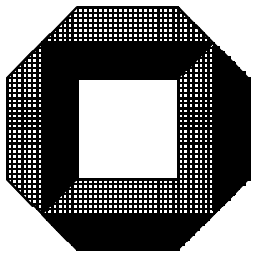

Einführung in UNIX

W. Alex, G. Bernör und B. Alex

1998

Universität Karlsruhe

Email: wulf.alex@ciw.uni-karlsruhe.de

gerhard.bernoer@ciw.uni-karlsruhe.de

Telefon: 0721/608-2404

Fax: 0721/693965

Ausgabedatum: 20. April 1998.

Geschützte Namen wie UNIX oder Postscript werden ohne Kennzeichnung verwendet.

Geschrieben mit dem Editor `vi(1)` auf einer Hewlett-Packard 9000/712 unter HP-UX (UNIX System V), formatiert mit LaTeX auf einem PC unter LINUX, ausgegeben auf einem Hewlett-Packard Laserjet 4Si unter Verwendung von Postscript.

Alle Programmbeispiele sind im Internet mittels Anonymous-FTP von `ftp.ciw.uni-karlsruhe.de`, Verzeichnis `pub/skriptum/...` abrufbar, ebenso die angeführte elektronische Literatur im Verzeichnis `pub/docs/...` Ferner finden sich unter `http://www.ciw.uni-karlsruhe.de/technik.html` Hinweise auf weitere Informationen.

Das Skriptum ist bei der Skriptenverkaufsstelle des Studentenwerks der Universität Karlsruhe erhältlich, außerdem per Anonymous-FTP als Postscript-File (100 % logisch abbaubar) im Netz, siehe oben.

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

Unser Buch wendet sich an Leser mit wenigen Vorkenntnissen in der Elektronischen Datenverarbeitung (EDV); es soll – wie FRITZ REUTERS *Urgeschicht von Meckelnborg* – ok för Schaulkinner tau bruken sin. Für die wissenschaftliche Welt zitieren wir 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.", wobei wir ergänzen, daß uns dieser Satz eine noch nicht erreichte Verpflichtung ist und vermutlich bleiben wird. Das Nahziel ist eine Vertrautheit mit dem Betriebssystem UNIX, der Programmiersprache C/C++ und dem internationalen Computernetz Internet, die so weit reicht, daß der Leser selbständig weiterarbeiten kann. Ausgelernt hat man nie.

Der Text besteht aus acht Teilen. Nach anfänglichen Schritten zur Eingewöhnung in den Umgang mit dem Computer beschreibt der zweite Teil kurz die Hardware, der dritte das Betriebssystem UNIX, der vierte die Programmiersprache C/C++, der fünfte das Internet mit Schwerpunkt Netzdienste, der sechste einige Anwendungen und der siebte Rechtsfragen im Zusammenhang mit der EDV. Ein Anhang enthält Fakten, die man immer wieder braucht. Für die zweite Auflage wurden viele Kleinigkeiten verbessert, dem Internet ein eigenes Kapitel gewidmet und die objektorientierten Zweige von C berücksichtigt. Bei der Stoffauswahl haben wir uns von unserer Arbeit als Benutzer und Verwalter international vernetzter UNIX-Systeme sowie als Programmierer vorzugsweise in C/C++ und FORTRAN leiten lassen.

Hinsichtlich vieler Einzelheiten wird auf die Referenz-Handbücher zu den Rechenanlagen und Programmiersprachen verwiesen. Wir wollen nicht den Text durch Dinge aufblähen, die man besser dort nachschlägt. Den Umfang haben wir auf rund 600 Seiten beschränkt, um den Leser nicht durch zu hohe Anforderungen an seinen Geldbeutel und an seine Zeit abzuschrecken. *Alles über UNIX, C und das Internet* ist kein Buch, sondern ein Bücherschrank.

UNIX ist das erste und einzige Betriebssystem, das auf einer Vielzahl von Computertypen läuft. Das ist sein größter Vorzug. Wir haben versucht, möglichst unabhängig von einer bestimmten Anlage zu schreiben. Über örtliche Besonderheiten müssen Sie sich daher aus weiteren Quellen unterrichten. In der Universität Karlsruhe kommt dafür das *UNIX-Handbuch* des Rechenzentrums (`http://www.uni-karlsruhe.de/~rz90/gesamt.dok.html`) in Frage. Anderenorts gibt es ähnliche Hilfen. Eng mit UNIX zusammen hängt das X Window System (X11), ein netzfähiges grafisches Fenstersystem, das heute fast überall die Kommandozeile als Benutzerschnittstelle ersetzt hat.

Die Programmiersprache C mit ihrer Erweiterung C++ ist – im Vergleich zu BASIC etwa – ziemlich einheitlich. Wir haben die Programmbeispiele unter mehreren Compilern getestet. Ob C/C++ besser ist als FORTRAN oder PASCAL oder sonst irgendeine neuere Programmiersprache, darüber läßt sich end- und fruchtlos streiten, aber nicht mit uns.

Das Internet ist das größte internationale Computernetz, eigentlich ein Zusammenschluß vieler regionaler Netze. Vor allem Universitäten und Behörden sind eingebunden, zum Teil auch die Industrie. Es ist nicht nur eine Daten-Autobahn, sondern eine ganze Landschaft. Wir gehen etwas optimistisch davon aus, daß jeder Leser einen Zugang zum Netz hat. Bei der gegenwärtigen raschen Entwicklung ist der Netzzugang tatsächlich nur noch eine Zeitfrage. Diesem Buch liegt daher keine Diskette oder Compact Disk bei, die Programme und ergänzende Texte stehen im Netz zur Verfügung. UNIX, C/C++ und das Internet könnten unabhängig voneinander betrachtet werden, in der Praxis jedoch sind sie miteinander verflochten.

An einigen Stellen gehen wir außer auf das Wie auch auf das Warum ein. Von Zeit zu Zeit sollte man den Blick weg von den Bäumen auf den Wald richten, sonst häuft man nur kurzlebiges Wissen an.

Man kann den Gebrauch eines Betriebssystems, einer Programmiersprache oder der Netzdienste nicht ohne praktische Übungen erlernen – das ist wie beim Klavierspielen oder Kuchenbacken. Die Beispiele und Übungen wurden auf einer Hewlett-Packard 9000/712 unter HP-UX (UNIX V) 10.2 und einem Pentium-PC der Marke Weingartener Katzenberg Auslese unter Microsoft DOS 6.2 sowie unter LINUX entwickelt. Als Shell wurde die Korn-Shell (11/16/88) bevorzugt, als Compiler wurden neben den zu den jeweiligen Betriebssystemen gehörenden Produkten der GNU gcc 2.6.3 und der Watcom 10.6 verwendet.

Dem Text liegen eigene Erfahrungen aus vier Jahrzehnten Umgang mit elektronischen Rechenanlagen und aus Kursen über BASIC, FORTRAN, C/C++ und UNIX zugrunde. Wir haben auch fremde Hilfe beansprucht und danken Kollegen in den Universitäten Karlsruhe und Lyon sowie Mitarbeitern der Firmen IBM und Hewlett-Packard für schriftliche Unterlagen und mündliche Hilfe sowie zahlreichen Studenten für Anregungen und Diskussionen. Dr. iur. Elke L. Barnstedt, ihrerzeit Universität Karlsruhe, hat freundlicherweise die erste Fassung des Kapitels *Computerrecht* beigesteuert. Darüber hinaus haben wir nach Kräften das Internet angezapft und viele dort umlaufende Guides, Primers, Tutorials und Sammlungen von Frequently Asked Questions (FAQs) verwendet. Dem Springer-Verlag danken wir dafür, daß er uns geholfen hat, aus einem lockeren Skriptum ein ernsthaftes Buch zu machen.

So eine Arbeit wird eigentlich nie fertig, man muß sie für fertig erklären, wenn man nach Zeit und Umständen das Möglichste getan hat, um es mit Johann Wolfgang von Goethe zu sagen (Italienische Reise; Caserta, den 16. März 1787). Wir erklären unsere Arbeit für unfertig und bitten, uns die Mängel nachzusehen.

Weingarten (Baden), 4. Januar 1998 Wulf Alex

Übersicht

Zum Gebrauch

- Hervorhebungen im Text werden *kursiv* dargestellt.
- Zitate und Titel von Veröffentlichungen oder Abschnitten werden im Text *kursiv* markiert.
- In Aussagen über Wörter werden diese *kursiv* abgesetzt.
- Stichwörter (wie sie für einen Vortrag oder eine Vorlesung benötigt werden) erscheinen in **fetterer** Schrift.
- Namen von Personen werden in KAPITÄLCHEN geschrieben.
- Eingaben von der Tastatur und Ausgaben auf den Bildschirm werden in `Schreibmaschinenschrift` wiedergegeben.
- Hinsichtlich der deutschen Rechtschreibung befinden wir uns in einem Übergangsstadium.
- Hinter UNIX-Kommandos folgt oft in Klammern die Nummer der Sektion des Referenz-Handbuches, in der das Kommando erläutert wird. 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. Die Übersetzung folgt in Klammern hinter dem deutschen Wort und ist nicht durch eine besondere Schrift gekennzeichnet.
- Suchen Sie die deutsche Übersetzung eines englischen oder französischen Fachwortes, so finden Sie einen Verweis im Sach- und Namensverzeichnis.
- An einigen Stellen wird auf Abschnitte aus dem C/C++-Skriptum verwiesen. Dies rührt daher, daß beide Skripten gemeinsam die Grundlage für ein Buch bilden. Ebenso fehlt in beiden Skripten das Kapitel über Hardware.
- UNIX verstehen wir immer im weiteren Sinne als die Familie der aus dem ursprünglich bei AT&T um 1970 entwickelten Unix abgeleiteten Betriebssysteme, nicht als geschützten Namen eines bestimmten Produktes.
- Wir geben möglichst genaue Hinweise auf weiterführende Dokumente im Netz. Der Leser sollte sich aber dessen bewußt sein, daß sich sowohl Inhalte wie Adressen (URLs) rasch ändern.
- Unter *Benutzer*, *Programmierer*, *System-Manager* usw. verstehen wir immer auch ihre weiblichen Erscheinungsformen, ohne dies hervorzuheben.
- Wir reden den Benutzer mit *Sie* an, obwohl unter Studenten und im Netz die Anrede mit *Du* üblich ist. Gegenwärtig erscheint uns diese Wahl passender. In Zukunft mag sich das ändern.

Inhaltsverzeichnis

Abbildungen

Programme

Rien n'est simple.
Sempé

1 Über den Umgang mit Computern

1.1 Was macht ein Computer?

Eine elektronische Datenverarbeitungsanlage, ein **Computer**, 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 Computer 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. Die 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: 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 Computerhilfe schneller und sicherer zu bewältigen. Computer finden sich nicht nur in Form grauer Kästen auf oder neben Schreibtischen, sondern auch versteckt in Fotoapparaten, Waschmaschinen, Heizungsregelungen, Autos und Telefonen.

Das Wort *Computer* stammt aus dem Englischen, wo es vor hundert Jahren eine Person bezeichnete, die berufsmäßig rechnete, zu deutsch ein Rechenknecht. Heute versteht man nur noch die Maschinen darunter. Das englische Wort wiederum geht auf lateinisch *computare* zurück, was berechnen, veranschlagen, erwägen, überlegen bedeutet. Die Franzosen sprechen vom *ordinateur*, die Spanier vom

[1] Schon geht es los mit den Fußnoten: Bei genauem Hinsehen gibt es Unterschiede zwischen Information, Nachricht und Daten, siehe Abschnitt 2.16 *Exkurs über Informationen*.

ordenador, dessen lateinischer Ursprung *ordo* Reihe, Ordnung bedeutet. Die Portugiesen – vielleicht um sich von den Spaniern abzuheben – verwenden das Wort *computador*. Die Schweden nennen die Maschine *dator*, analog zu *Motor*, die Finnen *tietokone*, was *Wissensmaschine* heißt. Hierzulande sprach man eine Zeit lang von *Elektronengehirnen*, etwas weniger respektvoll von *Blechbregen*. Wir ziehen das englische Wort *Computer* dem deutschen Wort *Rechner* vor, weil uns Rechnen zu eng mit dem Begriff der Zahl verbunden ist.

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. Der Begriff Informatik[2] ist rund vierzig Jahre alt, Computer gibt es seit fünfzig Jahren, Überlegungen dazu stellten Charles Babbage vor rund zweihundert und Gottfried Wilhelm Leibniz vor vierhundert Jahren an, ohne Erfolg bei der praktischen Verwirklichung ihrer Gedanken zu haben. 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. Insofern könnte man Informatik ohne Computer betreiben und hat das – unter anderen Namen – auch getan. Die Informatik beschränkt sich insbesondere *nicht* auf das Herstellen von Computerprogrammen. Der Computer hat jedoch die Aufgaben und die Möglichkeiten der Informatik ausgeweitet. Unter **Technischer Informatik** – gelegentlich Lötkolben-Informatik geheißen – 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.

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 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. Der-

[2]Die früheste uns bekannte Erwähnung des Begriffes findet sich in der Firmenzeitschrift SEG-Nachrichten (Technische Mitteilungen der Standard Elektrik Gruppe) 1957 Nr. 4, S. 171: Karl Steinbuch, Informatik: Automatische Informationsverarbeitung.

selbe 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. Der Verwandlungskunst sind natürlich Grenzen gesetzt, Kaffee kochen sie vorläufig nicht. 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.

Ebenso wie man die Größe von Massen, Kräften oder Längen mißt, werden auch **Informationsmengen** gemessen. Nun liegen Informationen in unterschiedlicher Form 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 der **Dualdarstellung** von Zahlen, bei der die Zahlen auf Summen von Potenzen zur Basis 2 zurückgeführt werden. Eine Dualdarstellung ist immer auch binär, das Umgekehrte gilt nicht.

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 Computer gebaut.

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. Auf dem Papier wird ein Byte oft durch ein Paar hexadezimaler Ziffern – ein **Hexpärchen** – wiedergegeben. Das **Hexadezimalsystem** – das Zahlensystem zur Basis 16 – wird uns häufig begegnen, in 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.

Der Computer verarbeitet die Informationen in Einheiten eines **Maschinenwortes**, das je nach der Breite der Datenregister des Prozessors ein bis 16 Bytes umfaßt. Der durchschnittliche Benutzer kommt mit dieser Einheit selten in Berührung; für den Assembler-Programmierer sind die Datentypen am einfachsten, die sich gerade in einem Maschinenwort darstellen lassen.

1.2 Woraus besteht ein Computer?

Der Benutzer sieht von einem Computer vor allem den **Bildschirm**[3] (screen, écran) und die **Tastatur** (keyboard, clavier), auch Hackbrett genannt. Diese beiden Geräte werden zusammen als **Terminal** (terminal, terminal) bezeichnet und stellen die Verbindung zwischen Benutzer und Computer dar. Mittels der Tastatur spricht der Benutzer zum Computer, auf dem Bildschirm erscheint die Antwort.

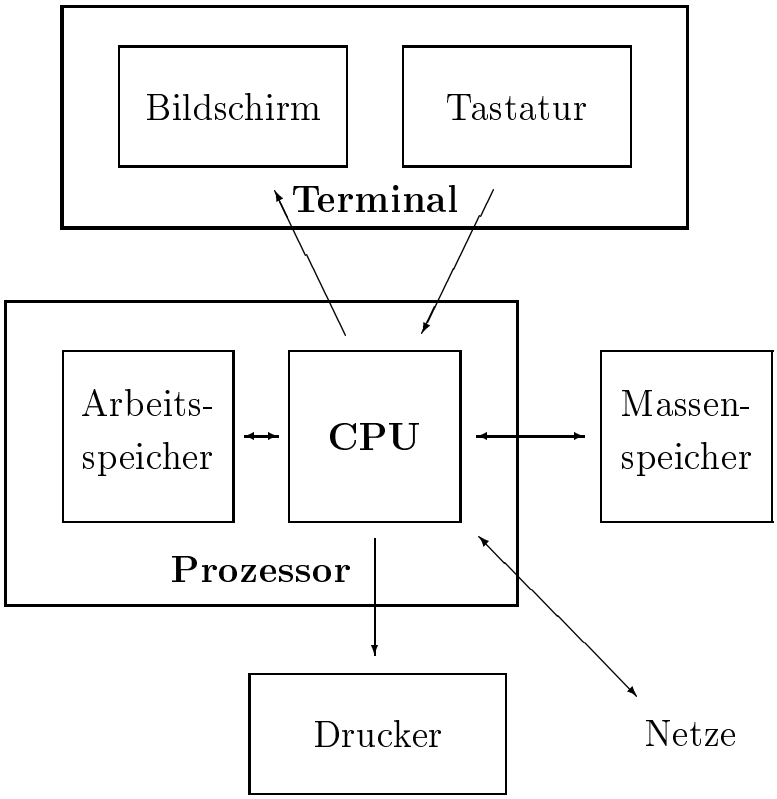

Abb. 1.1: Aufbau eines Computers

Der eigentliche Computer, die **Prozessoreinheit** (Zentraleinheit, central unit, 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, central processing unit, processeur central) und der **Arbeitsspeicher** (memory, mémoire centrale, mémoire secondaire).

Um recht in Freuden arbeiten zu können, braucht man noch einen **Massenspeicher** (mass storage, mémoire de masse), der seinen Inhalt nicht vergißt, wenn der Computer ausgeschaltet wird. Nach dem heutigen Stand der Technik arbeiten die meisten Massenspeicher mit magnetischen Datenträgern ähnlich wie Ton- oder Videobandgeräte. Tatsächlich verwendeten die ersten Personal Computer Tonbandkassetten. Weit verbreitet sind scheibenförmige magnetische Datenträger

[3] Aus der Fernsehtechnik kommend wird der Bildschirm oft Monitor genannt. Da dieses Wort hier nicht ganz trifft und auch ein Programm bezeichnet, vermeiden wir es.

in Form von **Disketten** (floppy disk, disquette) und **Festplatten** (hard disk, disque dur).

Disketten, auch Schlappscheiben genannt, werden nach Gebrauch aus dem **Laufwerk** (drive, dérouleur) des Computers herausgenommen und im Schreibtisch vergraben oder mit der Post verschickt. Festplatten verbleiben in ihrem Laufwerk.

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

Was um den eigentlichen Computer (Prozessoreinheit) herumsteht, wird als **Peripherie** bezeichnet. Die peripheren Geräte sind über **Schnittstellen** (Datensteckdosen, 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** (operating system, système d'exploitation) wie UNIX.

1.3 Was muß man wissen?

Ihre ersten Gedanken werden darum kreisen, wie man dem Computer vernünftige Reaktionen entlockt. Sie brauchen keine Angst zu haben: durch Tastatureingaben (außer Kaffee und ähnlichen Programming Fluids) ist ein Computer nicht zu zerstören. Zum Arbeiten mit einem Computer muß man drei Dinge lernen:

- den Umgang mit der **Hardware**[4],
- den Umgang mit dem **Betriebssystem**,
- den Umgang mit einem **Anwendungsprogramm** (application program, logiciel d'application), zum Beispiel einer Textverarbeitung.

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 klargeworden zu sein.

[4]Wir wissen, daß wir ein deutsch-englisches Kauderwelsch gebrauchen, aber wir haben schon so viele schlechte Übersetzungen der amerikanischen Fachwörter gelesen, daß wir der Deutlichkeit halber teilweise die amerikanischen Wörter vorziehen. Oft sind auch die deutschen Wörter mit unerwünschten Assoziationen befrachtet. Wenn die Mediziner lateinische Fachausdrücke verwenden, die Musiker italienische und die Gastronomen französische, warum sollten dann die Informatiker nicht auch ihre termini technici aus einer anderen Sprache übernehmen dürfen? Die Gallier sind da streng: Es ist bei Strafe verboten, in der Öffentlichkeit von Software zu reden. Man hat Logiciel zu sagen.

Die **Hardware** (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. Wallenstein spricht sie zu Max Piccolomini. Was sich hart im Raume stößt, gehört zur Hardware, was leicht beieinander wohnt, die Gedanken, ist **Software** (logiciel). Die Gedanken stecken in den Programmen und den Daten. Mit Worten von René Descartes ("cogito ergo sum") könnte man die Software als res cogitans, die Hardware als res extensa ansehen, wobei keine ohne die andere etwas bewirken kann. Er verstand unter der res cogitans allerdings nicht nur das Denken, sondern auch das Bewußtsein und die Seele und hätte jede Beziehung zwischen einer Maschine und seiner res cogitans abgelehnt.

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 Computer aus. Wir bezeichnen diese Kombination als **System**. Andere 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 UNIX).

Bekannte Betriebssysteme sind MS-DOS und Windows 95 bzw. NT von Microsoft sowie IBM OS/2 für IBM-PCs und ihre Verwandtschaft, VMS für die VAXen der Digital Equipment Corporation (DEC) sowie die UNIX-Familie für eine ganze Reihe von mittleren Computern verschiedener Hersteller.

