Karlsruher Test

Nicht jedermann eignet sich für so schwierige Dinge wie die elektronische Datenverarbeitung. Um Ihnen die Entscheidung zu erleichtern, ob Sie in die EDV einsteigen oder sich angenehmeren Dingen widmen sollten, haben wir ganz besonders für Sie einen Test entwickelt. Woran denken Sie bei:

Bit	Bier aus der Eifel (1 Punkt) Schraubendrehereinsatz (1) kleinste Dateneinheit (2 Punkte)
Festplatte	Was zum Essen, vom Partyservice (1) Schallplatte (0) Massenspeicher (2)
Menü	Was zum Essen (1) Dialogtechnik (2) mittelalterlicher Tanz (0)
CPU	politische Partei (0) Zentralprozessor (2) Carnevalsverein (0)
Linker	Linkshänder (0) Anhänger einer Linkspartei (1) Programm zum Binden von Modulen (2)
IBM	Ich Bin Müde (1) International Business Machines (2) International Brotherhood of Magicians (1)
Schnittstelle	Verletzung (1) Verbindungsstelle zweier EDV-Geräte (2) Werkstatt eines Bartscherers (0)
Slot	Steckerleiste im Computer (2) einarmiger Bandit (1) niederdeutsch für Kamin (0)

Fortran	starker Lebertran (0) Formal Trash Notation (0) Programmiersprache (2)
Mainframe	Frachtkahn auf dem Main (0) Damit wollte FRIDTJOF NANSEN zum Nordpol (0) großer Computer (2)
PC	Plumpsklo (Gravitationstoilette) (1) Personal Computer (2) Power Computing Language (0)
Puffer	Was zum Essen, aus Kartoffeln (1) Was am Eisenbahnwagen (1) Zwischenspeicher (2)
Software	Rohstoff für Softice (0) Programme, Daten und so Zeugs (2) was zum Trinken (0)
Port	was zum Trinken (1) Hafen (1) Steckdose für Peripheriegeräte (2)
Strichcode	maschinell lesbarer Code (2) Geheimsprache im Rotlichtviertel (0) Urliste in der Statistik (0)
Chip	was zum Essen (1) was zum Spielen (1) Halbleiterbaustein (2)
Pointer	Hund (1) starker Whisky (0) Zeiger auf Daten, Adresse (2)
Page	Hotelboy (1) englisch, Seite in einem Buch (1) Untergliederung eines Speichers (2)
Character	was manchen Politikern fehlt (1) Schriftzeichen (2) Wasserfall (0)

Betriebssystem	Konzern (0) betriebsinternes Telefonsystem (0) wichtigstes Programm im Computer (2)
Traktor	Papiereinzugsvorrichtung (2) landwirtschaftliches Fahrzeug (1) Zahl beim Multiplizieren (0)
Treiber	Hilfsperson bei der Jagd (1) Programm zum Ansprechen der Peripherie (2) Vorarbeiter (0)
Animator	was zum Trinken (1) Unterhalter (1) Programm für bewegte Grafik (2)
Hackbrett	Musikinstrument (1) Werkzeug im Hackbau (0) Tastatur (2)
emulieren	nachahmen (2) Öl in Wasser verteilen (0) entpflichten (0)
Font	Menge von Schriftzeichen (2) Soßengrundlage (1) Hintergrund, Geldmenge (0)
Server	Brettsegler (0) Kellner (0) Computer für Dienstleistungen (2)
Yabbawhap	Datenkompressionsprogramm (2) Kriegsruf der Südstadt-Indianer (0) was zum Essen (0)
Terminal	Schnittstelle Mensch - Computer (2) Bahnhof oder Hafen (1) Zubehör zu Drahttauwerk (1)
Ampersand	Sand aus der Amper (1) et-Zeichen, Kaufmanns-Und (2) Untiefe im Wattenmeer (0)

Alias	altgriechisches Epos (0) alttestamentarischer Prophet (0) Zweitname (2)
Buscontroller	Busfahrer (0) Busschaffner (0) Programm zur Steuerung eines Datenbusses (2)
Algol	was zum Trinken (0) Doppelstern (1) Programmiersprache (2)
Rom	Stadt in Italien (1) schwedisch für Rum (1) Read only memory (2)
Dram	Dynamic random access memory (2) dänisch für Schnaps (1) Straßenbahn (0)
Diskette	Mädchen, das oft in Discos geht (0) weiblicher Diskjockey (0) Massenspeicher (2)
Directory	oberste Etage einer Firma (0) Inhaltsverzeichnis (2) Kunststil zur Zeit der Franz. Revolution (0)
Dekrement	was die Verdauung übrig läßt (0) Anordnung von oben (0) Wert, um den ein Zähler verringert wird (2)
Sprungbefehl	Vorkommnis während Ihres Wehrdienstes (0) Kommando im Pferdesport (0) Anweisung in einem Programm (2)
Oktalzahl	Maß für die Klopffestigkeit (0) Zahl zur Basis 8 (2) Anzahl der Oktaven einer Orgel (0)
Subroutine	Kleidungsstück eines Priesters (0) was im Unterbewußten (0) Unterprogramm (2)