Um eine bestimmte Aufgabe zu erledigen – um einen Text zu schreiben oder ein Gleichungssystem zu lösen – braucht man noch ein **Anwendungsprogramm**. Dieses kauft man fertig, zum Beispiel ein Programm zur Textverarbeitung oder zur Tabellenkalkulation, oder schreibt es selbst. In diesem Fall muß man eine **Programmiersprache** (programming language, langage de programmation) beherrschen. Die bekanntesten Sprachen sind BASIC, COBOL, FORTRAN, PASCAL und C/C++. Es gibt mehr als tausend[5].

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

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

[5]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 Computer. 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.

Bei den Büchern muß man zwischen **Lehrbüchern** (Einführungen, Tutorials, Primers, Guides) und **Nachschlagewerken** (Referenz-Handbücher, Reference Manuals) unterscheiden. Lehrbücher führen durch das Wissensgebiet, treffen eine Auswahl, werten oder diskutieren und verzichten auf Einzelheiten. Nachschlagewerke sind nach Stichwörtern geordnet, beschreiben alle Einzelheiten und helfen bei allgemeinen Schwierigkeiten gar nicht. Will man wissen, welche Werkzeuge UNIX zur Textverarbeitung bereit hält, braucht man ein Lehrbuch. Will man hingegen wissen, wie man den Editor `vi(1)` veranlaßt, nach einer Zeichenfolge zu suchen, so schlägt man im Referenz-Handbuch nach. Auf UNIX-Systemen ist das UNIX-Referenz-Handbuch online verfügbar, siehe `man(1)`.

Die Einträge in den Referenz-Handbüchern sind knapp gehalten. Bei einfachen Kommandos wie `pwd(1)` oder `who(1)` sind sie dennoch auf den ersten Blick verständlich. Zu Kommandos wie `vi(1)`, `sh(1)` oder `xdb(1)`, die umfangreiche Aufgaben erledigen, gehören schwer verständliche Einträge, die voraussetzen, daß man die wesentlichen Züge des Kommandos bereits kennt. Diese Kenntnis vermitteln **Einzelwerke**, die es zu einer Reihe von UNIX-Kommandos gibt, siehe Anhang O *Literatur*.

Ohne Computer bleibt das Bücherwissen trocken und abstrakt. Man sollte daher die Bücher in der Nähe eines Terminals lesen, so daß man sein Wissen sofort ausprobieren kann[6]. Das Durcharbeiten der Übungen gehört dazu, auch wegen der Erfolgserlebnisse.

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

Electronic Information besteht aus Mitteilungen in den Computernetzen. Das sind Bulletin Boards (Schwarze Bretter), Computerkonferenzen, Electronic Mail, Netnews, Veröffentlichungen, die per Anonymous-FTP kopiert werden, 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 Tip sind die FAQ-Listen (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.

[6]Es heißt, 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 daher nicht als Ersatz für die klassischen Wege des Lernens gelten.

Das Zusammenwirken von Büchern oder Zeitschriften mit Electronic Information sieht vielversprechend aus. Manchen Computerbüchern liegt eine Diskette oder CD bei. Das sind statische Informationen ohne Möglichkeit zum Dialog mit den Urhebern. Wir haben einen FTP-Server `ftp.ciw.uni-karlsruhe.de` eingerichtet, auf dem ergänzende Informationen verfügbar sind. Auf der WWW-Seite `http://www.ciw.uni-karlsruhe.de/technik.html` haben wir – in erster Linie für unseren eigenen Gebrauch – viele Verweise (Hyperlinks, URLs) zu den Themen dieses Buchs gesammelt, die zumindest als Einstieg verwendet werden können. Unsere Email-Anschrift steht im Impressum des Buches.

Es gibt **Lernprogramme** zu Hardware, Betriebssystemen und Anwendungsprogrammen. Man könnte meinen, daß sich gerade der Umgang mit dem Computer mit Hilfe des Computers lernen läßt. Moderne Computer mit **Hypertext**[7], bewegter farbiger Grafik, Dialogfähigkeit und Tonausgabe bieten tatsächlich Möglichkeiten, die dem Buch verwehrt sind. Der Aufwand für ein Lernprogramm, das diese Möglichkeiten ausnutzt, ist allerdings beträchtlich, und deshalb sind manche Lernprogramme nicht gerade ermunternd. Es gibt zwar Programme – sogenannte Autorensysteme – die das Schreiben von Lernsoftware erleichtern, aber Arbeit bleibt es trotzdem. Auch gibt es vorläufig keinen befriedigenden Ersatz für Unterstreichungen und Randbemerkungen, mit denen einige Leser ihren Büchern eine persönliche Note geben. Erst recht ersetzt ein Programm nicht die Ausstrahlung eines guten Pädagogen.

Über den modernen Wegen der Wissensvermittlung hätten wir 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 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*[8] als Antwort.

1.4 Wie läuft eine Sitzung ab?

Die Arbeit mit dem Computer vollzieht sich meist im Sitzen vor einem Terminal und wird daher **Sitzung** (session) genannt. Mittels der Tastatur teilt man dem Computer seine Wünsche mit, auf dem Bildschirm antwortet er. Diese Ar-

[7] Hypertext ist ein Text, bei dem Sie erklärungsbedürftige Wörter mit der Maus 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 Ted Nelson in den USA geprägt.

[8] siehe Anhang H *Slang im Netz*: Read The Fantastic Manual

beitsweise wird **interaktiv** genannt und als (Bildschirm-)**Dialog** bezeichnet, zu deutsch Zwiegespräch. 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 dazu. Der Bildschirm ist ein naher Verwandter des Fernsehers.

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 Mensch für Sie erledigt, der Systemverwalter oder **System-Manager**. Sie sollten seine Freundschaft suchen[9].

Nach dem Einschalten lädt der Computer sein Betriebssystem, er bootet, wie man so sagt. **Booten** heißt eigentlich Bootstrappen und das hinwiederum, sich an den eigenen Stiefelbändern oder Schnürsenkeln (bootstraps) aus dem Sumpf der Unwissenheit herausziehen wie weiland der Lügenbaron KARL FRIEDRICH HIERONYMUS FREIHERR VON MÜNCHHAUSEN an seinem Zopf[10]. Zu Beginn kann der Computer nämlich noch nicht lesen, muß aber sein Betriebssystem vom Massenspeicher lesen, um lesen zu können.

Ist dieser heikle Vorgang erfolgreich abgeschlossen, gibt der Computer einen **Prompt** 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 Computer auf Ihre Eingaben wartet. Der Prompt wird auch Systemanfrage, Bereitzeichen oder Eingabeaufforderung genannt. Können Sie nachempfinden, warum wir Prompt sagen?

Nun dürfen Sie in die Tasten greifen. Bei einem Mehrbenutzersystem erwartet der Computer als erstes Ihre **Anmeldung**, das heißt die Eingabe des Namens, unter dem Sie der System-Manager eingetragen hat. Auf vielen Anlagen gibt es den Benutzer `gast` oder `guest`. Außer bei Gästen wird als nächstes die Eingabe eines Passwortes verlangt. Das **Passwort** (password, mot de passe) ist der Schlüssel zum Computer. Es wird auf dem Bildschirm nicht wiedergegeben. Bei der Eingabe von Namen und Passwort sind keine Korrekturen zugelassen, Groß- und Kleinschreibung wird unterschieden. War Ihre Anmeldung in Ordnung, heißt der Computer Sie herzlich willkommen und promptet wieder. Die Arbeit beginnt. Auf einem PC geben Sie beispielsweise `dir` ein, auf einer UNIX-Anlage `ls`. Jede Eingabe wird mit der **Return-Taste** (auch mit Enter, CR oder einem geknickten Pfeil nach links bezeichnet) abgeschlossen[11].

Zum Eingewöhnen führen wir eine kleine Sitzung durch. Suchen Sie sich ein freies UNIX-Terminal. Betätigen Sie ein paar Mal die Return- oder Enter-Taste. Auf die Aufforderung zur Anmeldung (`login`) geben Sie den Namen `gast` oder `guest` ein, Return-Taste nicht vergessen. Ein Passwort ist für diesen Be-

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

[10] 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

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

nutzernamen nicht vonnöten. Es könnte allerdings sein, daß auf dem System kein Gast-Konto eingerichtet ist, dann müssen Sie den System-Manager fragen. Nach dem Willkommensgruß des Systems geben wir folgende UNIX-Kommandos ein (Return-Taste!) und versuchen, ihre Bedeutung mithilfe des UNIX-Referenz-Handbuchs, Sektion (1) näherungsweise zu verstehen:

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

Falls auf dem Bildschirm links unten das Wort `more` erscheint, betätigen Sie die Zwischenraum-Taste (space bar). `more(1)` ist ein Pager, ein Programm, das einen Text seiten- oder bildschirmweise ausgibt.

Die Grundform eines **UNIX-Kommandos** ist (ähnlich wie bei MS-DOS):

```
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 Filenamen oder andere Informationen, die das Kommando benötigt, oben der Verzeichnisname `/bin`. Bei den Namen der 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 gelegentlich 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(1)` 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 (button) namens `exit`. Einen PC dürfen Sie selbst ausschalten, ansonsten erledigt das wieder der System-Manager. Das Ausschalten des Terminals einer Mehrbenutzeranlage hat für den Computer keine Bedeutung, die Sitzung läuft weiter!

Merke: Für UNIX und C/C++ sind große und kleine Buchstaben verschiedene Zeichen. Ferner sind die Ziffer 0 und der Buchstabe O auseinanderzuhalten.

1.5 Wo schlägt man nach?

Wenn es um Einzelheiten geht, sind die zu jedem UNIX-System gehörenden und einheitlich aufgebauten **Referenz-Handbücher** – auf Papier oder Bildschirm – die wichtigste Hilfe[12]. Sie gliedern sich in folgende **Sektionen**:

- 1 Kommandos und Anwendungsprogramme
- 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 Fileformate
- 5 Vermischtes (z. B. Filehierarchie, Zeichensätze)
- 6 Spiele
- 7 Gerätefiles
- 8 Systemverwaltung
- 9 Glossar

Subroutinen sind in diesem Zusammenhang vorgefertigte Funktionen für eigene Programme, Standardfunktionen 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 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 Eintragungen zu den Kommandos oder Stichwörtern sind wieder gleich aufgebaut:

- Name (Name des Kommandos, Zweck)
- Synopsis, Syntax (Gebrauch des Kommandos)
- Remarks (Anmerkungen)
- Description (Beschreibung des Kommandos)
- Return Value (Rückgabewert nach Programmende)
- Examples (Beispiele)
- Hardware Dependencies (hardwareabhängige Eigenheiten)
- Author (Urheber des Kommandos)
- Files (vom Kommando betroffene Files)
- See Also (ähnliche oder verwandte Kommandos)

[12] Real programmers don't read manuals, behauptet das Netz.

- Diagnostics (Fehlermeldungen)
- Bugs (Mängel, soweit bekannt)
- Caveats, Warnings (Warnungen)
- International Support (Unterstützung europäischer Absonderlichkeiten)

Bei vielen Kommandos finden sich nur Name, Synopsis und Description. Der Sinn oder Nutzen des Kommandos wird verheimlicht; deshalb versuchen wir, diesen Punkt zu erhellen. Was hilft die Beschreibung eines Schweißbrenners, wenn Sie nicht wissen, was und warum man schweißt? Am Fuß jeder Handbuch-Seite steht das Datum der Veröffentlichung des Eintrags. Schlagen Sie unter `pwd(1)` und `time(2)` nach.

Einige Kommandos oder Standardfunktionen haben keinen eigenen Eintrag, sondern sind mit anderen zusammengefaßt. So findet man das Kommando `mv(1)` unter der Eintragung für das Kommando `cp(1)` oder die Standardfunktion `gmtime(3)` bei der Standardfunktion `ctime(3)`. In solchen Fällen muß man das Sachregister, den Index des Handbuchs befragen.

Mittels des Kommandos `man(1)` holt man die Einträge aus dem gespeicherten Referenz-Handbuch (On-line-Manual, man-pages) auf den Bildschirm oder Drucker. Das On-line-Manual sollte zu den auf dem System vorhandenen Kommandos passen, während das papierne Handbuch älter sein kann. Versuchen Sie folgende Eingaben:

```
man time
man 2 time
man man
man man | col -b | lp
```

Die Zahlenangabe bei der zweiten Eingabe bezieht sich auf die gewünschte Sektion. Die letzte Eingabezeile druckt die Handbuchseiten zum Kommando `man(1)` auf dem Default-Drucker aus (fragen Sie vorsichtshalber Ihren Arzt oder Apotheker oder besser noch Ihren System-Manager, für das Drucken gibt es viele Wege). Drucken Sie sich aber nicht das ganze Handbuch aus, die meisten Seiten braucht man selten oder nie.

1.6 Warum verwendet man Computer (nicht)?

Philosophische Interessen sind bei Ingenieuren häufig eine Alterserscheinung, meint der Wiener Computerpionier Heinz Zemanek. Wir glauben, das nötige Alter zu haben, um dann und wann das Wort *warum* in den Mund nehmen oder in die Tastatur hacken zu dürfen. Sehr 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 nach der Umstellung erfordert das Computersystem eine ständige Pflege. Einige Arbeiten gehen mit

Computerhilfe schneller von der Hand, dafür verursacht der Computer selbst Arbeit. Auf Dauer sollte eine Ersparnis herauskommen, aber die Erwartungen sind oft überzogen.

Nach drei bis zehn Jahren Betrieb ist ein Computersystem 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 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 Computer 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 dreißig Jahren. Große 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 Computer.

Noch eine Warnung ist angebracht. Die Arbeit wird durch den Computer nur selten einfacher. Mit einem Bleistift können die meisten umgehen, die Benutzung eines Texteditors erfordert in jedem Fall eine **Einarbeitung** und die Ausnutzung aller Möglichkeiten eines leistungsfähigen Textsystems eine lange Einarbeitung und ständige **Weiterbildung**. Ein Schriftstück wie das vorliegende wäre vor dreißig Jahren nicht am Schreibtisch herzustellen gewesen; heute ist das mit Computerhilfe kein Hexenwerk, setzt aber eine eingehende Beschäftigung mit mehreren Programmen voraus.

Man darf nicht vergessen, daß der Computer ein Hilfsmittel, 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, die im Lauf der Zeit ausgebügelt wird, sondern eine grundsätzliche Eigenschaft. Die Fähigkeit zur Verantwortung setzt die **Willensfreiheit** voraus und diese beinhaltet den eigenen Willen. Ein Computer, der anfängt, einen eigenen Willen zu entwickeln, ist ein Fall für die Werkstatt.

Der Computer soll den Menschen ebensowenig ersetzen wie ein Hammer die

Hand ersetzt, sondern den Menschen ergänzen. Das hört sich banal an, aber manchmal ist die Aufgabenverteilung zwischen Mensch und Computer schwierig zu erkennen. Und es ist bequem, die Entscheidung samt der Verantwortung dem Computer zuzuschieben. Es gibt auch Aufgaben, bei denen der Computer einen Menschen vielleicht ersetzen kann – wenn nicht heute, dann künftig – aber dennoch nicht soll. Nehmen wir zwei Extremfälle. Wenn ich die Telefonnummer 0721/19429 anrufe, 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 Computer als Lehrer aus? Darf ein Computer Schüler oder Studenten prüfen? Soll ein Arzt eine Diagnose vom Computer stellen lassen? Hat der Computer als Spielpartner einen Einfluß auf die seelische Entwicklung seines Benutzers? Ist ein Computer 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 Computer? Und wenn der Präsident zwar entscheidet, sich aber auf die Auskünfte seines Computers verlassen muß? Wer ist dann wichtiger, der Präsident oder sein Computer?

Je besser die Computer 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 Computer in andere Sprachen übersetzen lassen, gelten als stilistisch mangelhaft. Bei Meinungsverschiedenheiten über personenbezogene Daten hat zunächst einmal der Computer 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 Computer, die bei Störfällen unmittelbar zu spüren ist – sei es, daß der Computer streikt oder daß der Strom ausfällt. Da gibt es Augenblicke, in denen sich die System-Manager fragen, warum sie nicht Minnesänger oder Leuchtturmwärter (oder beides, wie OTTO) 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 Computer 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.

Vermutlich wird es uns mit den Computern ähnlich ergehen wie mit der Elektrizität: wir werden uns daran gewöhnen. Wie man für Stromunterbrechungen eine Petroleumlampe und einen Campingkocher bereithält, sollte man für Computerausfälle etwas Papier, einen Bleistift und ein gutes, zum Umblättern geeignetes Buch zurücklegen.

Seid einig, einig, einig!
Schiller, Tell

2 UNIX

Dieses Kapitel erläutert das Betriebssystem UNIX samt seinen Familienangehörigen (AIX, HP-UX, LINUX, SINIX, Solaris, ULTRIX usw.). Das zugehörige Referenz-Handbuch oder Online-Manual ist eine unerläßliche Begleitlektüre.

2.1 Grundbegriffe

2.1.1 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 ein 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 Vorspanns 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.

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 einem File 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 MS-DOS ist ein MS-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]

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

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

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

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 eine Aufgabe. 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, das jeweilige Programm zu Ende zu bringen. Ist die Zeitspanne (im Millisekundenbereich) abgelaufen, beendet das Betriebssystem das Programm vorläufig und reiht es wieder in die Warteschlange ein. Nach Bedienung aller anstehenden Programme beginnt das Betriebssystem wieder beim ersten, so daß bei den Benutzern der Eindruck mehrerer gleichzeitig laufender Programme entsteht. Dieser Vorgang läßt sich durch eine gleichmäßig rotierende **Zeitscheibe** veranschaulichen, von der jedes Programm 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 (Apple System 7, MS-Windows) verwalten zwar auch eine Warteschlange von Programmen, vollenden aber einen Auftrag, ehe der nächste an die Reihe kommt. Die Programme können sich **kooperativ** zeigen und ihren Platz an der Sonne freiwillig räumen, um ihren Mitbewerbern eine Chance zu geben; das Betriebssystem erzwingt dies jedoch nicht. Versucht ein nicht-kooperatives Programm, die größte Primzahl zu berechnen, warten die Mitbenutzer lange. Noch einfachere Betriebssysteme (MS-DOS) richten nicht einmal eine Warteschlange ein.

Den Algorithmus zur Verteilung der Prozessorzeit (scheduling algorithm) kann man verfeinern. So gibt es Programme, die wenig Zeit beanspruchen, diese aber sofort haben möchten (Terminaldialog), andere brauchen mehr Zeit, aber nicht sofort (Hintergrundprogramme). Ein Programm, 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-File) 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 **File-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 ein File eine zusammengehörige Menge von Daten, die er über den Filenamen anspricht. Daß die Daten eines Files physikalisch über mehrere, nicht zusammenhängende Bereiche auf der Festplatte verstreut sein können, geht nur das Betriebssystem etwas an. Ein File kann sogar über mehrere Platten, unter Umständen auf mehrere Computer verteilt sein. Im schlimmsten Fall existiert das File, mit dem der Benutzer zu arbeiten wähnt, überhaupt nicht, sondern wird aus Teilen verschiedener Files bei Bedarf zusammengesetzt. Beim Arbeiten mit Datenbanken kommt das vor. Zum Benutzer hin sehen alle UNIX-File-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 **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 2.13 *Echtzeit-Erweiterungen*.

Nach der Anzahl der scheinbar gleichzeitig bearbeiteten Aufgaben – 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. Ein Multi-User-System enthält vor allem Vorrichtungen, die verhindern, daß sich die Benutzer in die Quere kommen (Benutzerkonten, Zugriffsrechte an Files).

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 **Netzwerk-Betriebssysteme**, die mehrere über ein Netz verteilte Prozessoren wie einen einzigen Computer verwalten, im Gegensatz zu Netzwerken 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. MS-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 Netzwerk-Betriebssystem, das auf vernetzten PCs anstelle von MS-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 MS-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 MS 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 MS-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 2.12 *Systemverwaltung* 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 MS-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 – siehe die Seitennumerierung zu Anfang des Buches – verwenden, 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 Entstehungsgeschichte ist dennoch ungewöhnlich. 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, 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), der bereits viele Ideen vorwegnahm, aber für die damaligen Hardware- und Programmiermöglichkeiten wohl etwas zu komplex 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.

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, 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, u. a. auf den HP 9000/7*. Eine der Portierungen auf PCs hieß XENIX und machte vor einigen Jahren 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.

[3]Eine authentische Zusammenfassung findet sich in The Bell System Technical Journal, Vol. 57, July-August 1978, Nr.6, Part 2, p. 1897 - 2312

[4]Die PDP 11 hatte einen Adressraum von 64 KByte. Ein heutiger PC/AT hat 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.

Im Jahr 1991 hat AT&T die UNIX-Geschäfte in eine Tochtergesellschaft namens Unix System Laboratories (USL) ausgelagert, mit der der amerikanische Netzwerkhersteller Novell 1992 ein gemeinsames Unternehmen Univel gegründet hat. Anfang 1993 schließlich hat Novell USL übernommen. Ende 1993 hat Novell den Namen UNIX der Open Software Foundation vermacht. Der Handel geht weiter.

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. Ein ähnliches Ziel schwebt auch **HAX** vor, das mit einem einfachen Basissystem anfängt und sich durch Skalierbarkeit auszeichnet. Skalierbar heißt ausbaufähig, an die Bedürfnisse anpaßbar.

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** läuft bereits auf einigen Anlagen (z. B. 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.

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.

Das amerikanische Institute of Electrical and Electronics Engineers (IEEE) hat seit 1986 eine Sammlung von Standards namens **POSIX** (Portable Operating System Interface for Computer Environments) geschaffen, die die grundsätzlichen Anforderungen an UNIX-Systeme beschreibt. 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 selbst ist also kein Betriebssystem.

Neben diesen kommerziellen UNIXen gibt es mehr oder weniger freie Abkömmlinge. An mehreren 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, LINUX, FreeBSD und NetBSD. 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 2.14 *GNU is not UNIX*. Der erste Betriebssystem-Kernel namens **Hurd** kam 1996 heraus. Viel aus dem GNU-Projekt findet sich auch bei LINUX wieder.

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):

- `http://www.pasc.org/abstracts/posix.htm`
- `http://www.freebsd.org/`
- `http://wwww.netbsd.org/`
- `http://www.openbsd.org/`
- `http://www.linux.org/`
- `http://plan9.bell-labs.com/plan9/`
- `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/mach/public/www/mach.html`
- `http://www.cs.utah.edu/projects/flexmach/mach4/html/Mach4-proj.html`
- `http://www.gnu.org/` (Free Software Foundation)
- `http://www.camb.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 voneinander ab. Dennoch sind die Unterschiede gering im Vergleich zu den Unterschieden zwischen grundsätzlich fremden Systemen. Weltweit laufen zur Zeit etwa ein bis zwei Millionen Installationen.

Bei aller Verwandschaft der UNIXe ist trotzdem 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[5]. Aber: ein IBM-Betriebssystem läuft nicht auf einem DEC-Rechner und

[5]Real programmers use IBM OS/370.

umgekehrt. UNIX hingegen stammt von einer Firma, die nicht als Computerhersteller aufgefallen ist, und läuft auf den Maschinen vieler Hersteller. 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 Treiberprogramme 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-Files** beschrieben sind, die Speicherplatz und Prozessorzeit verbrauchen. Stimmen die Eintragungen in diesen Files nicht, gibt es Störungen. Und meist ist an einer Störung ein File mehr beteiligt, als man denkt. Die Konfigurations-Files sind meist 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 File-Hierarchie, die Umlenkung von Ein- und Ausgabe, Pipes, der Kommando-Interpreter, das Ansprechen der Peripherie als Files, leistungs- und erweiterungsfähige Werkzeuge und Dienstprogramme (fertige Programme zum Erledigen häufig vorkommender Aufgaben). Man muß den Weitblick der Väter von UNIX bewundern.

Die Stärken von UNIX liegen in der Programmierumgebung, in der Kommunikation und in der inzwischen etwas zurückgebliebenen Textverarbeitung. Schwächen von UNIX sind das Fehlen von Grafik- und Datenbankfunktionen sowie eine nur mittlere Sicherheit. Manchen ist die **Benutzeroberfläche** zu spartanisch[6]. 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, ohne andere Prozesse zu stören? Die Vielzahl und der Einfallsreichtum der Programmierer, die an UNIX mitgearbeitet haben und noch weiterarbeiten, hat stellenweise zu einer etwas unübersichtlichen und den Anfänger verwirrenden Fülle von Werkzeugen geführt. Statt *einer* Shell gibt es gleich ein halbes Dutzend Geschmacksrichtungen. Nach heutiger Erkenntnis hat UNIX auch Schwächen theoretischer Art, siehe Andrew S. Tanenbaum.

UNIX gilt als schwierig im Vergleich zu MS-DOS. In dieser Allgemeinheit teilen wir die Meinung nicht. Für den Benutzer ist UNIX eher einfacher, weil es mehr kann, zuverlässiger ist und viele Dinge selbsttätig erledigt. Auch für den Anwen-

[6] Gegen eine kleinen Aufpreis hilft das X Window System aber diesem Mangel ab.

dungsprogrammierer ist UNIX einfacher, insbesondere was die Speicherverwaltung angeht (für UNIX steht der Speicher zusammenhängend zur Verfügung, und wenn er nicht reicht, wird geschwoppt). Das System-Management hingegen ist schwieriger. An einer Aufgabe wie Electronic Mail oder Druckerausgabe sind ein Dutzend Files beteiligt, die zusammenarbeiten müssen.

UNIX ist sicherer als MS-DOS oder ähnliche Betriebssysteme. Den Reset-Knopf brauchen wir 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 einige hundert **Dienstprogramme** (utility, utilitaire) zur Verfügung. Dienstprogramme werden mit dem Betriebssystem geliefert, gehören aber nicht zum Kern, sondern haben den Rang von Anwendungsprogrammen. Sie erledigen immer wieder und überall vorkommende Arbeiten wie Anzeige von Fileverzeichnissen, 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 (`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 nicht. UNIX rechnet mit dem mündigen Bürger. Ein gewisser Gegensatz zur MS-Windows-Welt ist zu erkennen.

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 und sperrt sie ein.

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 (Abb. 2.1). Im Keller steht die **Hardware**. Darüber sitzt im Erdgeschoß der **UNIX-Kern** (kernel, noyau), der mit der Hardware über die Treiberprogramme und mit den höheren Etagen über die Systemaufrufe (system call, fonction système) verkehrt. Außerdem enthält der Kern die Prozessverwaltung und das File-System. Der betriebsfähige Kern ist ein einziges Programm, das im File-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 **An-**

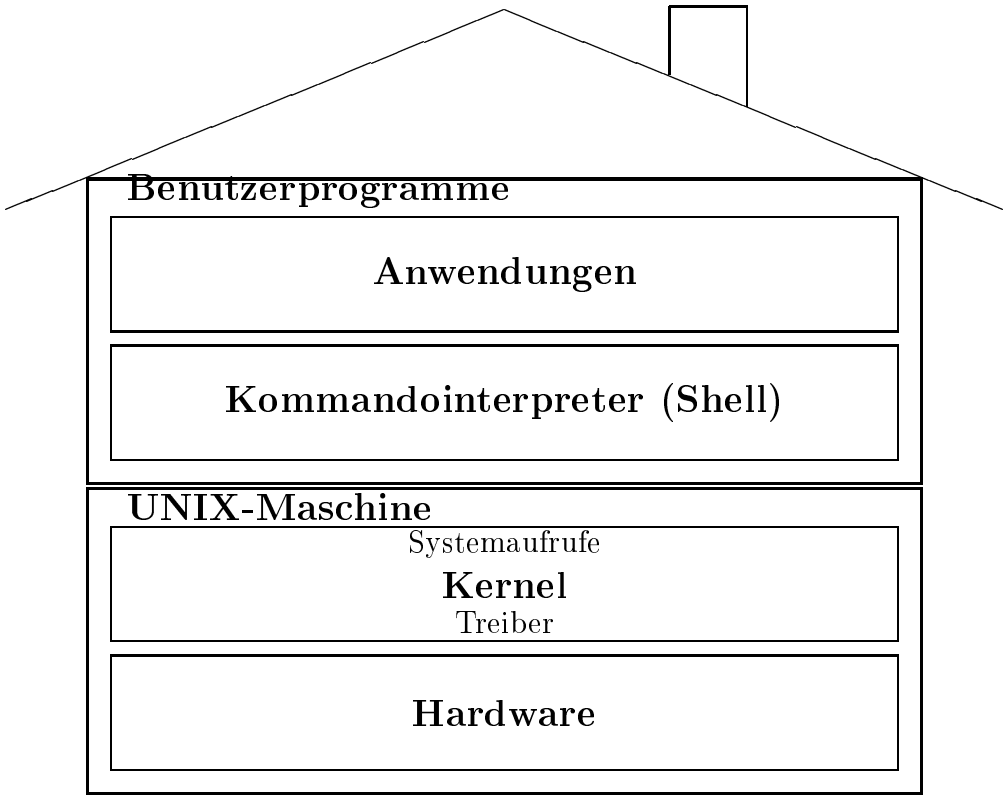

Abb. 2.1: Schematischer Aufbau von UNIX

wendungsprogramme, 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 File-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 O *Literatur* 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.

2.3 Prozesse

2.3.1 Was ist ein Prozess?

Wir müssen – zumindest vorübergehend – unterscheiden zwischen einem Programm und einem Prozess, auch Task genannt. Ein Programm *läuft* nicht, sondern ruht als File im File-System. 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 Verbindung hält.

Ein UNIX-Prozess besteht im Arbeitsspeicher aus drei Teilen: einem **Code-Segment** (auch Text-Segment genannt, obwohl aus unlesbarem Maschinencode bestehend), einem **Benutzerdaten-Segment** und einem **Systemdaten-Segment**. Er bekommt eine eindeutige Nummer, die **Prozess-ID** (PID). Das Code-Segment wird bei der Erzeugung des Prozesses beschrieben und ist 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 Files. Durch die Verteilung der Rechte wird verhindert, daß ein wildgewordener Prozess das ganze System lahmlegt[7]. 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 so aus:

```
  UID   PID  PPID  C    STIME TTY      TIME COMMAND
 root     0     0  0  Jan 22  ?        0:04 swapper
 root     1     0  0  Jan 22  ?        1:13 init
 root     2     0  0  Jan 22  ?        0:00 pagedaemon
 root     3     0  0  Jan 22  ?        0:00 statdaemon
 root    31     1  0  Jan 22  ?        0:18 /etc/cron
 root    46     1  0  Jan 22  console  0:00 sleep
 root    48     1  0  Jan 22  console  0:00 sleep
 root    59     1  0  Jan 22  ?        0:00 /etc/delog
```

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

```
 wualex1  1279     1  0  Feb  4  console  0:00 -ksh [ksh]
    root  1820     1  0 00:48:00 tty0p2   0:00 /etc/getty
      lp  1879     1  0 00:51:17 ?        0:00 lpsched
    root  2476     1  0 13:02:40 tty1p0   0:00 /etc/getty
    root  2497     1  0 13:04:31 tty0p1   0:00 /etc/getty
    root  2589     1  0 13:31:34 tty1p1   0:00 /etc/getty
 wualex1  2595  1279  4 13:32:39 console  0:00 -ksh [ksh]
 wualex1  2596  2595 21 13:32:40 console  0:00 ps -ef
 wualex1  2597  2595  3 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
- C Prozessorbenutzung (für Scheduler Priority)
- STIME Start Time des Prozesses
- TTY Kontroll-Terminal des Prozesses
- TIME Dauer der Ausführung des Prozesses
- COMMAND Name des zugehörigen Programmes

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

Alle Prozesse, die von einem gemeinsamen Vorfahren abstammen, gehören zu einer **Prozessgruppe**. Sie verkehren mit der Außenwelt über dasselbe **Kontrollterminal** 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

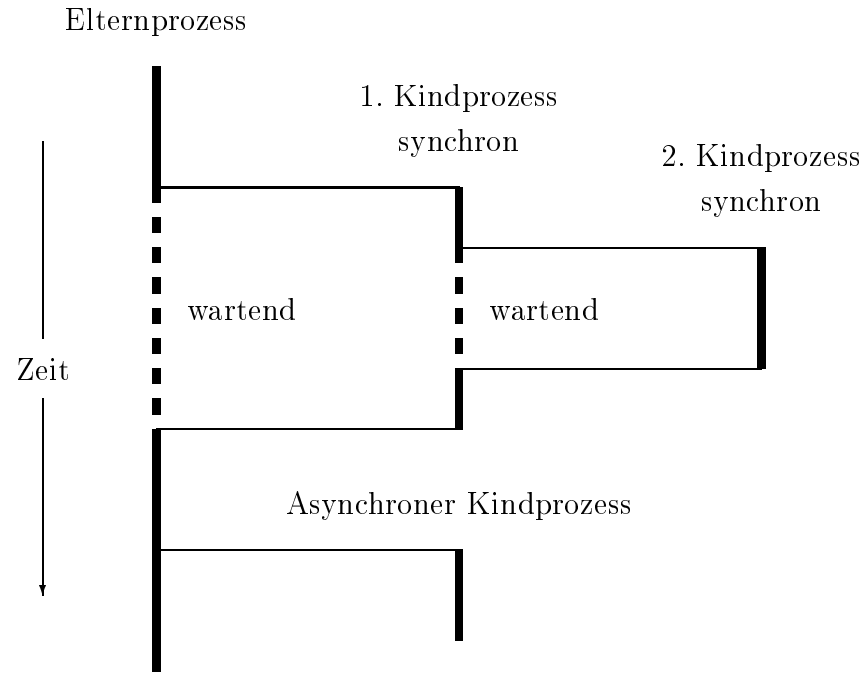

Abb. 2.2: Synchrone und asynchrone Prozesse

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 (Abb. 2.2). 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.

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 prozessid` in den Hintergrund schicken. Umgekehrt holt ihn `fg prozessid` 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 Shellscript zu verpacken. Die Ausgabe eines nogehupten Programmes geht automatisch in ein File namens `nohup.out`, falls sie nicht umgelenkt wird.

Starten wir das Shellscript `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 Kontrollterminal 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. Der Standardwert eines Vordergrundprozesses ist 20. Mit dem Aufruf:

```
nice myprocess
```

setzt man den `nice`-Faktor des Prozesses `myprocess` auf 30 herauf, seine Priorität im System wird schlechter. Mittels:

```
nice -19 myprocess
```

bekommt der `nice`-Faktor den schlechtesten Wert (39). Größere Zahlen werden als 19 interpretiert. Negative Werte 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. 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`, das die Priorität eines laufenden Prozesses zu ändern ermöglicht. Weitere Überlegungen zur Priorität stehen im Abschnitt 2.13 *Echtzeit-Erweiterungen*.

2.3.5 Dämonen

2.3.5.1 Was ist ein Dämon?

Das griechische Wort $\delta\alpha\iota\mu\omega\nu$ 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 Kontrollterminal 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 Shellscript `/etc/rc` ins Leben gerufen worden sind, weisen in der Prozessliste als Kontrollterminal ein Fragezeichen auf. Mittlerweile ist der Start der Dämonen beim Booten etwas komplizierter geworden und auf eine Kette von Shellscripts verteilt.

2.3.5.2 Dämon mit Uhr (cron)

Im System waltet ein Dämon namens `cron(1M)`. Der schaut jede Minute[8] in `/var/spool/cron/crontabs` und `/var/spool/cron/atjobs` nach, ob zu dem jeweiligen Zeitpunkt etwas für ihn zu tun ist. Die Files 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 ein, die regelmäßig wiederkehrend vorgenommen werden sollen. In unserer Anlage werden beispielsweise jede Nacht zwischen 4 und 5 Uhr sämtliche Sitzungen abgebrochen, Files 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. Auch für das Ziehen von Backup-Kopien wichtiger Files auf eine zweite Platte ist die Tabelle 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 meinem Homeverzeichnis gestartet. Weitere Zeitformate sind möglich, siehe Handbuch unter `at(1)`.

[8] Die UNIX-Uhr zählt Sekunden seit dem 1. Januar 1970, 00:00 Uhr GMT

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 `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 siehe Abschnitt 2.7.12 *Druckerausgabe*.

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. Es darf immer nur ein Internet-Dämon laufen. Weiteres siehe Abschnitt 3.7 *Netzdienste im Überblick*.

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 Mail-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 Delivery Agent) vorgeschaltet.

2.3.6 Interprozess-Kommunikation (IPC)

2.3.6.1 IPC mittels Files

Mehrere Prozesse können auf dasselbe File auf dem Massenspeicher lesend und schreibend zugreifen, wobei es am Benutzer liegt, das Durcheinander in Grenzen zu halten. Hiervon wird oft bei sogenannten **Lock-Files** (engl. to lock = zuschließen, versperren) Gebrauch gemacht. Beipielsweise darf nur ein `elm(1)`-Prozess

zum Verarbeiten der Electronic Mail pro Benutzer existieren. Also schaut `elm(1)` beim Aufruf nach, ob ein Lock-File `/tmp/mbox.username` existiert. Falls nein, legt `elm(1)` ein solches File 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 das Lock-File gelöscht. Wird der Prozess gewaltsam abgebrochen, bleibt das Lock-File erhalten und täuscht einen `elm(1)`-Prozess vor. Das Lock-File von Hand löschen.

Die Kommunikation über Files 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**[9], Pipeline oder Fließband genannt und durch den senkrechten Strich (ASCII-Nr. 124) bezeichnet:

```
cat filename | more
```

`cat(1)` schreibt das File `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 das File einlesen (anders als in MS-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 einglagerten 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:

```
mknod mypipe p
```

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

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

```
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 F *UNIX-Signale* nachzulesen oder im Handbuch unter `signal(2)`. Ein Prozess reagiert in dreierlei Weise auf ein Signal:

- er beendet sich (Default[10]) 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. Shellscript (was das ist, folgt in Abschnitt 2.5.2 *Shellscripts*) auf ein Signal reagiert, legt man in Shellscripts 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

```
trap "Kommandoliste" Signalnummer
```

Empfängt die das Script 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

[10] 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 "echo Arrivederci; exit" 0
```

im File `/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 Shellscript namens `traptest`:

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

Programm 2.1 : Shellscript traptest mit Signalbehandlung

Setzen Sie die Zugriffsrechte mit `chmod 750 traptest`. Wenn Sie das Shellscript 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 Shellscript 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 Shellscript. 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 Shellscript 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 sollten nicht auf Dauer in der Prozesstabelle auftauchen. Notfalls booten.

Die weiteren Mittel zur Kommunikation zwischen Prozessen gehören in die Systemprogrammierung und gehen über den Bereich 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 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 außer der Reihe zugegriffen werden.

2.3.6.6 Semaphore

Semaphore 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 dasselbe File 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. Darunter 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 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 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.

2.3.8 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. Passwörter dürfen nicht zu einfach sein, eine Kombination aus sechs Buchstaben und zwei Ziffern oder Satzzeichen ist gut. 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. Wenn der Prompt (das 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)

2.4 Files

2.4.1 Filearten

Eine **Datei** (file, fichier) ist eine Menge zusammengehöriger Daten, auf die mittels eines Filenamens zugegriffen wird. Wir bevorzugen das Wort **File**, weil das Wort Datei – das auch nicht rein deutsch ist – mit nicht immer zutreffenden Assoziationen wie Daten und Kartei behaftet ist. Das englische Wort geht auf lateinisch *filum* = Draht zurück und bezeichnete früher eine auf einem Draht aufgereihte Sammlung von Schriftstücken. Man kann ein File als einen Datentyp höherer Ordnung auffassen. Die Struktur ist in dem File enthalten oder wird durch das schreibende bzw. lesende Programm einem an sich strukturlosen **Zeichenstrom** (byte stream) aufgeprägt. In UNIX ist das letztere der Fall.

Im UNIX-File-System kommen im wesentlichen drei Arten von Files vor: gewöhnliche Files (normales File, regular file, fichier regulair), Verzeichnisse (Katalog, directory, data set, folder, répertoire) und Gerätefiles (special device file, fichier spécial). **Gewöhnliche Files** enthalten Meßdaten, Texte, Programme. **Verzeichnisse** sind Listen von Files, die wiederum allen drei Gruppen angehören können. **Gerätefiles** sind eine Besonderheit von UNIX, das periphere Geräte (Laufwerke, Terminals, Drucker) formal als Files ansieht. Die gesamte Datenein- und -ausgabe erfolgt über einheitliche Schnittstellen. Alle Gerätefiles finden sich im Geräteverzeichnis `/dev`, das insofern eine Sonderstellung einnimmt (*device* = Gerät).

Darüber hinaus unterscheidet man bei gewöhnlichen Files noch zwischen **lesbaren** und **binären** Files. Lesbare Files (text file) enthalten nur lesbare ASCII-Zeichen und können auf Bildschirm oder Drucker ausgegeben werden. Binäre Files (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. Intelligente Lese- oder Druckprogramme unterscheiden beide Arten und weigern sich, binäre Files zu verarbeiten. Auch bei Übertragungen im Netz wird zwischen ASCII-Files und binären Files unterschieden, siehe Abschnitt 3.9 *File-Transfer*.

2.4.2 File-System – Sicht von unten

Alle Daten sind auf dem Massenspeicher in einem **File-System** abgelegt, wobei auf einer Platte ein oder mehrere File-Systeme eingerichtet sind. In modernen Anlagen kann ein File-System auch über mehrere Platten gehen (spanning). Jedes UNIX-File-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 Abb. 2.3. Der **Boot-Block** enthält Software zum Booten des Systems und muß der erste Block im File-System sein. Er wird nur im `root`-System gebraucht – also einmal auf der ganzen Anlage – ist aber in jedem File-System vorhanden. Er wird beim Einrichten des File-Systems mittels der Kommandos `mkfs(1M)` oder `newfs(1M)` angelegt.