Spoiler	Was zum Essen (0)
	Posting in den Netnews (2)
	Was am Auto (1)
virtuell	tugendhaft (0)
	die Augen betreffend (0)
	nicht wirklich vorhanden, scheinbar (2)
Klammeraffe	ASCII-Zeichen (2)
	Bürogerät (1)
	Affenart in Südamerika (0)
ESC	Eisenbahner-Spar- und Creditverein (0)
	Eishockeyclub (0)
	escape, Fluchtsymbol (2)
Monitor	Karlsruher Brauerei (0)
	Fernsehsendung (1)
	Bildschirmgerät, Überwachungsprogramm (2)
Unix	Tütensuppe (0)
	Freund von Asterix und Obelix (0)
	hervorragendes Betriebssystem (2)
Joystick	Computerzubehör (2)
	männlicher Körperteil (0)
	Hebel am Spielautomat (0)
Maus	kleines Säugetier (1)
	Computerzubehör (2)
	junge Dame (1)
Icon	russisches Heiligenbild (0)
	Sinnbild (2)
	Kamerafabrik (0)
Pascal	französischer Mathematiker (1)
	Maßeinheit für Druck (1)
	Programmiersprache (2)
Wysiwyg	englisch für Wolpertinger (0)
	französisch für Elmentritschen (0)
	what you see is what you get (2)

Register	was in Flensburg (1) was an der Orgel (1) Speicher (2)
Record	was im Sport (1) englisch für Blockflöte (0) Datensatz (2)
HP	High Price (0) Hewlett-Packard (2) Horse Power (1)
Kermit	Klebstoff (0) Frosch aus der Muppet-Show (1) Fileübertragungs-Protokoll (2)
Ethernet	Baustoff (Asbestzement) (0) Local Area Network (2) Student der ETH Zürich (0)
Algorithmus	Übermäßiger Genuß geistiger Getränke (0) Krankheit (0) Rechenvorschrift (2)
File	Was zum Essen (0) Menge von Daten (2) Durchtriebener Kerl (0)
Bug	Vorderteil eines Schiffes (1) Fehler im Programm (2) englisch für Wanze (1)
Router	jemand mit Routine (0) französischer LKW-Fahrer (0) Verbindungsglied zweier Netze (2)
Zylinder	Kopfbedeckung (1) Teil einer Kolbenmaschine (1) Unterteilung eines Plattenspeichers (2)
FTP	kleine, aber liberale Partei (0) File Transfer Protocol (2) Floating Point Processor (0)

Domäne	Geist(0) Bereich (2) Blume (0)
Bridge	Kartenspiel (1) internationales Computernetz (0) Verbindung zweier Computernetze (2)
Email	Glasur (1) elektronische Post (2) Sultanspalast (0)
Baum	was im Wald (Wurzel unten) (1) was auf einem Schiff (keine Wurzel) (1) was aus der Informatik (Wurzel oben) (2)
Internet	Schule mit Schlafgelegenheit (0) Zwischenraum (0) Weltweites Computernetz (2)
Split	UNIX-Kommando (2) kantige Steinchen (0) Stadt in Dalmatien (1)
Mini	Damenoberbekleidung (1) kleiner Computer (2) Frau von Mickey Mouse (0)
Cut	Herrenoberbekleidung (1) Colonia Ulpia Traiana (1) UNIX-Kommando (2)
2B | !2B	Parallelprozessor (0) Assembler-Befehl (0) ein Wort Hamlets (2)
Shell	Filmschauspielerin (Maria S.) (0) Kommando-Interpreter (2) Mineralöl-Gesellschaft (1)
Slip	Unterbekleidung (1) Schlupfschuh (0) Internet-Protokoll (2)

Diäresis	Durchfall (0) Diakritisches Zeichen (Umlaute) (2) Ernährungslehre (0)
Space Bar	Kneipe im Weltraum (www.spacebar.com) (0) Maßeinheit für den Druck im Weltraum (0) Größte Taste auf der Tastatur (2)
Popper	Popcorn-Röster (0) Mail-Programm (2) Philosoph aus Wien (1)
Rohling	Wüster Kerl (1) Noch zu beschreibende CD/DVD (2) Rohkost-Liebhaber (0)
Schleife	Kleidungsstück (1) Schlitterbahn (1) Kontrollanweisung eines Programmes (2)
Alex	Altlasten-Expertensystem (1) Automatic Login Executor (1) Globales Filesystem (1)
Altair	Stern (Alpha Aquilae) (1) Gebirge in Zentralasien (0) früher Personal Computer (2)
Halbbitter	Was zum Essen (Schokolade) (1) Strom- und bitsparender Prozessor (0) Was zum Trinken (0)
Eure Priorität	Anrede des Priors in einem Kloster (0) Anrede des Ersten Sekretärs im Vatikan (0) Anrede des System-Managers (6)

Zählen Sie Ihre Punkte zusammen. Die Auswertung ergibt Folgendes:

- über 170 Punkte: Überlassen Sie das Rechnen künftig dem Computer.
- 85 bis 170 Punkte: Mit etwas Fleiß wird aus Ihnen ein EDV-Experte.
- 18 bis 84 Punkte: Machen Sie eine möglichst steile Karriere außerhalb der EDV und suchen Sie sich fähige Mitarbeiter.
- unter 18 Punkten: Vielleicht hatten Sie schlechte Lehrer?