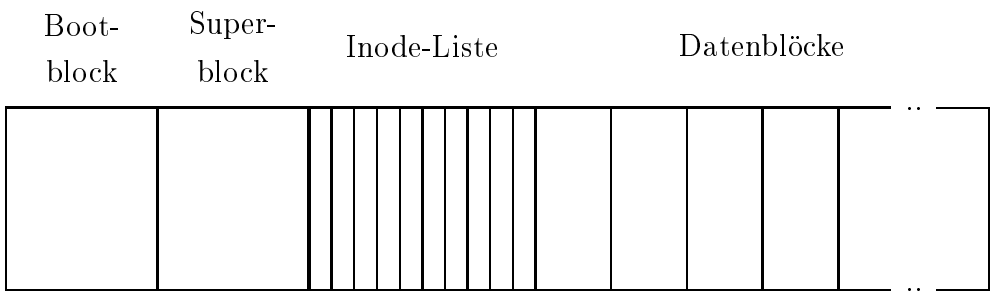

Abb. 2.3: UNIX-File-System, untere Ebene

Der **Super-Block** wird ebenfalls bei der Einrichtung des File-Systems geschrieben und enthält Informationen zu dem File-System als Ganzem wie die Anzahl der Datenblöcke, die Anzahl der freien Datenblöcke und die Anzahl der Inodes.

Die **Inode-Liste** enthält alle Informationen zu den einzelnen Files außer den File-Namen. Zu jedem File gehört eine Inode mit einer eindeutigen **Nummer**. Einzelheiten siehe Abschnitt 2.4.7 *Inodes und Links*. Die Files 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 Files einschließlich der Verzeichnisse untergebracht. Die Blockgröße beträgt 512 Bytes oder ein Vielfaches davon. Große Blöcke erhöhen die Schreib- und Lesegeschwindigkeit, verschwenden aber bei kleinen Files Speicherplatz, weil jedem File mindestens ein Block zugeordnet wird.

Ein MS-DOS-Filesystem beginnt mit dem Boot-Sektor, gefolgt von mehreren Sektoren mit der File Allocation Table (FAT), dem Root-Verzeichnis und schließlich den Sektoren mit den Files, also ähnlich, wenn auch wegen des fehlenden Inode-Konzeptes nicht gleich. Es gibt weitere File-Systeme, wobei mit größer werdenden Massenspeichern die Zugriffsgeschwindigkeit eine immer wichtigere Rolle spielt.

2.4.3 File-System – Sicht von oben

Selbst auf einer kleinen Anlage kommt man leicht auf zehntausend Files. Da muß man Ordnung halten. Unter UNIX werden zum Benutzer hin alle Files in einer File-Hierarchie, einer Baumstruktur angeordnet, siehe Abb. 2.4. An der Spitze steht ein Verzeichnis namens `root`[11], das nur mit einem Schrägstrich bezeichnet wird. Dieses Verzeichnis enthält einige gewöhnliche Files und vor allem weitere

[11] `root` ist der Name des obersten Verzeichnisses und zugleich der Name des System-Managers. Das Verzeichnis `root` ist genau genommen gar nicht vorhanden, sondern bezeichnet eine bestimmte Adresse auf dem Massenspeicher, auf der der Filebaum 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.

Verzeichnisse. Die Hierarchie kann viele Stufen enthalten, der Benutzer verliert die Übersicht eher als der Computer. In der Wurzel des Filebaumes, dem `root`-Verzeichnis, finden sich folgende Unterverzeichnisse:

- `bin` (UNIX-Kommandos und -Programme)
- `dev` (Gerätefiles)
- `etc` (Konfigurationsfiles, Kommandos zur Systemverwaltung)
- `homes` (Home-Verzeichnisse der Benutzer)
- `lib` (Bibliotheken)
- `lost+found` (Fundbüro)
- `mnt` (Mounting Point)
- `opt` (optionale Software, Compiler, Editoren)
- `sbin` (Kommandos zur Systemverwaltung)
- `tmp` (temporäre Files)
- `usr` (Fortsetzung von `bin`)
- `var` (Verschiedenes)

und noch einige spezielle Verzeichnisse und Files für die Systemverwaltung. Diese Gliederung ist auf allen UNIX-Systemen anzutreffen. Mit den wachsenden Anforderungen an UNIX (X11, Netze) ändert sich von Zeit zu Zeit die Einteilung etwas. Das Verzeichnis mit den Homes der Benutzer, hier `homes`, heißt woanders `home`, `user`, `users` oder auch `mnt`, ist aber auf jeden Fall vorhanden. Ein **Mounting Point** ist ein leeres Verzeichnis, in das die Wurzel eines weiteren Filesystems eingehängt werden kann. Auf diese Weise läßt sich der ursprüngliche Filebaum nahezu unbegrenzt erweitern. Das Verzeichnis `/usr` enthält einige wichtige Unterverzeichnisse:

- `adm` (Systemverwaltung, Accounting)
- `bin` (UNIX-Kommandos und -Programme)
- `contrib` (Beiträge des Computerherstellers)
- `lib` (Bibliotheken und Kommandos)
- `local` (lokale Kommandos)
- `mail` (Mailsystem)
- `man` (Referenz-Handbuch)
- `news` (News, Nachrichten an alle)
- `tmp` (weitere temporäre Files)
- `spool` (Drucker-Spoolsystem, `cron`-Files)

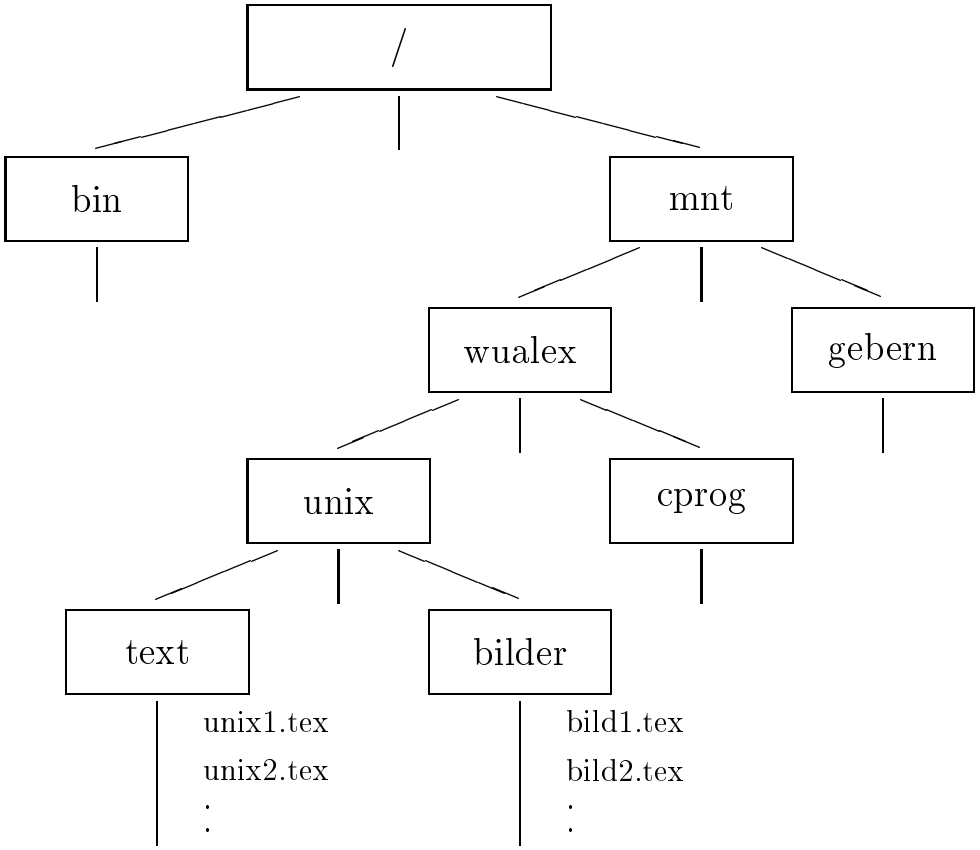

Abb. 2.4: UNIX-Filehierarchie

und weitere Unterverzeichnisse für spezielle Zwecke.

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 Device
- `rdsk` Platte als Character Device (raw, zeichenorientiert)
- `rmt` Magnetband als Character Device
- `tty` Kontrollterminal eines Prozesses
- `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).

Nach der Anmeldung landet der Benutzer in seinem **Home-Verzeichnis** (home directory, répertoire principal). Dort darf er nach Herzenslust Files und Unterverzeichnisse anlegen und löschen. Das Kommando `ls(1)` listet ein Verzeichnis auf und ist das UNIX-Kommando mit den meisten Optionen. Die Form `ll(1)` ist gleichwertig `ls -l`. Sollte sie auf Ihrem System nicht verfügbar sein, läßt sie sich durch ein Alias oder ein Shellscript verwirklichen, ebenso andere Varianten von `ls(1)`.

Das Home-Verzeichnis ist zu Beginn das **Arbeits-Verzeichnis** oder aktuelle Verzeichnis (working directory, répertoire courant, répertoire de travail), dessen Files unmittelbar über ihren Namen ohne die bei `root` beginnende Verzeichniskette verfügbar sind. Man kann jedes Unterverzeichnis seines Home-Verzeichnisses vorübergehend zum Arbeits-Verzeichnis machen, auch andere Verzeichnisse, sofern man dazu berechtigt ist. Mit `cd(1)` wechselt man in ein anderes Arbeits-Verzeichnis. Nach einer Faustregel soll man ein Verzeichnis weiter unterteilen, wenn es mehr als 100 Eintragungen enthält. Das Kommando `pwd(1)` nennt das Arbeits-Verzeichnis, falls man die Orientierung verloren hat.

Der **Name** eines Files 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 Shellscripts gebraucht. Umgekehrt zieht das Kommando `dirname(1)` aus einem absoluten Filenamen alle Vorderglieder (= Namen von Verzeichnissen) heraus.

Filenamen dürfen 14 Zeichen[12] 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

[12]Es gibt UNIX-Systeme, die 255 Zeichen erlauben. Wird bei der Einrichtung des Filesystems festgelegt. Vierzehn Zeichen sind heutzutage knapp.

sind nur der Schrägstrich, der als Trennzeichen in der Pfadangabe dient, und das Zeichen ASCII-Nr. 0, das einen String abschließt. Filenamen sollten mindestens vier Zeichen lang sein, um die Gefahr einer Verwechslung mit UNIX-Kommandos zu verringern. Innerhalb eines Verzeichnisses darf ein Name nur einmal vorkommen; der gleiche Name in verschiedenen Verzeichnissen benennt verschiedene Files, zum Beispiel `/bin/passwd(1)` und `/etc/passwd(4)`. Bei Shellkommandos, die Filenamen als Argument benötigen, kann man in den Filenamen **Jokerzeichen** verwenden, siehe Abschnitt 2.5.1.1 *Kommandointerpreter*.

Die Verwendung von **Namenserweiterungen** (`file.doc`, `file.dat`, `file.bak`) oder **Kennungen** (extension) ist erlaubt, aber nicht so gebräuchlich wie unter MS-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 auch 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 und im Anhang.

Das jeweilige Arbeits-Verzeichnis wird mit einem Punkt bezeichnet, das unmittelbar übergeordnete Verzeichnis mit zwei Punkten (wie in MS-DOS). Das Kommando `cd ..` führt also immer eine Stufe höher in der Filehierarchie. Mit `cd ../..` kommt man zwei Stufen höher. Will man erzwingen, daß ein 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 Filenamen von System zu System unterschiedlich sind. MS-DOS gestattet beispielsweise nur acht Zeichen, dann einen Punkt und nochmals drei Zeichen. Ferner unterscheidet es nicht zwischen Groß- und Kleinschreibung. In der Macintosh-Welt sind Filenamen aus mehreren Wörtern gebräuchlich. Will man sicher gehen, so paßt man die Filenamen von Hand an, ehe man sich auf irgendwelche Automatismen der Übertragungsprogramme verläßt.

In den Home-Verzeichnissen werden einige Files vom System erzeugt und verwaltet. Diese interessieren den Benutzer selten. Ihr Name beginnt mit einem Punkt (dot), zum Beispiel `.profile`, daher werden sie von `ls(1)` nicht angezeigt. Gibt man `ls(1)` mit der Option `-a`, so erscheinen auch sie. Solche Files (dotfile) werden als **verborgen** (hidden) bezeichnet, sind aber in keiner Weise geheim.

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 MS-DOS ist unter UNIX nicht vorgesehen. Man hat es stets nur mit einer einzigen File-Hierarchie zu tun. Weitere File-Hierarchien – zum Beispiel auf Disketten oder Platten, lokal oder im Netz – können vorübergehend in die File-Hierarchie des Systems eingehängt werden. Dabei wird das Wurzel-Verzeichnis des einzuhängenden File-Systems auf ein Verzeichnis, einen Mounting Point des root-File-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 Files und Unterverzeichnisse so lange nicht

verfügbar, wie ein File-System eingehängt ist. Man nennt das **mounten**, Kommando `mount(1M)`. Mountet man das File-System eines entfernbaren Datenträgers (Diskette) und entfernt diesen, ohne ihn vorher mittels `umount(1M)` unzumounten (zu unmounten?), gibt es Ärger. Beim Mounten treten Probleme mit den Zugriffsrechten auf. Deshalb gestatten die System-Manager dem Benutzer diese Möglichkeit nur auf besonderen Wunsch. File-Systeme können auch über das Netz gemountet werden, siehe Network File System (NFS). Wir mounten beispielsweise sowohl lokale File-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 Filesystemen unverschlüsselt durch die Kabel gehen und mitgelesen werden können.

Auf einen entfernbaren Datenträger – ob Diskette oder Band ist unerheblich – kann auf zwei Arten zugegriffen werden. Ist auf dem Datenträger mittels `newfs(1M)` oder `mkfs(1M)` ein Filesystem eingerichtet, muß dieses in das beim Systemstart geöffnete `root`-Filesystem an irgendeiner Stelle mit dem Kommando `mount(1M)` in ein vorhandenes Verzeichnis gemountet werden und wird damit vorübergehend ein Zweig von diesem. Ist der Datenträger dagegen nur formatiert (Kommando `mediainit(1)`), d. h. zum Lesen und Schreiben eingerichtet, ohne daß ein Filesystem angelegt wurde, so kann man mit den Kommandos `cpio(1)` oder `tar(1)` darauf zugreifen. Kommandos wie `cd(1)` oder `ls(1)` machen dann keinen Sinn, es gibt auch keine Inodes. Der Datenträger ist über den Namen des Gerätefiles anzusprechen, beispielsweise `/dev/rfd.0` oder `/dev/rmt/0m`, mehr weiß das System nicht von ihm. Wer sowohl unter MS-DOS wie unter UNIX arbeitet, mache sich den Unterschied zwischen einem Wechsel des Laufwerks (A, B, C ...) unter MS-DOS und dem Mounten eines File-Systems sorgfältig klar.

2.4.4 Zugriffsrechte

Auf einem Mehrbenutzersystem ist es untragbar, daß jeder Benutzer mit allen Files alles machen darf. Jedes File einschließlich der Verzeichnisse wird daher in UNIX durch einen Satz von neun **Zugriffsrechten** (permission, droit d'accès) geschützt.

Die Benutzer werden eingeteilt in den **Besitzer** (owner, propriétaire), seine **Gruppe** (group, groupe) und die Menge der sonstigen Benutzer (ohne den Besitzer und seine Gruppe), auch **Rest der Welt** (others) genannt. Die Rechte werden ferner nach **Lesen** (read), **Schreiben** (write) und **Suchen/Ausführen** (search/execute) unterschieden. Bei einem gewöhnlichen File bedeutet execute Ausführen (was nur bei Programmen Sinn macht), bei einem Verzeichnis Durchsuchen. Jedes Zugriffsrecht kann nur vom Besitzer erteilt und wieder entzogen werden. Und natürlich – wie immer – vom System-Manager.

Der Besitzer eines Files ist zunächst derjenige, der es 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 ein File für andere lesbar machen, so reicht es nicht, dem File die entsprechende Leseerlaubnis zuzuordnen oder den Besitzer zu wechseln. Vielmehr müssen alle übergeordneten Verzeichnisse von `/` an lückenlos das Suchen gestatten. Das wird oft vergessen.

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

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

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 gar nichts machen. Alle übrigen (Welt minus Besitzer minus Gruppe) dürfen das File lesen, ändern und ausführen. Der Besitzer kann 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 Files. Es gibt noch weitere Formen des `chmod(1)`-Kommandos. Aus Sicherheitsgründen soll man die Zugriffsrechte möglichst einschränken. Will ein Benutzer auf ein File zugreifen und darf das nicht, wird er sich schon rühren. Mittels des Kommandos:

```
ls -l filename
```

erfährt man die Zugriffsrechte eines Files. Die Ausgabe sieht so aus:

```
-rw-r----- 1 wualex1 users 59209 May 15 16:21 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 2.4.7 *Inodes und Links*. Dann folgen Besitzer und Gruppe sowie die Größe des Files in Bytes. Das Datum gibt die Zeit des letzten schreibenden (verändernden) Zugriffs an. Schließlich der Name. Ist das Argument des Kommandos `ls(1)` der Name eines Verzeichnisses, werden die Files des Verzeichnisses in alphabetischer Folge aufgelistet.

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 lassen sich nicht trennen. Das Kommando zum Kopieren lautet:

```
cp originalfile copyfile
```

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

Die Default-Rechte werden mittels des Kommandos `umask(1)` im File `/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.

Gelegentlich möchte man einzelnen Benutzern Rechte erteilen, nicht gleich einer ganzen Gruppe. Während so etwas unter Windows NT vorgesehen ist, gehört es nicht zum Standard von UNIX. Unter HP-UX läßt sich jedoch einem File eine **Access Control List** zuordnen, in die Sonderrechte eingetragen werden, Näheres mittels `man 5 acl`.

Für das System ist ein Benutzer im Grunde nur ein Bündel von Rechten. Ob dahinter eine natürliche oder juristische Person, eine Gruppe von Personen oder ein Dämon steht, ist unwesentlich. Es gibt Betriebssysteme wie Windows NT oder Datenbanken wie Oracle, die stärker differenzieren – sowohl nach der Art der Rechte wie nach der Einteilung der Benutzer - aber mit diesem Satz von neun 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 System-Manager oder Administrator üblicherweise – ist an die Zugriffsrechte nicht gebunden. Wollen Sie Ihre höchst private Mail vor seinen Augen schützen, müssen Sie sie verschlüsseln.

Merke: Damit jemand auf ein File zugreifen kann, müssen zwei Bedingungen erfüllt sein:

- Er muß einen ununterbrochenen Suchpfad vom Root-Verzeichnis (/) bis zu dem File haben, und
- er muß die entsprechenden Rechte an dem File haben.

2.4.5 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 verbleiben und somit sofort verfügbar sind. Wir haben das Sticky Bit eine Zeitlang beim Editor `vi(1)` verwendet. Bei einem Verzeichnis führt das Sticky Bit dazu, daß nur noch der Besitzer eines darin eingeordneten Files dieses löschen oder umbenennen kann, auch wenn die übrigen Zugriffsrechte des Verzeichnisses diese Operationen für andere erlauben, Beispiel `/tmp`. Das Sticky Bit kann nur der Superuser vergeben.

Der Wert 2 ist das **Set-Group-ID-Bit**, der Wert 4 das **Set-User-ID-Bit**, auch Setuid-Bit oder 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, der `root` gehört und bei gesetztem suid-Bit auf Files zugreifen darf, die der `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.

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

```
-r-sr-xr-x   1 root  bin  112640 Nov 22  1989 /bin/passwd
```

Damit ist es möglich, daß jeder Benutzer sein Passwort in dem File `/etc/passwd(4)` ändern darf, ohne die Schreiberlaubnis für dieses File zu besitzen:

```
---r--r--r   1 root  other  3107  Dec 2  10:39 /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 Shellscripts, 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 das File `/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 schon Kap. **??** *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 File-Besitzers enthält, ist nicht direkt mittels eines Editors beschreibbar. Leider steht im Referenz-Handbuch, daß der Systemaufruf bei gewöhnlichen Files 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 System-Manager beauftragt daher den Dämon `cron(1M)` mit ihrer regelmäßigen Überwachung. Da das suid-Bit selten vergeben wird, könnte der System-Manager auch ein eingeschränktes `chmod`-Kommando schreiben und die Ausführungsrechte des ursprünglichen Kommandos eingrenzen.

Sticky Bit, suid-Bit und sgid-Bit werden beim Kommando `ls -l` durch Modifikationen der Anzeige der Zugriffsrechte kenntlich gemacht. Das Sticky-Bit ist an einem t bei dem execute-Recht für alle zu erkennen, suid-Bit und sgid-Bit an einem `s` bei den execute-Rechten für Besitzer beziehungsweise Gruppe.