K GNU Lizenzen

K.1 GNU General Public License

Die aktuellen Fassungen aller GNU-Lizenzen sind über `www.gnu.org/licenses/licenses.html` zu erreichen.

K.2 GNU Free Documentation License

Kopiert von `http://www.gnu.org/copyleft/`.

Preamble

The purpose of this license is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this license preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This license is a kind of *copyleft*, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this license in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this license is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this license principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This license applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this license. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **Document**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **you**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **Modified Version** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **Secondary Section** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **Invariant Sections** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this license. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **Cover Texts** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this license. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **Transparent** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the Document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not *Transparent* is called **Opaque**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **Title Page** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this license requires to appear in the title page. For works in formats which do not have any title page as such, TTitle Pagemmeans the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **Entitled XYZ** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **Acknowledgements**, **Dedications**, **Endorsements**, or **History**.) To **Preserve the Title** of such a section when you modify the Document means that it remains a section *Entitled XYZ* according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this license applies to the Document. These Warranty Disclaimers are considered to be included by reference in this license, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this license.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this license, the copyright notices, and the license notice saying this license applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this license. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this license, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this license, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this license.

I. Preserve the section entitled *History*, Preserve its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled *History* in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the *History* section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section entitled *Acknowledgements* or *Dedications*, Preserve the title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled *Endorsements*. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be entitled *Endorsements* or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled *Endorsements*, provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this license give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this license, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this license, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled *History* in the various original documents, forming one section entitled *History*; likewise combine any sections entitled *Acknowledgements*, and any sections entitled *Dedications*. You must delete all sections entitled *Endorsements*.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this license, and replace the individual copies of this license in the various documents with a single copy that is included in the collection, provided that you follow the rules of this license for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this license, provided you insert a copy of this license into the extracted document, and follow this license in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an aggregate if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this license does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this license, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this license and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this license or a notice or disclaimer, the original version will prevail.

If a section in the Document is entitled *Acknowledgements*, *Dedications*, or *History*, the requirement (section 4) to Preserve its title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this license. Any other attempt to copy, modify, sublicense or distribute the Document 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.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http://www.gnu.org/copyleft/.

Each version of the license is given a distinguishing version number. If the Document specifies that a particular numbered version of this license *or any later version* applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this license, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this license for your documents

To use this license in a document you have written, include a copy of the license in the Document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled *GNU Free Documentation License*.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the *with...Texts.* line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

L Zum Weiterlesen

1. Lexika, Glossare, Wörterbücher

   – RFC 1392 (FYI 18): Internet Users' Glossary
     `ftp://ftp.nic.de/pub/rfc/rfc1392.txt`
     1993, 53 S.

   – Duden Informatik
     Dudenverlag, Mannheim, 1993, 800 S.
     Nachschlagewerk, sorgfältig gemacht, theorielastig,
     Begriffe wie Ethernet, LAN, SQL, Internet fehlen.

   – Fachausdrücke der Informationsverarbeitung Englisch – Deutsch,
     Deutsch – Englisch
     IBM Deutschland, Form-Nr. Q12-1044, 1698 S.
     Wörterbuch und Glossar

   – IBM Terminology
     `http://www-3.ibm.com/ibm/terminology/`

   W. Alex Abkürzungs-Liste ABKLEX (Informatik, Telekommunikation)
     `http://www.abklex.de/abklex/`
     Rund 9000 Abkürzungen aus Informatik und Telekommunikation

   M. Broy, O. Spaniol Lexikon Informatik und Kommunikationstechnik
     Springer, Berlin + Heidelberg, 1999, 863 S.

   E. Kajan Information Technology Encyclopedia and Acronyms
     Springer, Berlin + Heidelberg, 2002, 720 S.

   E. S. Raymond The New Hacker's Dictionary
     The MIT Press, Cambridge, 1996, 547 S.
     Begriffe aus dem Netz, die nicht im Duden stehen

2. Informatik

   W. Coy Aufbau und Arbeitsweise von Rechenanlagen
     Vieweg, Braunschweig, 1992, 367 S.
     Digitale Schaltungen, Rechnerarchitektur, Betriebssysteme am
     Beispiel von UNIX

   T. Flik, H. Liebig Mikroprozessortechnik
     Springer, Berlin + Heidelberg, 1998, 585 S.
     CISC, RISC, Systemaufbau, Assembler und C

   W. K. Giloi Rechnerarchitektur
     Springer, Berlin + Heidelberg, 1999, 488 S.

   G. Goos Vorlesungen über Informatik
     Band 1: Grundlagen und funktionales Programmieren,

Springer, Berlin + Heidelberg, 1997, 394 S.
Band 2: Objektorientiertes Programmieren und Algorithmen,
Springer, Berlin + Heidelberg, 1999, 396 S.
Band 3: Berechenbarkeit, formale Sprachen, Spezifikationen,
Springer, Berlin + Heidelberg, 1997, 284 S.
Band 4: Paralleles Rechnen und nicht-analytische Lösungsverfahren,
Springer, Berlin + Heidelberg, 1998, 292 S.
`i44www.info.uni-karlsruhe.de/~i44www/goos-buch.html`

D. E. Knuth The Art of Computer Programming, 3 Bände
Addison-Wesley, Boston,
Klassiker, stellenweise mathematisch, 7 Bände geplant,
Band 4 soll 2004 fertig sein, Band 5 im Jahr 2009, Homepage
des Meisters: `www-cs-staff.stanford.edu/~uno/index.html`

W. Schiffmann, R. Schmitz Technische Informatik
Springer, Berlin + Heidelberg, 1993/94, 1. Teil Grundlagen der digitalen Elektronik, 282 S.; 2. Teil Grundlagen der Computertechnik, 283 S.

K. W. Wagner Einführung in die Theoretische Informatik
Springer, Berlin + Heidelberg, 1994, 238 S.
Grundlagen, Berechenbarkeit, Komplexität, BOOLEsche Funktionen, Automaten, Grammatiken, Formale Sprachen

3. Algorithmen, Numerische Mathematik

J. L. Bentley Programming Pearls
Addison-Wesley, Boston, 1999, 256 S.
Pfiffige Algorithmen und Programmierideen

G. Engeln-Müllges, F. Reutter Formelsammlung zur Numerischen Mathematik mit C-Programmen
BI-Wissenschaftsverlag, Mannheim, 1990, 744 S.
Algorithmen und Formeln der Numerischen Mathematik samt C-Programmen.

G. Engeln-Müllges, F. Uhlig Numerical Algorithms with C
Springer, Berlin + Heidelberg, 1996, 596 S.

D. E. Knuth Algorithmen
(deutsche Übersetzung von *Fundamental Algorithms*)
Springer, Berlin + Heidelberg, 2004, 700 S.

K. Loudon Mastering Algorithms in C
O'Reilly, Sebastopol, 1999, 560 S.

T. Ottmann, P. Widmayer Algorithmen und Datenstrukturen
BI-Wissenschafts-Verlag, Mannheim, 1993, 755 S.

W. H. Press u. a. Numerical Recipes in C
Cambridge University Press, 1993, 994 S.

H. R. Schwarz Numerische Mathematik
Teubner, Stuttgart, 1993, 575 S.

R. Sedgewick Algorithmen in C
Addison-Wesley, Bonn, 1992, 742 S.
Erklärung gebräuchlicher Algorithmen und Umsetzung in C

R. Sedgewick Algorithmen in C++
Addison-Wesley, Bonn, 1992, 742 S.

J. Stoer, R. Bulirsch Numerische Mathematik
Springer, Berlin + Heidelberg, 1. Teil 1999, 378 S.,
2. Teil 2000, 375 S.

4. Betriebssysteme

L. Bic, A. C. Shaw Betriebssysteme
Hanser, München, 1990, 420 S.
Allgemeiner als Tanenbaum + Woodhull

A. S. Tanenbaum, A. S. Woodhull Operating Systems, Design and Implementation
Prentice-Hall, London, 1997, 939 S.
Einführung in Betriebssysteme am Beispiel von UNIX

A. S. Tanenbaum Modern Operating Systems
Prentice-Hall, London, 1992, 728 S.
Allgemeiner und moderner als vorstehendes Buch;
erläutert MS-DOS, UNIX, MACH und Amoeba

A. S. Tanenbaum Distributed Operating Systems
Prentice-Hall, London, 1994, 648 S.

H. Wettstein Systemarchitektur
Hanser, München, 1993, 514 S.
Grundlagen, kein bestimmtes Betriebssystem

5. Linux/UNIX allgemein

W. Alex Debian GNU/Linux in der Praxis
Springer, Berlin + Heidelberg, 2005, 560 S.
Anwendungen, Konzepte, Werkzeuge unter Debian GNU/Linux

M. J. Bach Design of the UNIX Operating System
Prentice-Hall, London, 1987, 512 S.
Dateisystem und Prozesse, wenig zur Shell

S. R. Bourne Das UNIX System V (The UNIX V Environment)
Addison-Wesley, Bonn, 1988, 464 S.
Einführung in UNIX und die Bourne-Shell

P. H. Ganten, W. Alex Debian GNU/Linux
Springer, Berlin + Heidelberg, 2004, 970 S.
Einrichtung, Konfiguration und Betrieb von Debian GNU/Linux

J. Gulbins, K. Obermayr, Snoopy Linux
Springer, Berlin + Heidelberg, 2003, 900 S.
Benutzung von Linux/UNIX, geht in Einzelheiten der Kommandos

H. Hahn A Student's Guide to UNIX
McGraw-Hill, New York, 1993, 633 S.
Einführendes Lehrbuch, mit Internet-Diensten

B. W. Kernighan, R. Pike Der UNIX-Werkzeugkasten
Hanser, München, 1986, 402 S.
Gebrauch vieler UNIX-Kommandos

M. Kofler Linux – Installation, Konfiguration, Anwendung
Addison-Wesley, Bonn, 2005, 1318 S.
7. Auflage, spricht für das Buch.