2.4.6 Zeitstempel

Zu jedem UNIX-File gehören drei Zeitangaben, die Zeitstempel genannt und automatisch verwaltet werden:

- die Zeit des jüngsten lesenden Zugriffs (access time),
- die Zeit der jüngsten schreibenden Zugriff (modification time),
- die Zeit der jüngsten Änderung des Filestatus (status change time).

Der Filestatus umfaßt den Fileinhalt, die Zugriffsrechte und den Linkzähler. Ein Schreibzugriff ändert also zwei Zeitstempel. Bei Verzeichnissen gilt das Durchsuchen nicht als lesender Zugriff, Löschen oder Hinzufügen von Files gilt als schreibender Zugriff.

Das Kommando `ls -l` zeigt das Datum des jüngsten schreibenden Zugriffs an, mit `ls -lu` erfährt man das Datum des jüngsten lesenden Zugriffs, mit `ls -lc` das Datum der jüngsten Änderung des Status. Das folgende C-Programm gibt zu einem Filenamen alle drei Zeitstempel aus, 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;
}
```

Programm 2.2 : C-Programm zur Anzeige der Zeitstempel eines Files

Der Zeitpunkt der Erschaffung eines Files wird *nicht* festgehalten und ist auch aus technischer Sicht uninteressant. Änderungen an den Daten hingegen sind für Werkzeuge wie `make(1)` wichtig.

2.4.7 Inodes und Links

Die Verzeichnisse enthalten nur die Zuordnung File-Name zu einer File-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 ein File einschließlich der Startadresse und der Größe des Datenbereiches. Insbesondere sind dort die Zugriffsrechte und die Zeitstempel vermerkt. Einzelheiten sind im Handbuch unter `inode(5)` und `fs(5)` zu finden. Das Kom-

mando `ls -i` zeigt die Inode-Nummern an. Wie die Informationen der Inode in eigenen Programmen abgefragt werden, steht in Abschnitt 2.11.3 *Beispiel File-Informationen*.

Diese Zweiteilung in Verzeichnisse und Inode-Liste erlaubt eine nützliche Konstruktion, die in MS-DOS oder IBM-OS/2 bei aller sonstigen Ähnlichkeit nicht möglich ist. Man kann einem File sprich einer Inode-Nummer nämlich mehrere Filenamen, unter Umständen in verschiedenen Verzeichnissen, zuordnen. Das nennt man **linken**[13]. Das File, auf das mehrere Filenamen gelinkt sind, existiert nur einmal (deshalb macht es keinen Sinn, von einem Original zu reden), aber es gibt mehrere Zugangswege, siehe Abb. 2.5. Zwangsläufig gehören zu gelinkten Filenamen dieselben Zeitstempel und Zugriffsrechte, da den Namen nur eine einzige Inode zugrunde liegt. Das Kommando zum Linken zweier Filenamen lautet `ln(1)`:

```
ln oldname newname
```

Auf diese Weise spart man Speicher und braucht beim Aktualisieren nur ein einziges File zu berücksichtigen. Die Kopie eines Files mittels `cp(1)` hingegen ist ein eigenes File mit eigener Inode-Nr., dessen weiterer Lebenslauf unabhängig vom Original ist. Beim Linken eines Files wird sein **Linkzähler** um eins erhöht. Beim Löschen eines Links wird der Zähler herabgesetzt; ist er auf Null angekommen, wird der vom File belegte Speicherplatz freigegeben. Bei einem Verzeichnis hat

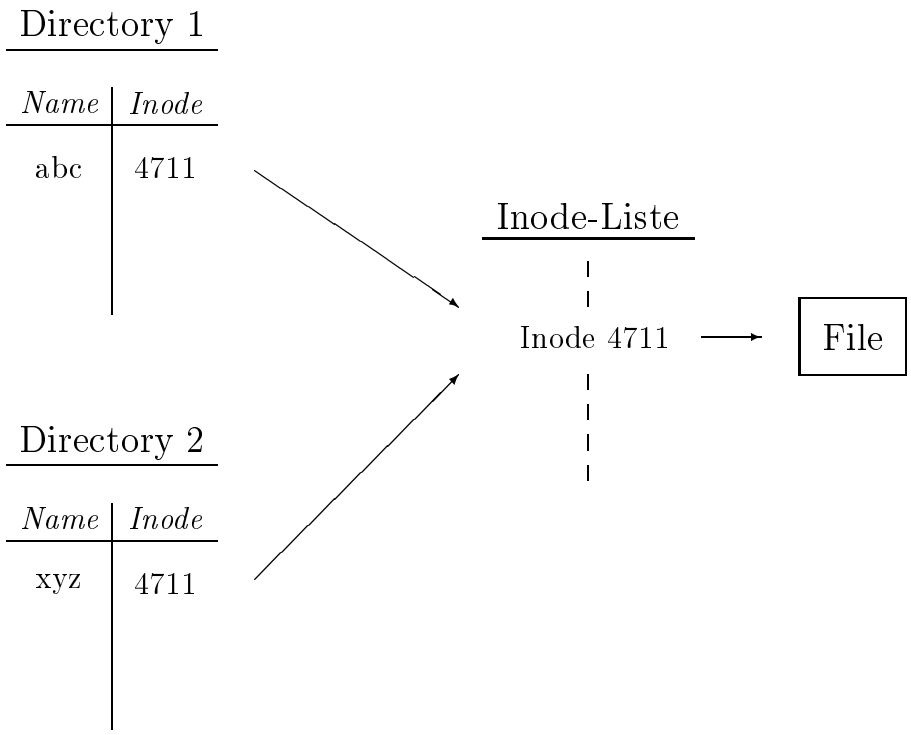

Abb. 2.5: Harter Link

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

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 Filesystems bei Unfällen. Als Benutzer kann man Verzeichnisse weder kopieren noch linken, sondern nur die in einem Verzeichnis versammelten Files. Old Root kann natürlich wieder mehr, siehe `link(1M)` und `link(2)`.

Dieser sogenannte **harte Link** kann sich nicht über die Grenze eines File-Systems erstrecken, auch falls es gemountet sein sollte. Der Grund ist einfach: jedes File-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** oder **weiche Links** über die Grenze eines File-Systems erstrecken und sind auch bei Verzeichnissen erlaubt und beliebt. Sie werden mit dem Kommando `ln(1)` mit der Option `-s` erzeugt. Ein weicher Link ist ein File mit eigener Inode-Nummer, das einen Verweis auf einen weiteren absoluten oder relativen Filenamen enthält, siehe Abb. 2.6. Das Kommando `ls -l` zeigt weiche Links folgendermaßen an:

```
lrwx------  1 wualex1  manager  4 Jun 2 17:13 scriptum -> unix
```

Das Verzeichnis `scriptum` ist ein weicher Link auf das Verzeichnis `unix`. Zugriffsrechte eines weichen Links werden vom System nicht beachtet, das Kommando `chmod(1)` wirkt auf das zugrunde liegende File, `rm(1)` glücklicherweise nur auf den Link. Weiteres siehe `ln(1)` unter `cp(1)`, `symlink(2)` und `symlink(4)`. Links

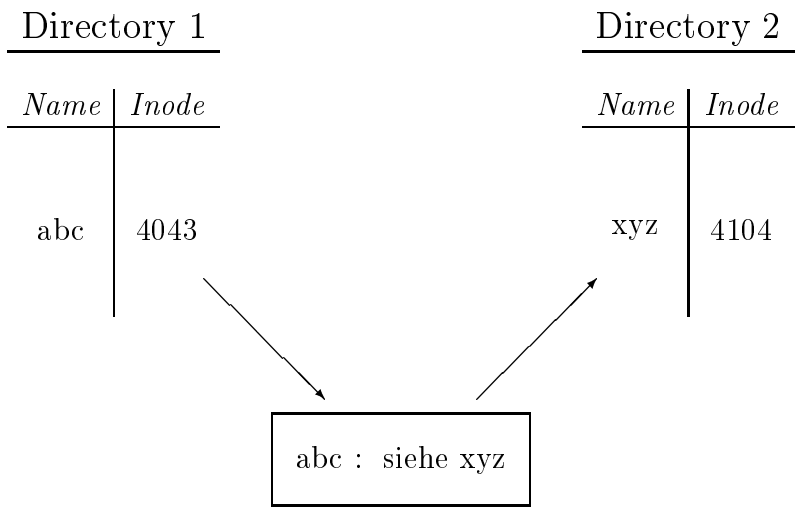

Abb. 2.6: Weicher, Symbolischer oder Soft Link

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 ein einziges File. Bei weichen Links wird auch eine Kette von Verweisen richtig verarbeitet. Insbesondere erkennt das Kopierkommando `cp(1)` die Links und verweigert ein Kopieren eines Files auf seinen 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 das dem Link zugrunde liegende File zielen. Wird einem weichen Link sein File weggenommen, besteht er weiter, Zugriffe über den Link auf das File sind erfolglos. Hat man die Wahl zwischen einem harten und einem weichen Link, so dürfte der harte geringfügig schneller im Zugriff sein.

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

Merke: Nach einem Kopiervorgang hat man zwei voneinander unabhängige Files, das Original und die Kopie. Nach einem Linkvorgang hat man zwei Namen für dasselbe File.

2.4.8 stdin, stdout, stderr

Drei Files 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 Files angesprochen. Andere Systeme kennen noch `stdaux` (Standard Auxiliary Device) und `stdprn` (Standard Printer Device).

Zu den File-Pointern `stdin`, `stdout` und `stderr` gehören die File-Deskriptoren 0, 1 und 2. **File-Pointer** sind Namen (genauer Namen von Pointern auf eine C-Struktur vom Typ FILE), **File-Deskriptoren** fortlaufende Nummern der für ein Programm geöffneten Files. Microsoft bezeichnet in MS-DOS die Deskriptoren als **Handles**. In Programmen wird durch einen `open`-Aufruf einem Filenamen ein File-Pointer oder ein File-Deskriptor zugeordnet, mit dem dann die weiteren Anweisungen arbeiten. Die UNIX-Systemaufrufe (2) verwenden File-Deskriptoren, die C-Standardfunktionen (3) File-Pointer. Beispiele finden sich im C-Programm 2.39 *File-Informationen*.

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.

Ein leeres File wird mit der Umlenkung `> filename`, mit `cat(1)` oder `touch(1)` angelegt. Zum Leeren eines Files kopiert man `/dev/null` dorthin.

Das Kommando `tee(1)` liest von `stdin`, schreibt nach `stdout` und gleichzeitig eine Kopie der Ausgabe in ein File, 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* gesprochen.

2.4.9 Schreiben und Lesen von Files

Files werden mit einem Editor, z. B. dem `vi(1)`, geschrieben (siehe Abschnitt 2.7.3 *Editoren*), von Compilern oder anderen Programmen erzeugt oder laufen einem über das Netz zu. Zum Lesen von Files auf dem Bildschirm stehen die Kommandos `cat(1)`, `more(1)`, `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 das File `filename` auf den Bildschirm. Die Pager `more(1)` und `pg(1)` arbeiten ähnlich, halten aber nach jeweils einer Bildschirmseite an. `view(1)` ist der Editor `vi(1)` im Lesemodus, `vis(1)` wandelt etwaige nicht sichtbare Zeichen in ASCII-Nummern um. Der Versuch, Files zu lesen, die etwas anderes als in Zeilen gegliederten Text enthalten, führt in manchen Fällen zu einem Blockieren des Terminals.

Will man sich den Inhalt eines beliebigen Files genau ansehen, so schreibt man mit `od(1)`, gegebenenfalls mit der Option `-c`, einen Dump nach `stdout`, bei Schwierigkeiten nützlich. Ein **Dump** ist eine zeichengetreue Wiedergabe des Speicher- oder Fileinhalts ohne jede Bearbeitung. Begnügt man sich mit dem Anfang oder Ende eines Files, leisten die Kommandos `head(1)` und `tail(1)` gute Dienste.

2.4.10 Archivierer (tar, gtar)

Files 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* zu diesem Thema. Mit der Verschlüsselung hängen weitere Fragen zusammen, die in Netzen eine Rolle spielen; im Abschnitt 3.11 *Electronic Mail* wird der Punkt nochmals aufgerollt.

Zum Aufbewahren oder Verschicken von ganzen Filegruppen ist es oft zweckmäßig, sie in ein einziges File zu verpacken. Diesem Zweck dient das Kommando `tar(1)`. Der Aufruf

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

stopft alle Files des Arbeits-Verzeichnisses, auf die das Namensmuster (Jokerzeichen!) zutrifft, in ein File `name.tar`, das als Archiv bezeichnet wird. Die Option `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 Files 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             # (im Verzeichnis /buch/unix)
tar -cvf unix.tar ./*.tex           # (im Verzeichnis /buch/unix)
tar -cvf unix.tar unix/*.tex        # (im Verzeichnis /buch)
tar -cvf unix.tar /buch/unix/*.tex  # (an beliebiger Stelle)
```

archivieren zwar dieselben Files, aber unter verschiedenen Namen, 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):

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

Zweckmäßig kopiert man das auszupackende 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. Ein `tar`-Archivfile kann mit einem beliebigen Packer verdichtet werden (erst archivieren, dann packen). Das ist im Netz üblich, um den Übertragungsaufwand zu verringern. Das GNU-Kommando `gtar(1)` archiviert und komprimiert bei entsprechender Option in einem Arbeitsgang:

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

2.4.11 Packer (compress, gzip)

Die meisten Files 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 Files. Das Standard-Kommando dafür ist `compress(1)`, ein jüngeres und wirkungsvolleres Kommando `gzip(1)` aus dem GNU-Projekt. Das urpsrüngliche File wird gelöscht, das verdichtete File bekommt die Kennung `.Z` oder `.gz`. Zum Verdünnen auf die ursprüngliche Konzentration ruft man `uncompress(1)` oder `gunzip(1)` mit dem Filenamen auf. Das Packen ist vollkommen umkehrbar[14]. Probieren Sie folgende Kommandofolge aus (`textfile` sei ein mittelgroßes Textfile):

```
cp textfile textfile1
cp textfile textfile2
ll textfile*
compress textfile1
gzip textfile2
```

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

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

Auch binäre Files lassen sich verdichten. Ein mehrfaches Verdichten ist nicht zu empfehlen. In der MS-DOS-Welt gibt es eine Vielzahl anderer Packprogramme, teils frei, teils gegen Bares.

2.4.12 Weitere Kommandos

Mit `mv(1)` benennt man ein File um und verschiebt es gegebenenfalls in ein anderes Verzeichnis, seine 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 Files eines Verzeichnisses oder eine Auswahl davon verschieben. Ebenso läßt sich ein Verzeichnis umbennen, ohne es zu verschieben. Das Kommando `mvdir(1M)` verschiebt ein Verzeichnis an eine andere Stelle in selben Filesystem und ist dem System-Manager vorbehalten, da bei unvorsichtigem Gebrauch geschlossene Wege innerhalb des Filebaums entstehen.

Zum **Löschen** von Files 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 Files auf einen Schlag löschen. Das Löschen eines Files mittels `rm(1)` erfordert die Schreiberlaubnis im zugehörigen Verzeichnis, aber keine Rechte am File selbst. Gelöscht wird zunächst **logisch**, d. h. die angesprochene Inode samt zugehörigem Speicherplatz wird freigegeben, die Bits bleiben noch auf der Platte, sind aber nicht mehr erreichbar. Erst bei Bedarf an freiem Speicherplatz werden die Bits überschrieben, womit die Daten auch **physikalisch** beseitigt sind. Deshalb wird bei hohen Anforderungen an die Sicherheit ein File zunächst überschrieben und dann gelöscht.

Ein mit `rm(1)` gelöschtes File kann *nicht* wiederhergestellt werden, anders als unter MS-DOS. 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 $*
```

Programm 2.3 : Shellscript saferm zum verzögerten Löschen von Files

Zum Leeren eines Files, ohne es zu löschen, verwendet man am einfachsten folgende Zeile:

```
> filename
```

Das File hat anschließend die Größe 0 Bytes. Eine andere Möglichkeit ist das Kopieren von `/dev/null` in das File.

Nun zu einem Dauerbrenner in der entsprechenden Gruppe der Netnews. Wie werde ich ein File mit einem absonderlichen Namen los? In UNIX-Filenamen 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 Filenamen, 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 Files, 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 Files in ein anderes Verzeichnis, ehe Sie dem Übel zu Leibe rücken. Das File `-abc`, dessen 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 Files im Arbeitsverzeichnis, fragt aber zuvor bei jedem einzelnen File um Erlaubnis. Das zweite Kommando ermittelt im Arbeitsverzeichnis das File 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 das widerspenstige File nur umbennen, sieht das Kommando so aus:

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

Filenamen mit einem Schrägstrich oder ASCII-Null kommt man so jedoch nicht bei. In diesem Fall kopiert man sämtliche gesunden Files 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 System-Manager 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 Files dienen `which(1)`, `whereis(1)` und `find(1)`. `which(1)` sucht nach ausführbaren Files (Kommandos), `whereis(1)` nach Kommandos, deren Quellfiles und man-Seiten. `type(1)` und `whence(1)` geben ähnliche Informationen:

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

Das Werkzeug `find(1)` ist vielseitig und hat daher eine umfangreiche Syntax:

```
find . -name vorwort.* -print
find . -name '*.conf' | xargs grep -i hallo
find $HOME -size |1000 -print
find / -atime +32 -print
```

Das Kommando der ersten Zeile sucht im Arbeits-Verzeichnis und seinen Unterverzeichnissen (rekursiv) nach Files mit dem Namen `vorwort.*` und gibt die Namen auf `stdout` aus. Eigentlich sollte man `vorwort.*` in Hochkommas (Apostrophe, Single Quotes) setzen, da das Jokerzeichen nicht von der Sitzungsshell,

sondern von `find(1)` ausgewertet werden soll, aber es funktioniert auch so. In der nächsten Zeile setzen wir die Quotes und schicken 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 Files gesucht, die größer als 1000 Blöcke (zu 512 Bytes) sind. Der vierte Aufruf sucht im ganzen File-System nach Files, 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 System-Manager benutzt es gern, um Ladenhüter aufzuspüren.

Ein Kommando wie MS-DOS `tree` zur Anzeige des Filebaumes gibt es in UNIX leider nicht. Deshalb hier ein Shellscript 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"
```

Programm 2.4 : Shellscript tree zur Anzeige der Filehierarchie

Die Zwischenräume und Tüttelchen sind wichtig; fragen sie bitte jetzt noch nicht nach ihrer Bedeutung. Schreiben Sie das Shellscript in ein File 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 System-Manager (nur er, wegen der Zugriffsrechte) verschafft sich mit:

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

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

2.4.13 Memo Files

- Unter UNIX gibt es gewöhnliche Files, Verzeichnisse und Gerätefiles.
- Alle Files sind in einem einzigen Verzeichnis- und Filebaum untergebracht, an dessen Spitze (Wurzel) das root-Verzeichnis steht.
- Jedes File oder Verzeichnis gehört einem Besitzer und einer Gruppe.
- Die Zugriffsrechte bilden eine Matrix von Besitzer - Gruppe - Rest der Welt und Lesen - Schreiben - Ausführen/Durchsuchen.

- Jedes File oder Verzeichnis besitzt eine Inode-Nummer. In der Inode stehen die Informationen über das File, in den Verzeichnissen die Zuordnung Inode-Nummer - Name.
- Ein harter Link ist ein weiterer Name zu einer Inode-Nummer. Ein weicher Link ist ein File mit einem Verweis auf ein anderes File oder Verzeichnis.
- Ein Filesystem kann in einen Mounting Point (leeres Verzeichnis) eines anderen Filesystems eingehängt (gemountet) werden.
- Ein Archivierprogramm wie `tar(1)` packt mehrere Files oder Verzeichnisse ins ein einziges File (Archiv).
- Ein Packprogramm wie `gzip(1)` verdichtet ein File ohne Informationsverlust (reversibel).

2.4.14 Übung Files

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?)
`users`	(dito, nur anders)
`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)
`ls /mnt`	(mnt-Verzeichnis, enthält die Home-Verzeichnisse)
`cat lsfile`	(lsfile lesen)
`news -a`	(alle News anzeigen)
`mail`	(Falls Ihnen Mail angezeigt wird, kommen Sie mit RETURN weiter.)
`mail root`	(Nun können Sie dem System-Manager einen Brief schreiben. Ende mit RETURN, `control-d`)
`mkdir privat`	(Verzeichnis erzeugen)
`cd privat`	(dorthin wechseln)
`cp /mnt/student/beispiel beispiel`	(Das File `/mnt/student/beispiel` ist bei uns ein kurzes, allgemein lesbares Textfile. Fragen Sie

Ihren System-Manager.)