D. G. Korn, M. I. Bolsky The Kornshell, Command and Programming Language
deutsch: Die KornShell, Hanser, München, 1991
Einführung in UNIX und die Korn-Shell

A. Robbins UNIX in a Nutshell
O'Reilly, Sebastopol, 2000, 632 S.
Nachschlagewerk zu den meisten UNIX-Kommandos, im UNIX CD Bookshelf enthalten. Auch auf Englisch.

M. J. Rochkind Advanced UNIX Programming
Addison-Wesley, Boston, 2004, 719 S.
Beschreibung der wichtigsten UNIX System Calls

K. Rosen u. a. UNIX: The Complete Reference
Osborne/McGraw-Hill, Berkeley, 1999, 1302 S.
Fast würfelförmiges Nachschlagewerk, insbesondere zu Linux, Solaris und HP-UX; breites Themenspektrum

E. Siever et al. LINUX in a Nutshell
O'Reilly, Sebastopol, 2001, 880 S.
Nachschlagewerk zu den meisten LINUX-Kommandos

W. R. Stevens Advanced Programming in the UNIX Environment
Addison-Wesley, Boston, 1992, 744 S.
Ähnlich wie Rochkind

6. Linux/UNIX Verwaltung

Æ. Frisch Essential System Administration
O'Reilly, Sebastopol, 1995, 760 S.
Übersicht für Benutzer auf dem Weg zum Sysadmin.

K. Heuer, R. Sippel UNIX-Systemadministration
Springer, Berlin + Heidelberg, 2004, 800 S.

E. Nemeth, G. Snyder, S. Seebass, T. R. Hein UNIX System Administration Handbook
Prentice-Hall, Englewood-Cliffs, 2001, 835 S.

Auf den neuesten Stand gebrachte Hilfe für Sysadmins, viel Stoff.

R. U. Rehman HP Certified – HP-UX System Administration
Prentice Hall PTR, Upper Saddle River, 2000, 800 S.
Begleitbuch zu einem Kurs, Einführung in und Verwaltung von HP-UX

M. Welsh, M. K. Dalheimer, L. Kaufmann Running Linux
O'Reilly, Sebastopol, 1999, 750 S.
Einrichtung und Betrieb eines LINUX-PCs

7. Linux/UNIX Einzelthemen

– Newsgruppen:
comp.unix.*

A. V. Aho, B. W. Kernighan, P. J. Weinberger The AWK Programming Language
Addison-Wesley, Boston, 1988, 210 S.
Standardwerk zum AWK

D. Cameron, B. Rosenblatt Learning GNU Emacs
O'Reilly, Sebastopol, 1991, 442 S.

D. Dougherty, A. Robbins sed & awk
O'Reilly, Sebastopol, 1997, 407 S.

H. Herold Linux Unix Profitools: awk, sed, lex, yacc und make
Addison-Wesley, München, 1998, 890 S.

L. Lamb, A. Robbins Textbearbeitung mit dem vi-Editor
O'Reilly, Köln, 1999, 333 S.

A. Oram, S. Talbott Managing Projects with make
O'Reilly, Sebastopol, 1993, 149 S.

L. Wall, T. Christiansen, J. Orwant Programming Perl
O'Reilly, Sebastopol, 2000, 1067 S.

8. X Window System (X11), Motif, Gnome, KDE

– Newsgruppen:
comp.windows.x.*

– OSF/Motif Users's Guide
OSF/Motif Programmer's Guide
OSF/Motif Programmer's Reference
Prentice-Hall, Englewood Cliffs, 1990

F. Culwin An X/Motif Programmer's Primer
Prentice-Hall, New York, 1994, 344 S.

T. + M. K. Dalheimer KDE Anwendung und Programmierung
O'Reilly, Sebastopol, 1999, 321 S.

K. Gottheil u. a. X und Motif
Springer, Berlin + Heidelberg, 1992, 694 S.

N. Mansfield The Joy of X
Addison-Wesley, Boston, 1993, 368 S.
Guter Einstieg für Anwender, leider nicht aktuell.

A. Nye XLib Programming Manual
O'Reilly, Sebastopol, 1990, 635 S.
Einführung in X11 und den Gebrauch der XLib

V. Quercia, T. O'Reilly X Window System Users Guide
O'Reilly, Sebastopol, 1990, 749 S.
Einführung in X11 für Anwender

R. J. Rost X and Motif Quick Reference Guide
Digital Press, Bedford, 1993, 400 S.

9. Textverarbeitung mit LaTeX

K. Braune, J. Lammarsch, M. Lammarsch LaTeX
Springer, Berlin + Heidelberg, 2006, 700 S.

M. K. Dalheimer LaTeX kurz & gut
O'Reilly, Köln, 2000, 72 S.

H. Kopka LaTeX, 3 Bände
Band 1: Einführung
Addison-Wesley, Bonn, 2000, 520 S.
Band 2: Ergänzungen
Addison-Wesley, Bonn, 1997, 456 S.
Band 3: Erweiterungen
Addison-Wesley, Bonn, 1996, 512 S.
Standardwerk im deutschen Sprachraum

L. Lamport Das LaTeX-Handbuch
Addison-Wesley, Bonn, 1995, 360 S.

H. Partl u. a. LaTeX-Kurzbeschreibung
Im Netz, Einführung mit deutschsprachigen Besonderheiten

10. Multimedia (Grafik, Sound)

- Newsgruppen:
  comp.graphics.*
  alt.graphics.*

J. D. Foley Computer Graphics – Principles and Practice
Addison-Wesley, Boston, 1992, 1200 S.
Standardwerk zur Computer-Raster-Grafik

R. F. Ferraro Programmer's Guide to the EGA and VGA Cards
Addison-Wesley, Boston, 1990, 1040 S.
Viele Grundlagen, die über EGA und VGA hinausgehen

K. Kylander, O. S. Kylander GIMP
MITP-Verlag, Bonn, 1999, 700 S.
Benutzerhandbuch zum GNU Image Manipulation Program

11. Netze allgemein (Internet, OSI)

- EFF's Guide to the Internet
`http://www.eff.org/pub/Publications/EFF\_Net\_Guide/`
Einführung in die Dienste des Internet

S. Carl-Mitchell, J. S. Quarterman Practical Internetworking with TCP/IP and UNIX
Addison-Wesley, Boston, 1993, 432 S.

D. E. Comer Internetworking with TCP/IP (4 Bände)
Prentice-Hall, Englewood Cliffs, I. Band 1991, 550 S.
II. Band 1991, 530 S., 88 DM; IIIa. Band (BSD) 1993, 500 S.
IIIb. Band (AT&T) 1994, 510 S.
Prinzipien, Protokolle und Architektur des Internet

H. Hahn, R. Stout The Internet Complete Reference
Osborne MacGraw-Hill, Berkeley, 1994, 818 S.
Das Netz und seine Dienste von Mail bis WWW; Lehrbuch und Nachschlagewerk für Benutzer des Internet

C. Hunt TCP/IP Netzwerk-Administration
O'Reilly, Sebastopol, 1998, 632 S.

B. P. Kehoe Zen and the Art of the Internet
`ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/zen.ps.gz`
1992, 100 S., Postscript
Einführung in die Dienste des Internet

O. Kirch, T. Dawson Linux Network Administrator's Guide
O'Reilly, Sebastopol, 2000, 500 S.

E. Krol The Hitchhikers Guide to the Internet
`ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/hitchhg.txt`
1987, 16 S., ASCII
Erklärung einiger Begriffe aus dem Internet

E. Krol The Whole Internet
O'Reilly, Sebastopol, 1992, 376 S.

J. F. Kurose, K. W. Ross Computer Networking
Addison-Wesley, Boston, 2003, 784 S.

M. Scheller u. a. Internet: Werkzeuge und Dienste
Springer, Berlin + Heidelberg, 1994, 280 S.
`http://www.ask.uni-karlsruhe.de/books/inetwd.html`

A. S. Tanenbaum Computer Networks
Prentice-Hall, London, 1996, 848 S.
Einführung in Netze mit Schwerpunkt auf dem OSI-Modell

12. Netzdienste Einzelthemen

P. Albitz, C. Liu DNS and BIND
O'Reilly, Sebastopol, 1998, 482 S.
Internet-Adressen und -Namen, Name-Server

B. Costales, E. Allman sendmail
O'Reilly, Sebastopol, 1997, 1021 S.
Das wichtigste netzseitige Email-Programm (MTA) ausführlich dargestellt, keine leichte Kost, aber unentbehrlich

J. E. Hellbusch Barrierefreies Webdesign
KnowWare, `www.knowware.de/`, 2001, 86 S.
Hinweise zur Gestaltung von Webseiten, kompakt und verständlich

P. J. Lynch, S. Horton Web Style Guide
Yale University Press, New Haven, 1999, 165 S.
Gestaltung und Organisation von Webseiten, wenig Technik

C. Meinel, H. Sack WWW
Springer, Berlin + Heidelberg, 2004, 1200 S.
Internet-Grundlagen, HTTP, HTML, CSS, XML, CGI

S. Münz, W. Nefzger HTML 4.0 Handbuch
Franzis, München, 1999, 992 S.
Deutsches Standardwerk zum Schreiben von Webseiten, abgewandelt auch unter dem Titel *Selfhtml* an mehreren Stellen im Netz verfügbar.