`cat beispiel` (File anzeigen)

`head beispiel` (Fileanfang anzeigen)

`more beispiel` (File bildschirmweise anzeigen)

`pg beispiel` (File bildschirmweise anzeigen)

`od -c beispiel` (File als ASCII-Text dumpen)

`od -x beispiel` (File hexadezimal dumpen)

`file beispiel` (Filetyp ermitteln)

`file /bin/cat`

`whereis /bin/cat` (File suchen)

`ln beispiel exempel` (linken)

`cp beispiel uebung` (File kopieren)

`ls -i`

`mv uebung schnarchsack`
(File umbenennen)

`ls`

`pg schnarchsack`

`rm schnarchsack` (File 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 System-Manager nach dem üblichen Druckkommando)

Abmelden mit `exit`

2.5 Shells

2.5.1 Gesprächspartner

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 im File `/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 (Abb. 2.1). Er ist ein Programm, das für das Betriebssystem auf gleicher Stufe steht wie vom Anwender geschriebene Programme. Er ist ersetzbar, es dürfen auch mehrere Kommandointerpreter gleichzeitig verfügbar sein (aber nicht mehrere Betriebssysteme).

Unter MS-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 Shellscripts 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 Funktionen, weniger Fehler) ist die **tc-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 Shellscripts 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 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. Die Korn-Shell gibt es auch für MS-DOS-Rechner (siehe Abschnitt 2.15.5 *MKS-Tools und andere*), was die Austauschbarkeit des Kommandointerpreters unterstreicht. 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 MS-DOS beispielsweise `dir`. Das `dir` ent-

sprechende, externe UNIX-Kommando `ls(1)` hingegen ist ein eigenes Programm, das wie viele andere Kommandos vom Interpreter aufgerufen wird und irgendwo in der File-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 übliche Form eines **UNIX-** oder **Shell-Kommandos** sieht so aus:

```
command -options argument1 argument2 ....          (RETURN-Taste!)
```

Die **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. Unter **Argumenten** werden Filenamen oder Strings verstanden, soweit das Sinn macht. Die Reihenfolge von Optionen und Argumenten ist bei vielen Kommandos beliebig, aber da die UNIX-Kommandos auf Programmierer mit unterschiedlichen Vorstellungen zurückgehen, hilft im Zweifelsfall nur der Blick ins Referenz-Handbuch. Kommandoeingabe per Menü ist unüblich, aber machbar, siehe Programm 2.9 *Shellscript für ein Menü*. Die Verwendung einer Maus setzt erweiterte curses-Funktionen voraus, siehe Abschnitt 2.6.1.3 *Fenster (Windows), curses-Bibliothek*, oder das X Window System.

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

- Zuerst prüft die Shell, ob das Wort ein Aliasname ist (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 – ein File 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`.
- Das File sei gefunden und ein Shellscript (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 Shellscripts oder Programmes, hängt also davon ab, was sich der Programmierer gedacht hat. Ein guter Programmierer läßt den Benutzer nicht ganz im Dunkeln tappen.
- Das File sei gefunden, sei aber ein Textfile wie ein Brief oder eine Programmquelle. Dann bedauert die Shell, damit nichts anfangen zu können, d. h. sie sieht den Text als ein Shellscript 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 Shellscript und einem übersetzten Programm merkt sie schnell. Um sich ein Textfile anzusehen, gibt man ein entsprechendes Kommando (`more(1)`, `pg(1)` oder `view(1)`) mit dem Namen des Textfiles als Argument ein. In der Maus-und-Fenster-Welt ist die Denkweise anders, sofern Denken überhaupt noch nötig ist. `karlsruhe` war ein leeres File mit den Zugriffsrechten 777. Was hätten Sie als Shell damit gemacht?

In Filenamen 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 Filename wie

```
ab?c
```

trifft auf Files wie

```
ab1c  abXc  abcc  ab_c
```

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

```
ls abc*z
```

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

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

Der Stern allein bedeutet *alle Files* 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 Filenamen, 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**[15].

[15] 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.

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.

Die C-Shell und die Korn-Shell haben einen **History**-Mechanismus, der die zuletzt eingetippten Kommandos in einem File `.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 das History-File 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 `script(1)` gibt. Alle Bildschirmausgaben werden gleichzeitig in ein File `typescript` geschrieben, das man später lesen oder drucken kann. Die Wirkung von `script(1)` wird durch das shellinterne Kommando `exit` beendet. Wir verwenden `script(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 File-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 einem eigenen File 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 MS-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 (File-Ende, 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) bereit. Darin sind eine Reihe von 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 die Umgebung verändern und aus Programmen oder Shellscripts heraus abfragen.

Einige dieser Parameter werden von der Sitzungsshell 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. Die anderen Parameter gelten nur für die jeweilige Shell, sie werden nicht vererbt und als **Shell-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:/usr/contrib/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 Trennzeichen, der interne Feld-Separator, der 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 MS-DOS gibt es eine ähnliche Einrichtung, die ebenfalls mit dem Kommando `set` auf dem Bildschirm erscheint.

Zum Ändern oder Anlegen einer Variablen geben Sie ein Kommando folgender Art ein (keine Spaces um das Gleichheitszeichen):

```
TERM=hp2393
NEU=Unsinn
```

Danach hat die bereits vorher vorhandene Variable TERM den Wert `hp2393`, und eine neue Variable NEU mit dem Wert `Unsinn` ist angelegt worden. Die Namen der Variablen werden üblicherweise groß geschrieben. Die ganze Gleichung ist ein String, dessen rechter Teil auch leer sein darf. In diesem Fall wird die Variable gelöscht. Soll der String Leerzeichen enthalten, muß er in Gänsefüßchen gesetzt werden:

```
PS1="A "
```

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. Soll eine Variable für die ganze Sitzung gelten, muß sie in der Sitzungsshell – also nicht in einer Subshell – eingerichtet und exportiert werden (zwei Schreibweisen):

```
NEU=Unsinn; export NEU
export NEU=Unsinn
```

Meist setzt man individuelle Variable in einem Shellscript namens `.profile` im Home-Verzeichnis entsprechend `autoexec.bat` unter MS-DOS. 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;
}
```

Programm 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 Filenamen 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 File `input` und schreibt es in das File `output`. Das Einlesen von `stdin` oder dem File `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 des Files, während der einfache Pfeil das File von Beginn an beschreibt. Existiert das File noch gar nicht, wird es in beiden Fällen erzeugt.

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

```
command 2>&1
```

den File-Deskriptor 2 (in der Regel `stderr`) des Kommandos `command` mit dem File-Deskriptor 1 (in der Regel `stdout`). Die Fehlermeldungen von `command` landen im selben File 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` (File-Deskriptor 1) in das File `output`. Anschließend wird `stderr` (File-Deskriptor 2) in das File umgelenkt, das mit dem File-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 das File `output`. Im File `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 das File geleert wird.

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

2.5.2 Shellscripts

Wenn man eine Folge von Kommandos häufiger braucht, schreibt man sie in ein File und übergibt dem Kommandointerpreter den Namen dieses Files. Unter MS-DOS heißt ein solches File Stapeldatei oder Batchfile, unter UNIX **Shellscript** und bei manchen Verfassern Kommandoprozedur, Makro oder Makrobefehl. Es ist nicht selbstverständlich, aber zweckmäßig, für die Shellscripts dieselbe Kommandosprache zu verwenden wie im Dialog. Der Teil der Shell, der Shellscripts abarbeitet, wird auch als Abwickler bezeichnet. Es gibt weitere Scriptsprachen – vor allem Perl (nicht Pearl, das ist eine andere Geschichte) – anstelle der Shellsprache. Shellscripts dürfen geschachtelt werden (ohne `call` wie in MS-DOS). Die externen UNIX-Kommandos sind teils unlesbare kompilierte Programme, teils lesbare Shellscripts.

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

```
sh shellscript
```

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

```
shellscript
```

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 Shellscripts 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. Die Syntax gleicht einer Mischung von BASIC und C. Man muß das Referenz-Handbuch sorgfältig lesen, gerade wegen der Ähnlichkeiten. **Kommentar** wird mit einem Doppelkreuz eingeleitet und wirkt bis zum Zeilenende. Das folgende Beispiel zeigt, wie man eine längere Pipe in ein Shellscript verpackt:

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

Programm 2.6 : Shellscript Frequenzwörterliste

Dieses Shellscript – in einem File namens `frequenz` – nimmt die Namen von einem oder mehreren Textfiles als Argument entgegen, liest die Files mittels `cat`, ersetzt alle Großbuchstaben durch Kleinbuchstaben, ersetzt weiterhin alle Zeichen,

die keine Buchstaben sind, durch Linefeeds (d. h. schreibt jedes Wort in eine eigene Zeile), sortiert das Ganze, 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 zuvörderst. Der Aufruf des Scripts 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
  5 auf
  5 zu
  4 aus
```

Solche Frequenzwörterlisten verwendet man bei Stiluntersuchungen, zum Anlegen von Stichwortverzeichnissen und beim Lernen von Fremdsprachen.

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

```
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 Shellscripts 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.

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 Shellscript, 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
```

Programm 2.7 : Shellscript zur Anzeige von Positionsparametern

Nun ein umfangreicheres Beispiel. Das Shellscript `userlist` wertet die Files `/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 Buchstaben und
# einer Ziffer bestehen. Usernamen wie root, bin, who, gast
# werden also nicht in die Liste aufgenommen. 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 vom ‘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 vom ‘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
```

Programm 2.8 : Shellscript zur Erzeugung einer Benutzerliste

Das folgende Shellscript 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\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
```

Programm 2.9 : Shellscript 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:

```
# Shellscript primscript zur Berechnung von Primzahlen, 26.05.92

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 ]
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
                        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
```

Programm 2.10 : Shellscript zur Berechnung von Primzahlen

Eine geschachtelte Verzweigung wie in obigem Shellscript darf auch kürzer mit `if - then - elif - then - else - fi` geschrieben werden. Man gewinnt jedoch nicht viel damit.

Die `for`-**Schleife** hat in Shellscripts eine andere Bedeutung als in C. Im fol-

genden 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, 06.11.91

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

Programm 2.11 : Shellscript zum Zählen der Files 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
```

Programm 2.12 : Shellscript 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 Shellscript 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.

Shellscripts 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

# Funktion, rekursiv

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
```

Programm 2.13 : Shellscript 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 Shellscript ist die Anzahl der Scheiben auf 16 begrenzt, weil mit steigender Scheibenzahl die Zeiten lang werden (Anzahl der Züge minimal $2^n - 1$).

Das Hauptscript 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 Scripts für $n = 3$ sieht folgendermaßen aus:

```
Tuerme von Hanoi (Shellscript)
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 Prográmmchen haben wir etwas 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.

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

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

# Default system-wide profile file (/bin/ksh initialization).
# This should be kept to the bare 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 am       \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
```

Programm 2.14 : Shellscript /etc/profile

Das Shellscript `.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. Seine Phantasie darf der Benutzer in einem File `.autox` ausleben. Das File `.logdat` speichert den Zeitpunkt der Anmeldung, so daß man bei einer erneuten Anmeldung feststellen kann, wann die vorherige Anmeldung stattgefunden hat, eine Sicherheitsmaßnahme.

```
# .profile zum Linken in die HOME-Directories der Benutzer
# von /etc kopieren nach /mnt/.profile, von dort linken
# ausser gast und dergleichen. 16.02.1993

EDITOR=vi
FCEDIT=vi
TMOUT=1000
PATH=/bin:/usr/bin:/usr/local/bin:/oracle/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
```

Programm 2.15 : Shellscript /etc/.profile

In dem obigen Beispiel `/etc/.profile` wird ein weiteres Script 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 Shellscripts. Er bewirkt, daß das Shellscript nicht von einer Subshell ausgeführt wird, sondern von der Shell, die den Punktbefehl entgegen-

nimmt. Damit ist es möglich, in dem Shellscript 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 Shellscript wird als **Punktscript** bezeichnet, obwohl der Aufruf das Entscheidende ist, nicht das Script.

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 Shellscripts, die vor Augen führen sollten, was die Shell leistet. Der Umfang der Shellsprache ist damit noch lange nicht erschöpft. Die Möglichkeiten von Shellscripts voll auszunutzen erfordert eine längere Übung. Die Betonung liegt auf voll, einfache Shellscripts 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)`, und wechseln Sie auf eine leistungsfähigere Shell, wenn Sie an die Grenzen der Bourne-Shell stoßen. Die Bourne-Shell kennengelernt zu haben, ist auf keinen Fall verkehrt. Wir erinnern daran, daß die UNIX-Shells sowohl interaktive Kommando-Interpreter als auch Programmiersprachen für Shellscripts sind, zwei zunächst verschiedene Aufgaben.

2.5.3 Noch eine Scriptsprache: Perl

Perl ist eine jüngere Alternative zur Shell als Scriptsprache (nicht als Kommandointerpreter) und vereint Züge von `sh(1)`, `awk(1)`, `sed(1)` und der Programmiersprache C. Sie 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-Page (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 Shellscript zur Berechnung von Primzahlen.

```
#!/usr/local/bin/perl
# perl-Script zur Berechnung von Primzahlen, 28. Nov. 1996

$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");
```

Programm 2.16 : Perlscript zur Berechnung von Primzahlen

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

- Die erste Zeile *muß* wie angegeben den Perl-Interpreter verlangen.
- Die Namen von Variablen beginnen mit Dollar, Buchstabe.
- Die Namen von Arrays beginnen mit dem at-Zeichen (Klammeraffe).
- Die Kontrollstrukturen 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)`-Scripts, 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 Perlscript 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 Perlscript:

```
#!/usr/local/bin/perl
# perl-Script zum Zaehlen in Buecherliste, 28. Nov. 1996
```

```
# 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");
}
```

Programm 2.17 : Perlscript 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 Perlscripts macht, läßt sich auch mit Programmen – vorzugsweise in C – erreichen. Was ist besser? Ein Script ist schnell geschrieben oder geändert, braucht nicht kompiliert zu werden (weil es interpretiert wird), läuft aber langsamer als ein Programm. Ein Script eignet sich daher für kleine bis mittlere Aufgaben zur Textverarbeitung oder Systemverwaltung, wobei Perl mehr kann als die Shell. Für umfangreiche Rechnungen oder falls die Laufzeit entscheidet, ist ein kompiliertes Programm besser. Oft schreibt man auch zunächst ein Script, probiert es eine Zeitlang aus und ersetzt es dann durch ein Programm. Gelegentlich spielt die Portierbarkeit auf andere Betriebssysteme eine Rolle. Ein UNIX-Shellscript 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. Und schließlich hat man auch seine Gewohnheiten.

2.5.4 Memo Shells

- Die Shell – ein umfangreiches Programm – ist der Gesprächspartner (Kommandointerpreter) in einer Sitzung. Es gibt mehrere Shells zur Auswahl, die sich in Einzelheiten unterscheiden.

- Die Shell faßt jede Eingabe als Kommando (internes K., externes Kommando = Shellscript oder Programm) auf.
- Die Shell stellt für die Sitzung eine Umgebung bereit, die eine Reihe von Werten (Strings) enthält, die von Shellscripts und anderen Programmen benutzt werden.
- Die Shell ist zweitens ein Interpreter für Shellscripts, eine Art von Programmen, die nicht kompiliert werden. Shellscripts können alles außer Gleitkomma-Arithmetik.
- Perl ist eine Scriptsprache alternativ zur Shell als Sprache, nicht als Kommandointerpreter. Sie setzt den Perl-Interpreter voraus.

2.5.5 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)
pg /etc/profile     (Shellscript anschauen)
pg .profile
```

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

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

und schreiben Sie in Ihr File `.profile` folgende Zeilen:

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

`fi` (Die Spaces und Punkte sind wichtig. Die Zeilen rufen das File `.autox` auf, falls es exisitiert.

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

Schreiben Sie ein Shellscript 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`.

2.6 Benutzeroberflächen

2.6.1 Lokale Benutzeroberflächen

2.6.1.1 Kommandozeilen-Eingabe

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 geechot und nach dem Drücken der Return-Taste ausgeführt. Die Ausgabe des Systems erfolgt ebenfalls auf den Bildschirm, Zeile für Zeile nacheinander.

Diese Art der Eingabe heißt **Kommandozeilen-Eingabe**. 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.

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
```

Schreibt man selbst Programme, 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 in ein Menü packen. Ein Ausweg 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. 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.

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-Scripts aufrufen. Das Eingangsmenu für unsere Datenbank sieht so aus:

```
Oracle-Hauptmenu (21.03.97 A)
================

Bibliothek          1
Buchaltung          2
Personen            3
Projekte            4

Bitte Ziffer eingeben:
```

Nach Eingabe einer gültigen Ziffer gelangt man ins erste Untermenu usf. Hinter dem Menu steckt ein Shellscript mit einer `case`-Anweisung, das letzten Endes die entsprechenden Shell- und SQLscripts 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 Fenster, curses-Bibliothek

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 in Kap. **??** *Programmieren in C/C++*. Darüberhinaus 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 modern mit Farbe und Maus gestalten will, greift zum X Window System 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 **Visual User Environment** (VUE) von Hewlett-Packard braucht. Dieses setzt auf dem X Window System auf und teilt die Fenster in vier oder mehr Gruppen 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 Gruppen 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[16]. Letzteres wird seit altersher bei der ersten Inbetriebnahme elektronischer Geräte aller Art gemacht (smoke test), weniger während des ordnungsgemäßen Betriebes. 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*. 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 vor allem in Betracht:

- Behinderungen des Sehvermögens
- Behinderungen des Hörvermögens
- Behinderungen der körperlichen Beweglichkeit

[16] Nachricht in Markt & Technik vom 31. März 1994: IBM entwickelt künstliche Nase. Vielleicht fordert Sie ihr Computer demnächst auf, die Socken zu wechseln oder den Kaffee etwas stärker anzusetzen.

Die Benutzung eines Computers durch einen Behinderten erfordert eine besondere Anpassung der Maschine, insbesondere des Terminals. Andererseits kann ein Computer als Hilfsmittel bei der Bewältigung alltäglicher Probleme dienen, zum Beispiel beim Telefonieren. Der bekannteste behinderte 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. Eine `archie`-Suche nach dem Substring `handicap` brachte überwiegend Material zum Golfspiel. In der Universität Karlsruhe bemüht sich das *Studienzentrum für Sehgeschädigte*, diesen das Studium der Informatik zu erleichtern: `http://szswww.ira.uka.de/`.

2.6.2 X Window System (X11)

2.6.2.1 Zweck

Das **X Window System** (*nicht*: Windows) ist ein netzfähiges, hardware- und betriebssystem-unabhängiges, grafisches 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 als sechstes Release X11R6. Eine freie Portierung auf Intel-Prozessoren heißt XFree86. Netzfähig bedeutet, daß die Rechnungen (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**). 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. Das X Window System enthält nur minimale Sicherheitsvorkehrungen. Der Preis für die Flexibilität ist ein hoher Bedarf an Speicherkapazität und Prozessorzeit, proprietäre Lösungen waren bescheidener.

Das X Window System stellt die Funktionen bereit, um grafische Benutzeroberflächen zu gestalten, legt aber die Art der Oberfläche nur in Grundzügen fest. Die Einzelheiten der Oberfläche sind Sache besonderer Funktionsbibliotheken wie **Motif** bzw. Sache bestimmter Programme, der Window-Manager, die nicht immer Bestandteil des X Window Systems 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 Manager `vuewm(1)` bei, unter LINUX findet sich der Win95

Window Manager `fvwm95(1)`, dessen Fenster an Microsoft Windows 95 erinnern. Das X Window System ist also keine Benutzeroberfläche, sondern *hat* eine oder sogar mehrere.

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.

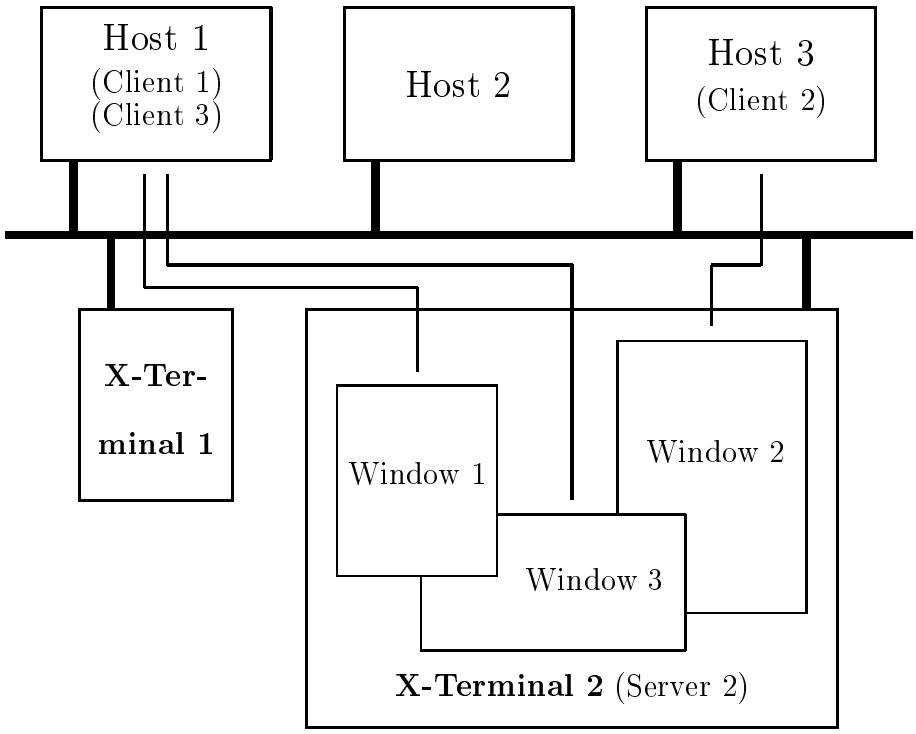

Abb. 2.7: X-Window-Computer und -Terminals, durch einen Ethernet-Bus verbunden

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 dem X Window System kann man auf dreierlei Weise zu tun bekommen:

- als Benutzer eines fertig eingerichteten X Window Systems (das gilt für wachsende Kreise von UNIX-Benutzern),
- als System-Manager, der ein X Window System auf mehreren Netzknoten einrichtet,
- als Programmierer, der Programme schreibt, die unmittelbar im Programmcode vom X Window System Gebrauch machen, also nicht wie gewöhnliche 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 entfernten Maschine namens `abcd` (wo seine Anwendung läuft, der X-Client) die Erlaubnis zum Zugriff. Das Kommando ohne Argument zeigt wie üblich die augenblicklichen Einstellungen an. Auf der entfernten Maschine `abcd` setzt er mit

```
export DISPLAY=efgh:0.0
```

die Umgebungsvariable `DISPALY` 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. Unter der Secure Shell `ssh(1)` werden beide Kommandos automatisch abgesetzt.