J. Niederst Web Design in a Nutshell
O'Reilly, Sebastopol, 1999, 560 S.
Das gesamte Web zum Nachschlagen, viel Technik

A. Schwartz Managing Mailing Lists
O'Reilly, Sebastopol, 1998, 320 S.
Majordomo, Listserv, List Processor und Smartlist

S. Spainhour, R. Eckstein Webmaster in a Nutshell
O'Reilly, Sebastopol, 1999, 523 S.
HTML, CSS, XML, JavaScript, CGI und Perl, PHP, HTTP, Apache

W. R. Stevens UNIX Network Programming
Vol. 1: Networking APIs: Sockets and XTI
Prentice Hall, Englewood Cliffs, 1998, 1009 S.
Vol. 2: Interprocess Communication
Prentice Hall, Englewood Cliffs, 1999, 592 S.
C-Programme für Clients und Server der Netzdienste

13. Sicherheit

– RFC 1244 (FYI 8): Site Security Handbook
`ftp://ftp.nic.de/pub/rfc/rfc1244.txt`
1991, 101 S., ASCII
Sicherheits-Ratgeber für Internet-Benutzer

– Department of Defense Trusted Computer Systems
  Evaluation Criteria (Orange Book)
  ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/secur/orange-book.gz
  1985, 120 S., ASCII. Abgelöst durch:
  Federal Criteria for Information Technology Security
  ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/secur/fcvol1.ps.gz
  ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/secur/fcvol2.ps.gz
  1992, 2 Bände mit zusammen 500 S., Postscript
  Die amtlichen amerikanischen Sicherheitsvorschriften

– Linux Hacker's Guide
  Markt + Technik, München, 1999, 816 S.

F. L. Bauer Kryptologie
Springer, Berlin + Heidelberg, 1994, 369 S.

R. L. Brand Coping with the Threat of Computer Security Incidents
A Primer from Prevention through Recovery
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/secur/primer.ps.gz
1990, 44 S., Postscript

D. A. Curry Improving the Security of Your UNIX System
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/secur/secdoc.ps.gz
1990, 50 S., Postscript
Hilfe für UNIX-System-Verwalter, mit Checkliste

S. Garfinkel, G. Spafford Practical Unix + Internet Security
O'Reilly, Sebastopol, 1996, 971 S.
Breit angelegte, verständliche Einführung in Sicherheitsthemen

B. Schneier Angewandte Kryptographie
Addison-Wesley, Bonn, 1996, 844 S.

M. Schumacher, U. Roedig, M.-L. Moschgath Hacker Contest
Springer, Berlin + Heidelberg, 2003, 300 S.

14. Computerrecht

– World Intellectual Property Organization (WIPO)
  http://www/wipo.int/

– Juristisches Internetprojekt Saarbrücken
  http://www.jura.uni-sb.de/

– Netlaw Library (Universität Münster)
  http://www.jura.uni-muenster.de/netlaw/

– Online-Recht http://www.online-recht.de/

– Computerrecht (Beck-Texte)
  Beck, München, 1994

U. Dammann, S. Simitis Bundesdatenschutzgesetz
Nomos Verlag, Baden-Baden, 1993, 606 S.
BDSG mit Landesdatenschutzgesetzen und Internationalen
Vorschriften; Texte, kein Kommentar

G. v. Gravenreuth Computerrecht von A – Z (Beck Rechtsberater)
Beck, München, 1992

H. Hubmann, M. Rehbinder Urheber- und Verlagsrecht
Beck, München, 1991, 319 S.

A. Junker Computerrecht. Gewerblicher Rechtsschutz, Mängelhaftung, Arbeitsrecht. Reihe Recht und Praxis
Nomos Verlag, Baden-Baden, 1988, 267 S.

F. Koch Handbuch Software- und Datenbank-Recht
Springer, Berlin + Heidelberg, 2003, 1000 S.

D. Kröger, M. A. Gimmy Handbuch zum Internetrecht
Springer, Berlin + Heidelberg, 2. Auflage 2002, 1000 S.

15. Geschichte der Informatik

– Kleine Chronik der IBM Deutschland
1910 – 1979, Form-Nr. D12-0017, 138 S.
1980 – 1991, Form-Nr. D12-0046, 82 S.
Reihe: Über das Unternehmen, IBM Deutschland

– Die Geschichte der maschinellen Datenverarbeitung Band 1
Reihe: Enzyklopädie der Informationsverarbeitung
IBM Deutschland, 228 S., Form-Nr. D12-0028

– 100 Jahre Datenverarbeitung Band 2
Reihe: Über die Informationsverarbeitung
IBM Deutschland, 262 S., Form-Nr. D12-0040

– Open Source
O'Reilly, Köln, 1999, 70 S.

P. E. Ceruzzi A History of Modern Computing
MIT Press, Cambridge/USA, 1998, 400 S.
Computergeschichte seit 1945 aus nordamerikanischer Sicht

O. A. W. Dilke Mathematik, Maße und Gewichte in der Antike (Universalbibliothek Nr. 8687 [2])
Reclam, Stuttgart, 1991, 135 S.