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.

Wer tiefer in das X Window System 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 das X Window System, 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 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, baut auf Motif auf. Es schieben sich immer mehr Schichten zwischen die CPU und den Benutzer.

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).

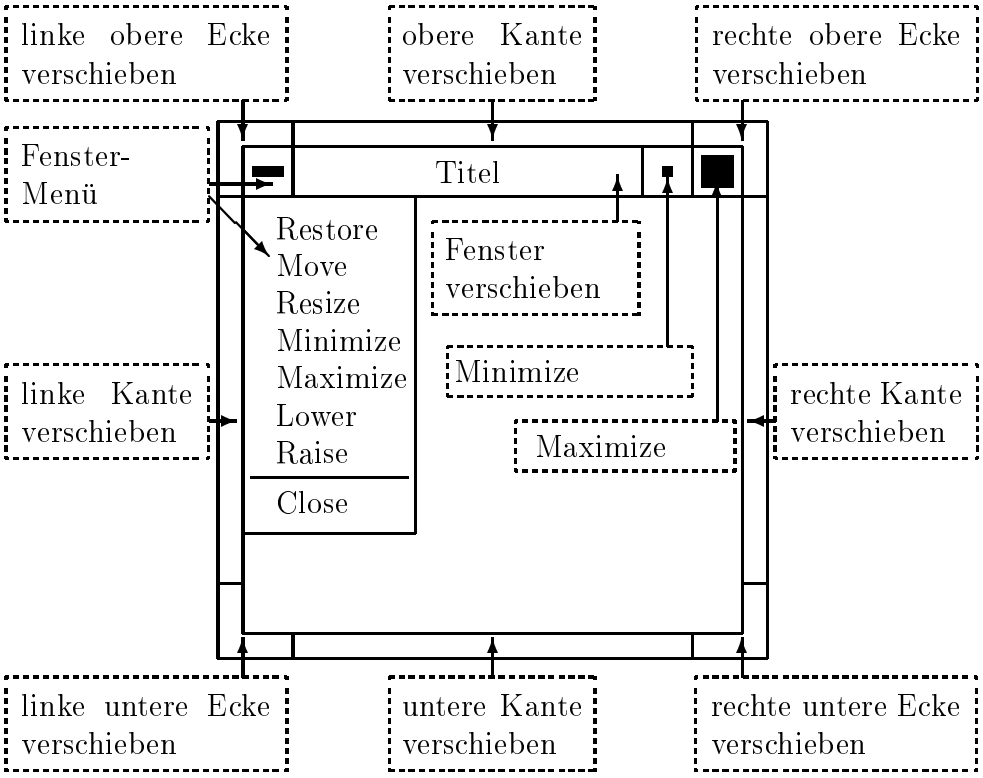

Abb. 2.8: OSF/Motif-Fenster

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 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 das **X Window System** und darin wiederum der **Motif Window Manager** `mvm(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) (Abb. 2.8).

Ehe ein Fenster bzw. der mit ihm verbundene Prozeß Eingaben annimmt, muß es durch Anklicken eines beliebigen Teils mit der Select-Maustaste **aktiviert** werden. Dabei ändert sich die Rahmenfarbe. Gibt man nun auf der Tastatur Zeichen ein, erscheinen sie im Fenster und gelangen auch zum Computer. 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.

Falls Sie nautisch vorbelastet sind und runde Fenster, sogenannte Bullaugen, bevorzugen, sollten Sie einmal nach **HAX/Rotif** Ausschau halten, eine vielversprechende Entwicklung aus fernerer Zukunft.

2.6.3 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.
- Das X Window System ist ein netzfähiges, grafisches Fenstersystem. Die Netzfähigkeit unterscheidet es von anderen grafischen Fenstersystemen.
- Der X-Server sorgt für die Ein- und Ausgabe auf einem grafischen 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-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 bietet.

- Für Programmierer stehen umfangreiche X- und Motif-Bibliotheken zur Verfügung.

2.6.4 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 Shellscript `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 das Beispiel aus Kap. **??** *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.

2.7 Writer's Workbench

Unter der *Werkbank des Schreibers* werden Werkzeuge zur Textverarbeitung zusammengefaßt. UNIX bietet eine ganze Reihe davon. Man darf jedoch nicht vergessen, daß UNIX kein Textsystem, sondern ein Betriebssystem ist.

2.7.1 Zeichensätze oder die Umlaut-Frage

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 außerhalb des englischen Sprachraumes überall auf.

Der Computer kennt nur Bits. Die Bedeutung erhalten die Bits durch die Programme. Ob eine Bitfolge eine Zahl, ein Wort oder einen Schnörkel darstellt, entscheidet die Software.

Zu Zeiten, als Bits noch knapp waren, haben die Yankees[17] eine 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 unterscheide ich $2^7 = 128$ Zeichen, numeriert von 0 bis 127. Diese Tabelle ist unter dem Namen *American Standard Code for Information Interchange* **ASCII** weit verbreitet. Genau heißt sie 7-bit-US-ASCII. Jeder Computer kennt sie.

Die ersten 32 Zeichen der ASCII-Tabelle dienen der Steuerung der Ausgabegeräte, es sind unsichtbare Zeichen. 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

[17] 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.

90. Die Kleinbuchstaben haben um 32 höhere Nummern als die zugehörigen Großbuchstaben. Der Rest sind Satzzeichen. Im Anhang ist die ASCII-Tabelle samt einigen weiteren Zeichensätzen wiedergegeben. Der Zeichensatz legt also fest, welche Zeichen verfügbar sind und welche Nummern ihnen zugeordnet werden. Es besagt nichts über das Aussehen der Zeichen.

Textausgabegeräte wie Bildschirme oder Drucker erhalten vom Computer 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.

Die US-ASCII-Tabelle 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* üblich. Für Frankreich oder Schweden lautet die Ersetzung anders. Diese Ersatztabelle liegt nicht im Computer, sondern im Ausgabegerät, das die Umsetzung der ASCII-Nummern in Zeichen vornimmt. Deshalb kann ein entsprechend ausgestatteter Bildschirm oder Drucker dasselbe Textfile einmal mit amerikanischen ASCII-Zeichen ausgeben, ein andermal mit deutschen ASCII-Zeichen.

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 ASCII-Zeichensatz identisch ist. Das hat den Vorzug, daß reine ASCII-Texte genau so verarbeitet werden wie ROMAN8-Texte. Leider hat sich dieser Zeichensatz nicht allgemein durchgesetzt.

Die Firma IBM hat schon frühzeitig bei größeren Anlagen den *Extended Binary Coded Decimal Interchange Code* **EBCDIC** mit acht Bits verwendet, der aber nirgends mit ASCII übereinstimmt. Hätte sich dieser Zeichensatz 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** (ISO 8859-1) Verbreitung gewonnen hat, vor allem in weltweiten Netzdiensten. Seine untere Hälfte ist wieder mit US-ASCII identisch. Polnische und tschechische Sonderzeichen sind in **Latin-2** enthalten.

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 kann. Die Programme müssen ebenfalls mitspielen. Der hergebrachte `vi(1)`-Editor, die `curses(3)`-Bibliothek für Bildschirmfunktionen und einige Mail-Programme verarbeiten nur 7-bit-Zeichen. Erst neuere Versionen von UNIX mit **Native Language Support** unterstützen 8-bit-Zeichensätze. Textverarbeitende Software, die 8-bit-Zeichensätze verträgt, wird als **8-bit-clean** bezeichnet. Bei Textübertragungen zwischen Computern ist Miß-

trauen angebracht. Die Konsequenz heißt in kritischen Fällen Beschränkung auf 7-bit-US-ASCII.

Was macht man, wenn es zu viele Standards gibt? Man erfindet einen neuen, der eine Obermenge der bisherigen ist. So wird zur Zeit ein internationaler Zeichensatz entwickelt, der mit 16 Bits alle abendländischen Zeichen berücksichtigt. Dieser **Intercode** ist aber noch nicht weit verbreitet. Und da sich immer Leute finden, die etwas besser machen, ist noch ein weltweiter **Unicode** mit ähnlicher Zielsetzung in der Mache.

Zur Umsetzung von Zeichen gibt es mehrere UNIX-Werkzeuge wie `tr(1)` und `sed(1)`. 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.
   Die 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);
        }
}
```

Programm 2.18 : C-Programm zur Zeichenumwandlung

Aus dem GNU-Projekt stammt ein Filter namens `recode(1)`, daß etwa hundert 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 (`&auml;` 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`. Das Ursprungsfile wird überschrieben, daher sicherheitshalber mit einer Kopie arbeiten. Nicht jede Umsetzung ist reversibel.

Man verwechsele nicht den Zeichensatz mit der Schriftart. Der Zeichensatz besagt nur, welche Zeichen verfügbar sind, nicht wie sie aussehen. Unter einer Schrift oder **Schriftart** (typeface) versteht man einen stilistisch einheitlichen Satz von Zeichen wie Times, Century Schoolbook, Garamond, Boldoni, Baskerville, Helvetica, Futura, Schwabacher, Courier, OCR (optical character recognition) und Schreibschriften. Diese Schriften liegen in verschiedenen **Schriftschnitten** (treatment) vor: mager, fett, 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, machen aber in Tabellen Mühe. Ein vollständiger Satz von Buchstaben, Ziffern, Satz- und Sonderzeichen einer Schrift, eines Schnittes, eines Grades und gegebenenfalls einer Schriftweite wird **Font** genannt. Die in diesem Text verwendeten Fonts heißen Roman (Times), `Courier`, Sans Serif, *Italic*, KAPITÄLCHEN, im Manuskript in 12 pt Größe.

Da die Papierformate länglich sind, spielt die **Orientierung** (orientation) eine Rolle. Das Hochformat wird englisch mit portrait, das Querformat mit landscape bezeichnet[18]. 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.

2.7.2 Reguläre Ausdrücke

Reguläre Ausdrücke (regular expression, expression régulière, RE) sind Zeichenmuster, die nach bestimmten Regeln gebildet und ausgewertet werden. Eine Zeichenkette (String) kann darauf hin untersucht werden, ob sie mit einem gegebenen regulären Ausdruck übereinstimmt oder nicht. Einige Textwerkzeuge wie die Editoren, `grep(1)`, `lex(1)` und `awk(1)` machen von regulären Ausdrücken Gebrauch, leider in nicht völlig übereinstimmender Weise. Die Jokerzeichen in Filenamen 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:

[18] Woraus man schließt, daß Engländer ein Flachland bewohnende Langschädler sind, während alpine Querköpfe die Bezeichnungen vermutlich andersherum gewählt hätten.

- Ein Zeichen mit Ausnahme der Sonderzeichen 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 Stern bedeutet ein beliebig häufiges Vorkommen dieses Zeichens, nullmaliges Vorkommen eingschlossen (erinnert an Jokerzeichen in Filenamen, aber dort kann der Stern auch ohne ein anderes Zeichen davor auftreten),
- eine Verkettung regulärer Ausdrücke trifft zu auf eine Verkettung von Strings, auf die die einzelnen regulären Ausdrücke zutreffen.

Die Regeln gehen noch 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 '\\[a-z][a-z]*{..*}' 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`, und das vierte Beispiel liefert die Zeilen mit LaTeX-Kommandos wie `\index{}`, `\begin{}`, `\end{}` zurück. Der vierte 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.

Wir wollen nun einen regulären Ausdruck zusammenstellen, der auf alle gültigen Internet-Email-Anschriften zutrifft. Dazu schauen wir uns einige Anschriften an:

```
wualex1@mvmhp64.ciw.uni-karlsruhe.de
wulf.alex@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 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:

```
^[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. 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 beschäftigen.

2.7.3 Editoren (ed, ex, vi, elvis, vim)

Ein **Editor** ist ein Programm zum Eingeben und Ändern von Texten, nach dem Kommando-Interpreter das am häufigsten benutzte Programm eines Systems. Alle Editoren stehen vor der Aufgabe, daß mittels derselben und einzigen Tastatur, gegebenenfalls noch mit Maus, sowohl der Text wie auch die Editierkommandos eingegeben werden müssen. Auf den meisten Computer-Tastaturen finden sich zwar einige Editiertasten (insert character, delete character usw.), diese reichen aber bei weitem nicht aus. Zudem sind sie von Tastatur zu Tastatur verschieden. Die beiden wichtigsten Editoren unter UNIX – der `vi(1)` und der `emacs(1)` – lösen die Aufgabe in unterschiedlicher Weise.

In Editoren kommt man leicht hinein, aber 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.
- 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`.

Falls das alles nicht wirkt, ist es Zeit, um Hilfe zu rufen.

Das einfachste Kommando zur Eingabe von Text ist `cat(1)`. Mittels

```
cat > textfile
```

schreibt man von der Tastatur in das File `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 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 MS-DOS `edlin` oder UNIX `ed(1)` werden heute nur noch für kurze Texte 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 benutzt. Der nachfolgend beschriebene

Editor `vi(1)` greift zwar oft auf `ex(1)`-Kommandos zurück, aber das braucht man nicht zu wissen. Da `ex` auf einigen anderen Systemen das Kommando zum Beenden der Sitzung ist und es immer wieder vorkommt, daß Benutzer unserer UNIX-Anlage dieses Kommando mit der letztgenannten Absicht eintippen, haben wir den Editor in `exed` umbenannt und unter `ex` ein hilfreiches Shellscript eingerichtet.

Auf dem `ex(1)` baut der verbreitete UNIX-Bildschirm-Editor `vi(1)` auf. Ein **Bildschirm-Editor** stellt einen ganzen Bildschirm oder mehr des Textes gleichzeitig zur Verfügung, so daß man mit dem Cursor im Text herumfahren kann. Dazu muß der `vi(1)` den Terminaltyp kennen, den er in der Umgebungs-Variablen TERM findet. Die zugehörige **Terminal-Beschreibung** sucht er im Verzeichnis `/usr/lib/terminfo`[19]. Falls diese fehlt oder – was noch unangenehmer ist – Fehler enthält, benimmt sich der `vi(1)` eigenartig. Näheres zur Terminalbeschreibung unter `terminfo(4)` sowie im Abschnitt 2.12.4.1 *Terminals*.

Da der `vi(1)` mit den unterschiedlichsten Tastaturen klar kommen muß, setzt er nur eine minimale Anzahl von Tasten voraus, im wesentlichen die Schreibmaschinentasten und Escape. Was sich sonst noch an Tasten oben und rechts befindet, ist nicht notwendig. Dies führt zu einer Doppelbelegung jeder Taste. Im **Schreibmodus** des `vi(1)` veranlaßt ein Tastendruck das Schreiben des jeweiligen Zeichens auf den Bildschirm und in den Speicher. Im **Kommandomodus** 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 den Text ab Cursorposition.

Die 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 und werden in der untersten Bildschirmzeile angezeigt. Weitere `vi(1)`-Kommandos im Anhang.

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 unseres Textfiles auf und geben folgendes Kommando ein:

[19] ehemals `/etc/termcap`

```
: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 Files verteilt. Soll eine Ersetzung in allen Files vorgenommen werden, schreibt man ein Shellscript `korr` und ruft es auf:

```
korr 's/kompil/compil/g' *.tex
```

Die korrigierten Texte findet man in den Files `*.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

print Ende korr
```

Programm 2.19 : Shellscript zur Textersetzung in mehreren Files

Beim Aufruf des `vi(1)` zusammen mit dem Namen eines existierenden Textfiles:

```
vi textfile
```

legt er eine Kopie des Files an und arbeitet nur mit der Kopie. Erst das abschließende `write`-Kommando – meist in der Form `:wq` für *write* und *quit* – schreibt die Kopie zurück auf den Massenspeicher. Hat man Unsinn gemacht, so quittiert man den Editor ohne zurückzuschreiben, und das Original ist nicht verdorben. Will man den `vi(1)` verlassen ohne zurückzuschreiben, warnt er. Greifen zwei Benutzer gleichzeitig schreibend auf dasselbe Textfile zu, so kann zunächst jeder seine Kopie editieren. Wer als letzter zurückschreibt, gewinnt.

In dem File `$HOME/.exrc` legt man individuelle **Tastatur-Anpassungen** nieder. Auch in einem Unterverzeichnis darf man noch einmal ein File `.exrc` unterbringen, dies gilt dann für `vi(1)`-Aufrufe aus dem Unterverzeichnis. Beispielsweise setzen wir für die Unterverzeichnisse, die unsere C-Quellen enthalten, die Tabulatorweite auf 4 statt 8 Stellen, um die Einrückungen nicht zu weit nach rechts wandern zu lassen. Das `.exrc`-File 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 sollte 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.

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 MS-DOS übertragen werden. Bei MINIX und LINUX gehört er zum Lieferumfang. Im Netz findet sich die `vi(1)`-Erweiterung `vim(1)`, auch für `vi(1)`-Liebhaber, die unter MS-DOS arbeiten.

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 MS-DOS sowie die Variante `microemacs`. Der grundsätzliche Unterschied zum `vi(1)` ist, daß der `emacs(1)` nur einen Modus kennt und die Editorkommandos 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-Browsers von:

- ftp.informatik.rwth-aachen.de/pub/gnu/
- ftp.informatik.tu-muenchen.de/pub/comp/os/unix/gnu/

oder anderen Servern. Das File heißt beisspielsweise `emacs-20.2.tar.gz`, ist 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 Files `README` und `INSTALL`, das File `PROBLEMS` heben wir uns für später auf. Im File `INSTALL` wird angeraten, sich aus dem File `./etc/MACHINES` die zutreffende Systembezeichnung herauszusuchen, in unserem Fall `hppa1.1-hp-hpux10`. Ferner soll man sich noch das File `leim-20.2.tar.gz` zur Verwendung internationaler Zeichensätze (Latin-1 usw.) besorgen und neben dem Emacs-File entpacken und aufdröseln; seine Files gehen in das Emacs-Verzeichnis. Dann ruft man ein Shellscript 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 Files löschen. Sobald alles funktioniert, sollte man auch das Verzeichnis `emacs-20.2` löschen, man hat ja noch das Archiv. Der fertige Editor – das File `/usr/local/bin/emacs` – sollte die Zugriffsrechte 755 haben. Mittels `man emacs` sollte die Referenz auf den Schirm kommen.

2.7.4.2 Benutzung

Der Aufruf `emacs mytext` startet den Editor zur Erzeugung oder Bearbeitung des Textfiles `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)`.

2.7.5 Joe's Own Editor (joe)

Der `joe(1)` 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 einem File `$HOME/.joerc` den eigenen Wünschen anpassen. Seine Verwendung unterliegt der GNU General Public License, das heißt sie ist praktisch 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, wie so manches andere.

2.7.6 Stream-Editor (sed)

Der **Stream-Editor** `sed(1)` bearbeitet ein Textfile nach Regeln, die man ihm als Option oder in einem getrennten File (sed-Script) mitgibt. Er ist im Gegensatz zu den bisher genannten Editoren nicht interaktiv, er führt keinen Dialog.

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 etwas umfangreich. Sie baut auf der Syntax des Zeileneditors `ed(1)` auf. Der Aufruf

```
sed 'Kommandos' filename
```

veranlaßt den `sed(1)`, das File `filename` einzulesen und gemäß den Kommandos bearbeitet nach `stdout` auszugeben. Der Aufruf

```
sed '1d' filename
```

löscht die erste Zeile im File `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:

```
sed 's/[A-Z]/[a-z]/g' filename
```

Im Kommando steht `s` für substitute. Dann folgt ein regulärer Ausdruck zur Kennzeichnung dessen, was ersetzt werden soll. An dritter Stelle ist der Ersatz (replacement) aufgeführt und schließlich ein Flag, das hier besagt, den Ersatz global (überall, nicht nur beim ersten Auftreten des regulären Ausdrucks in der Zeile) auszuführen.

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 Shellscripts sein und unbeaufsichtigt laufen kann.

2.7.7 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**. Er läßt sich auch als eine Programmiersprache für einen bestimmten, engen Zweck auffassen. Der `awk(1)` bearbeitet ein Textfile 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 Files 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
```

Das File `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 des Files 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 betrifft die zugehörige Aktion irgendwelche 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
```

Das File `wortliste` enthalte in alphabetischer Folge Wörter und gegebenenfalls weitere Bemerkungen zu den Wörtern, pro Wort eine Zeile. Der `awk(1)` liest das File 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 vorangegangen 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 ein eigenes File (awk-Script). Der Aufruf sieht dann so aus:

```
awk -f awkscript textfiles
```

awk-Scripts werden in einer Sprache geschrieben, die teils an Shellscripts, 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"
      }
```

Programm 2.20 : awk-Script 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), standardmäßig 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 dem ganzen Textfile verfahren.