M. Hauben, R. Hauben Netizens – On the History and Impact of Usenet and the Internet
IEEE Computer Society Press, Los Alamitos, 1997, 345 S.
`www.columbia.edu/~hauben/netbook/`

A. Hodges Alan Turing, Enigma
Kammerer & Unverzagt, Berlin, 1989, 680 S.

D. M. Lehmann Der EDV-Pionier Nikolaus Joachim Lehmann
Dr. Hänsel-Hohenhausen, Frankfurt (M), 2002,

S. Levy Hackers – Heroes of the Computer Revolution
Penguin Books, London, 1994, 455 S.

R. Oberliesen Information, Daten und Signale
Deutsches Museum, rororo Sachbuch Nr. 7709 (vergriffen)

D. Shasha, C. Lazere Out of Their Minds
Springer, Berlin + Heidelberg, 1995, 295 S.
Biografien berühmter Computerpioniere

D. Siefkes u. a. Pioniere der Informatik
Springer, Berlin + Heidelberg, 1998, 160 S.
Interviews mit fünf europäischen Computerpionieren

B. Sterling A short history of the Internet
`ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/history/origins`
1993, 6 S., ASCII

K. Zuse Der Computer - Mein Lebenswerk
Springer, Berlin + Heidelberg, 3. Aufl. 1993, 220 S.
Autobiografie Konrad Zuses

16. Allgemeinwissen und Philosophie

E. Dyson Release 2.1 – A Design for Living in the Digital Age
Petersen, Hamburg, 2000, 370 S.

D. R. Hofstadter Gödel, Escher, Bach - ein Endloses Geflochtenes Band
dtv/Klett-Cotta, München, 1992, 844 S.

J. Ladd Computer, Informationen und Verantwortung
in: Wissenschaft und Ethik, herausgegeben von H. Lenk
Reclam-Band 8698, Ph. Reclam, Stuttgart

H. Lenk Chancen und Probleme der Mikroelektronik, und:
Können Informationssysteme moralisch verantwortlich sein?
in: Hans Lenk, Macht und Machbarkeit der Technik
Reclam-Band 8989, Ph. Reclam, Stuttgart, 1994, 152 S.

P. Schefe u. a. Informatik und Philosophie
BI Wissenschaftsverlag, Mannheim, 1993, 326 S.
18 Aufsätze verschiedener Themen und Meinungen

K. Steinbuch Die desinformierte Gesellschaft
Busse + Seewald, Herford, 1989, 269 S. (vergriffen)

J. Weizenbaum Die Macht der Computer und die Ohnmacht der Vernunft (Computer Power and Human Reason. From Judgement to Calculation)
Suhrkamp Taschenbuch Wissenschaft 274, Frankfurt (Main), 1990, 369 S.

H. Zemanek Das geistige Umfeld der Informationstechnik
Springer, Berlin + Heidelberg, 1992, 303 S.
Zehn Vorlesungen über Technik, Geschichte und Philosophie des Computers, von einem der Pioniere

17. Zeitschriften

– c't
Verlag Heinz Heise, Hannover, vierzehntägig,

für alle Fragen der Computerei, technisch.
`http://www.ix.de/`

- IX
  Verlag Heinz Heise, Hannover, monatlich,
  für Anwender von Multi-User-Systemen, technisch.
  `http://www.ix.de/`
- The C/C++ Users Journal
  Miller Freeman Inc., USA, monatlich,
  `http://www.cuj.com/`
- Dr. Dobb's Journal
  Miller Freeman Inc., USA, monatlich,
  `http://www.ddj.com/`
  Software Tools for the Professional Programmer; viel C und C++

Und noch einige Verlage:

- Addison-Wesley, Bonn,
  `http://www.addison-wesley.de/`
- Addison Wesley Longman, USA,
  `http://www.awl.com/`
- Computer- und Literaturverlag, Vaterstetten,
  `http://www.cul.de/`
- Carl Hanser Verlag, München,
  `http://www.hanser.de/`
- Verlag Heinz Heise, Hannover,
  `http://www.heise.de/`
- International Thomson Publishing, Stamford,
  `http://www.thomson.com/`
- Klett-Verlag, Stuttgart,
  `http://www.klett.de/`
- MITP-Verlag, Bonn,
  `http://www.mitp.de/`
- R. Oldenbourg Verlag, München,
  `http://www.oldenbourg.de/`
- O'Reilly, Deutschland,
  `http://www.ora.de/`
- O'Reilly, Frankreich,
  `http://www.editions-oreilly.fr/`
- O'Reilly, USA,
  `http://www.ora.com/`
- Osborne McGraw-Hill, USA,
  `http://www.osborne.com/`

- Prentice-Hall, USA,
  `http://www.prenhall.com/`
- Sams Publishing (Macmillan Computer Publishing), USA,
  `http://www.mcp.com/`
- Springer-Verlag, Berlin, Heidelberg, New York usw.,
  `http://www.springer.de/`
- Wrox Press, Chicago, Birmingham, Paris,
  `http://www.wrox.com/`

> Und über allem, mein Sohn, lass dich warnen;
> denn des vielen Büchermachens ist kein Ende,
> und viel Studieren macht den Leib müde.
>
> Prediger 12, 12

Personenverzeichnis

Sachverzeichnis