Am Ende des Textfiles 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`-Script 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 Textfiles, 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 Shellscripts, siehe Abschnitt 2.5.3 *Noch eine Scriptsprache: Perl*.

2.7.8 Verschlüsseln (crypt)

2.7.8.1 Aufgaben der Verschlüsselung

Auf einem UNIX-System kann der Superuser (System-Manager) auf jedes File zugreifen, auf MS Windows NT 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). Diese Aufgabe behandeln wir im Abschnitt 3.11 *Electonic Mail*, weil sie dort eine Rolle spielt. Hier 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).

2.7.8.2 Symmetrische Verfahren

Im einfachsten Fall wird jedes Zeichen des Klartextes nach einer Regel durch ein anderes Zeichen desselben Alphabetes ersetzt. Die einfachste Regel 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 hatte viel Vertrauen in 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. Man kann die Verfahren raffinierter gestalten, indem man Zeichengruppen verschlüsselt, Blindzeichen unter den Geheimtext mischt, die Algorithmen wechselt usw.

Seit zwei Jahrzehnten 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 einfach 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 verstecken, 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. Ein weiteres Mitglied dieser Gruppe ist IDEA. 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). Die Crypt Breaker's Workbench enthält alles Nötige, um diese Verschlüsselung zu knacken (`http://axion.physics.ubc.ca/cbw.html`).

2.7.8.3 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. Das ist das Authentifizierungs-Problem, auf das wir bei der Email eingehen.

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 Computer, während die Zerlegung einer großen Zahl (um die zweihundert dezimale Stellen entsprechend etwa 500 Bits) in ihre Primfaktoren mit den heute bekannten Algorithmen und Computern 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 Computer kein Problem, für Taschenrechner schon 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. Auf `http://www.rsa.com/` findet sich Material zur Vertiefung des Themas. Eine zehnteilige FAQ-Sammlung zur Kryptografie liegt im Netz.

2.7.8.4 Angriffe

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. 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 Computerhilfe auch schnell von Erfolg gekrönt. Das Faktorisieren kleiner Zahlen ist ebenfalls kein Problem. Aber selbst bei großen Zahlen, die für einen einzelnen Computer – 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 Computern für seinen Zweck einsetzen kann. Das ist ein organisatorisches Problem, kein mathematisches, und bereits gelöst, siehe `http://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 immer das größte Risiko.

2.7.9 Formatierer (nroff, LaTeX)

2.7.9.1 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, auch **Aufmachung** oder **Layout** genannt. Beim Schreiben des Manuskriptes macht sich der Autor Gedanken über Struktur und Inhalt, aber kaum über Schrifttypen und Schriftgrößen, den Seitenumbruch, die Numerierung der Abbildungen. Das ist Aufgabe des Metteurs oder Layouters im Verlag, der seinerseits nur wenig am Text ändert. **Schreiben** und **Setzen** sind unterschiedliche Aufgaben, die unterschiedliche Kenntnisse erfordern.

Der Computer 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, stellt das Inhaltsverzeichnis zusammen usw.

Formatierer sind Werkzeuge, die strukturierten Text in bestimmter Weise, in einer bestimmten Form zu Papier oder auf den Bildschirm bringen. Eine typische Aufgabe ist der Zeilen- und Seitenumbruch. Man kann zwar diese Aufgabe auch von Hand im Editor erledigen, üblicherweise überläßt man sie aber dem Formatierer.

Der UNIX-Formatierer `nroff(1)` und LaTeX halten Struktur und Layout auseinander. Man schreibt mit einem beliebigen Editor den Text und formatiert anschließend. LaTeX verfolgt darüberhinaus den Gedanken, daß der Autor seine Objekte logisch beschreiben und von der typografischen Gestaltung, dem Layout, 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 erzeugen.

Sowohl `nroff(1)` wie LaTeX zielen auf die Wiedergabe der Dokumente mittels Drucker auf Papier ab. Die Hypertext Markup Language HTML, der 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.

Textverarbeitungsprogramme wie Word, Wordperfect oder Wordstar haben Formatierungsaufgaben integriert, so daß man sich am Bildschirm nicht nur den rohen Text, sondern während des Schreibens auch den formatierten ansehen kann. Diese Möglichkeit wird als **WYSIWYG** bezeichnet: *What you see is what you get*. Das erleichtert in vielen Fällen die Arbeit, birgt aber eine Versuchung in sich. Die reichen Mittel aus dem typografischen Kosmetikkoffer namens **Desktop Publishing** sind sparsam einzusetzen, unser Ziel heißt Lesbarkeit, nicht Kunst.

2.7.9.2 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.

2.7.9.3 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.

Ein File 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 im Textfile 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, Tests und Rechnungen möglich. S. R. Bourne führt in seinem im Anhang O *Literatur* genannten Buch die Makros auf, mit denen die amerikanische Ausgabe seines Buches formatiert wurde. Es gibt ganze Makrobibliotheken.

Da sich Formeln und Tabellen nur schlecht mit den Textbefehlen beschreiben lassen, gibt es 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.

2.7.9.4 LaTeX

TeX ist ein Formatierungsprogramm, das von Donald E. Knuth entwickelt wurde – dem Mann, der seit Jahrzehnten an dem siebenbändigen Werk *The Art of Computer Programming* schreibt und hoffentlich noch lange lebt – und wird von der American Mathematical Society herausgegeben. Seine Stärke sind umfangreiche mathematische Texte, seine Schwäche ist die Grafik. Inzwischen gibt es aber Zusatzprogramme (TeXCAD) zu LaTeX, die es erleichtern, den Text durch Zeichnungen zu ergänzen. Außerdem kann man Grafiken bestimmter Formate (Encapsulated Postscript) in den Text einbinden.

TeX ist sehr leistungsfähig, verlangt aber von seinem 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, was längeres Lernen voraussetzt.

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 die **LaTeX-Anweisungen**. Der Name des Textfiles muß die Kennung `.tex` haben. Dann schickt man das Textfile durch den **LaTeX-Compiler**. Dieser erzeugt ein Binärfile, dessen Namen die Kennung `.dvi` trägt. Das bedeutet **device independent**, das Binärfile ist also noch nicht auf ein bestimmtes Ausgabegerät hin ausgerichtet. Mittels eines geräteabhängigen Treiberprogrammes wird aus dem dvi-File das bit-File erzeugt – Kennung `.bit` – das mit einem UNIX-Kommando wie `cat` durch eine hundertprozentig transparente Schnittstelle zum Ausgabegerät geschickt wird. Es gibt auch ein Programm `dvips(1)`, das ein Postscript-File erzeugt, welches auf einem beliebigen Postscript-Drucker ausgegeben werden kann.

Format dieses Textes Der vorliegende Text wurde mit LaTeX2e auf einem LINUX-PC formatiert, mittels `dvips(1)` von Radical Eye auf Postscript umgesetzt und auf einem Laserdrucker von Hewlett-Packard ausgegeben.

Im wesentlichen verwenden wir die Standardvorgaben. Die Zeichnungen wurden mit TeXCAD entworfen. TeXCAD erzeugt LaTeX-Files, die man editieren kann, so man das für nötig befindet.

In dem Text kommen viele 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 `source`-Umgebung wird zusammen mit einigen weiteren Kleinigkeiten in einem File `alex.sty` definiert, das dem LaTeX-Kommando `usepackage` als Argument mitgegeben wird:

```
% alex.sty mit Erweiterungen fuer UNIX-Skriptum, 15. Mai 1994
% als Option in \documentstyle angeben:
% \documentstyle[12pt,twoside ... alex]{report}
%
% Aenderungen von Zeilen aus latex.tex, report.sty, rep12.sty
%
% Meldung fuer Bildschirm und main.log:
\typeout{Formatoption alex.sty fuer report.sty, 15. Mai 1994 W. Alex}
%
% Stichwoerter hervorheben und in Index aufnehmen:
% (hat sich als unzweckmaessig erwiesen)
\newcommand{\stw}[1]{{\em#1}\index{#1}}
%
% Hervorhebungen mittels \em:
\renewcommand{\em}{\bf}
%
% Fussnoten-Schriftgroesse vergroessern:
\renewcommand{\footnotesize}{\small}
%
% in Tabellen:
\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.):
\setcounter{totalnumber}{8}
\renewcommand{\textfraction}{0.1}
%
% Numerierung der Untergliederungen:
\setcounter{secnumdepth}{3}
\setcounter{tocdepth}{3}
%
% Satzspiegel vergroessern:
% Standardwerte 10, 138 und 187 mm; Springer 146, 236 mm
\topmargin-5mm
\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}
%
% 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, aber 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 fuer List of Sources (main.los):
\def\ext@source{los}
%
% neue Umgebung, Aufruf: \begin{source} .... \end{source},
% vor end{source} kommt caption:
\newenvironment{source}{\vskip 12pt \qunormal \def\@captype{source}}{\quactive \vs
%
% 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, aus 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}
%
% Man koennte noch ein Namensverzeichnis (name index) erfinden
%
```

Programm 2.21 : LaTeX-File alex.sty mit eigenen Makros, insbesondere der source-Umgebung

Das Prozentzeichen leitet Kommentar ein und wirkt bis zum Zeilenende. Der Text wurde auf mehrere Files aufgeteilt, die mittels `\include` in das Hauptfile `main.tex` eingebunden werden. Das Hauptfile sieht so aus:

```
% Hauptfile main.tex fuer das UNIX-Skriptum
% File alex.sty erforderlich, 15. Mai 1994, fuer source usw.
% Files bigtabular.sty und verbtext.sty erforderlich
% Umgestellt auf LaTeX2e 16.01.98 W. Alex
%
\NeedsTeXFormat{LaTeX2e}
\documentclass[12pt,twoside,a4paper]{report}
\usepackage{german,makeidx,bigtabular,verbatim,verbtext,alex}
\pagestyle{uxheadings}
\sloppy
%
% Universitaetslogo einziehen
\input{unilogo}
```

```
%
% Trennhilfe
\input{hyphen}
%
% Indexfile main.idx erzeugen
\makeindex
%
% nach Bedarf:
% \includeonly{copyright,vorwort,anhang}
%
\begin{document}
\unitlength1.0mm
\include{verweis}
\begin{titlepage}
\begin{center}
\vspace*{20mm}
\hspace*{13mm} \unilogo{32}\\
\vspace*{26mm}
\hspace*{14mm} {\Huge Einf"uhrung in UNIX\\}
\vspace*{11mm}
\hspace*{13mm} {\Large W. Alex, G. Bern"or und B. Alex\\}
\vspace*{11mm}
\hspace*{13mm} {\Large 1998\\}
\vspace*{80mm}
\hspace*{13mm} {\Large Universit"at Karlsruhe\\}
\end{center}
\end{titlepage}
\include{einleit/copyright}
\pagenumbering{roman}
\setcounter{page}{5}
\include{einleit/vorwort}
\overview
\tableofcontents
\listoffigures
%\listoftables
\listofsources
\include{einleit/gebrauch}
\cleardoublepage
\pagenumbering{arabic}
\include{einleit/umgang}
\include{unix/unix}
\include{internet/internet}
\begin{appendix}
\include{anhang/anhangU}
\end{appendix}
\cleardoublepage
\addcontentsline{toc}{chapter}{\indexname}
\printindex
\end{document}
```

Programm 2.22 : LaTeX-Hauptfile main.tex für Manuskript

LaTeX kennt mehrere Dokumentklassen, die sich unter anderem auf die Tiefe

der Strukturierung auswirken:

- `book` kennt Bände (volumes), Kapitel (chapters) usw.,
- `report` beginnt mit Kapiteln (chapters), Abschnitten (sections),
- `article` hat den Abschnitt (section) als oberste Einheit.

Im Gegensatz zum Inhaltsverzeichnis wird das **Sachverzeichnis** nicht automatisch erzeugt. LaTeX stellt nur Hilfen zur Verfügung. Der Befehl `makeindex` im Vorspann von `main.tex` führt zur Eintragung der im Text mit `\index` markierten Wörter samt ihren Seitenzahlen in ein File `main.idx`. In der Markierung der Wörter steckt viel Arbeit. Das `idx`-File 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 ein File namens `main.ind`, das ein bißchen editiert und durch den `\printindex`-Befehl am Ende des Manuskripts zum Dokument gebunden und ausgegeben wird.

2.7.9.5 Computer Aided Writing

Die Verwendung von Computern 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.

Neben diesen Äußerlichkeiten weisen Computertexte eine tiefer gehende Eigenart auf. In einem guten Text beziehen sich Sätze und Absätze auf vorangegangene Teile, sie bilden eine Kette, die nicht ohne weiteres unterbrochen werden darf. Ein roter Faden zieht sich durch das Ganze. Der Computer 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. Der rote Faden kommt unklar.

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 oder einem roten Faden. Wie ein Roman oder ein Gedicht in Hypertext aussehen könnten, ist noch nicht erprobt. Auf jeden Fall läßt sich Hypertext nicht vorlesen.

Heute kann ein Autor am Schreibtisch buchähnliche oder auch eigenständige Erzeugnisse schaffen, an deren Zustandekommen früher mehrere Berufe beteiligt waren und entsprechend Zeit benötigt haben. Bücher werden heute nach reproduktionsreifen (camera-ready) Vorlagen gedruckt, die aus dem Computer und dem Laser-Drucker stammen. Auch die Verteilung über elektronische Medien geht einfacher und schneller als auf dem hergebrachten Weg. Sogar die Rückkopplung vom Leser zum Autor ist im Netz eine Kleinigkeit.

2.7.10 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 Textfiles nach Zeichenmustern. Ein einfacher Fall: suche im File `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 MS-DOS heißt das entsprechende Werkzeug `find`, das nicht mit UNIX-`find(1)` verwechselt werden darf.

Für unsere Anlage haben wir mit dem `grep(1)` ein etwas leistungsfähigeres Shellscript 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"
```

Programm 2.23 : Shellscript zum Suchen in einem Telefonverzeichnis

`grep(1)` ist 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 eines Files miteinander verglichen. Bei entsprechendem Aufruf wird ein drittes File erzeugt, das dem Editor `ed(1)` als Kommandoscript (ed-Script) übergeben werden kann, so daß dieser aus der alten Version die neue erzeugt. Gebräuchlich zum Aktualisieren von Programmquellen. Schreiben Sie sich ein kleines Textfile `alt`, stellen Sie eine Kopie namens `neu` davon her, verändern Sie diese und rufen Sie dann `diff(1)` auf:

```
diff -e alt neu > edscript
```

Fügen Sie mit einem beliebigen Editor am Ende des `edscript` zwei Zeilen mit den `ed(1)`-Kommandos `w` und `q` (write und quit) hinzu. Dann rufen Sie den Editor `ed(1)` mit dem Kommandoscript auf:

```
ed - alt < edscript
```

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 Textfiles auf Muster
- `col` filtert Backspaces und Reverse Line Feeds heraus
- `comm` common, vergleicht zwei sortierte Files auf gemeinsame Zeilen
- `cut` schneidet Spalten aus Tabellen heraus
- `diff3` vergleicht drei Files
- `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 Files 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[20]
- `split` spaltet ein File in gleich große Teile
- `ssp` entfernt mehrfache leere Zeilen
- `tr` translate, ersetzt Zeichen
- `uniq` findet wiederholte Zeilen in einem sortierten File
- `vis` zeigt ein File an, das unsichtbare Zeichen enthält
- `wc` word counter, zählt Zeichen, Wörter, Zeilen

[20] Es gibt eine internationale Fassung `ispell` im GNU-Projekt.

Die Liste läßt sich durch eigene Werkzeuge beliebig erweitern. Das können Programme oder Shellscripts 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 Maß 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
```

Programm 2.24 : Shellscript 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 Computer; 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.

2.7.11 Textfiles aus anderen Welten (DOS, Mac)

In UNIX-Textfiles wird der Zeilenwechsel durch ein newline-Zeichen `\n` markiert, hinter dem das ASCII-Zeichen Nr. 10 (LF, Line feed) steckt, das auch durch die Tastenkombination `control-j` dargestellt wird. In MS-DOS-Textfiles wird ein Zeilenwechsel durch das Zeichenpaar Carriage return – Line feed (CR LF, ASCII Nr. 13 und 10, verb—control-m— und `control-j`) markiert, das Fileende 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 MS-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, das Textfile stammt jedoch aus der MS-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.

2.7.12 Druckerausgabe (lp, lpr)

Auf einer UNIX-Anlage arbeiten in der Regel mehrere Benutzer gleichzeitig, aber auch ein einzelner Benutzer kann gleichzeitig mehrere Textfiles 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.

Der Dämon sorgt auch dafür, daß die Drucker richtig eingestellt werden, beispielsweise auf Querformat oder deutschen Zeichensatz. Auf manchen Systemen findet sich ein File `/etc/printcap` mit einer Beschreibung der Drucker, ähnlich wie in `/usr/lib/terminfo` oder `/etc/termcap` die Terminals beschrieben werden.

Das Kommando zum Drucken lautet:

```
lp -dlp1 textfile
lpr -Plp1 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.

Bei der Einrichtung des Spoolers sind einige Punkte zu beachten. Wir wollen sie anhand eines Shellscripts `/etc/lpfix` erläutern, das den laufenden Spooler beendet und neu einrichtet. Dieses Shellscript wird auf unserer Anlage jede Nacht vom `cron` aufgerufen und sorgt dafür, daß morgens die Druckerwelt in Ordnung ist. Papier oder Toner füllt es nicht nach.

```
echo "Start /etc/lpfix"

# Skript zum Flottmachen des lp-Schedulers, 30.09.93
# Auftraege nicht retten, Warteschlangen putzen.

usl=/usr/spool/lp                       # lp-Directory

plist="lpjet lpplus plot"               # Liste der Drucker/Plotter

for p in $plist
do
/usr/lib/reject -rUnterbrechung $p      # Auftragsannahme schliessen
done

/usr/lib/lpshut                         # Jetzt herrscht Ruhe

rm -f $usl/pstatus $usl/qstatus         # Statusfiles putzen
touch $usl/pstatus $usl/qstatus
chown lp $usl/pstatus $usl/qstatus
chgrp bin $usl/pstatus $usl/qstatus

rm -f $usl/SCHEDLOCK                    # Lockfile loeschen

# interface-Files loeschen

rm -fr $usl/interface/* $usl/member/* $usl/request/*

# Konfigurieren der Schnittstellen

stty 9600  opost onlcr ixon ixoff < /dev/lpplus &
stty 19200 -opost ixon ixoff < /dev/lpjet &
stty 9600 ixon ignbrk icanon isig clocal < /dev/plot_mux &
stty erase "^-"  kill "^-"   < /dev/plot_mux &

sleep 4                         # Konfiguration dauert etwas

# Neuinstallation /dev/lpjet

/usr/lib/lpadmin -plpjet -v/dev/lpjet -mlpjet -h
/usr/lib/accept lpjet
```

```
/usr/bin/enable lpjet

# Neuinstallation /dev/lpplus

/usr/lib/lpadmin -plpplus -v/dev/lpplus -mlpplus -h
/usr/lib/accept lpplus
/usr/bin/enable lpplus

# Neuinstallation /dev/plot

/usr/lib/lpadmin -pplot -v/dev/plot -mhp7550a -h
/usr/lib/accept plot
/usr/bin/enable plot

/usr/lib/lpadmin -dlpjet                # default printer

/usr/lib/lpsched                        # Start lp-Scheduler

echo "Ende lpfix"
```

Programm 2.25 : Shellscript zum Flottmachen des Druckerspoolers

Das erste Spoolerkommando `/usr/lib/reject(1M)` – zu finden unter dem Kommando `accept(1M)` – sorgt dafür, daß der Spooler keine Aufträge mehr entgegennimmt. Ein Auftraggeber wird entsprechend unterrichtet. Das folgende Kommando `/usr/lib/lpshut(1M)` – unter `lpsched(1M)` beschrieben – beendet den Spoolprozeß.

Dann werden einige Files des Spoolsystems gelöscht und neu erzeugt, um zu verhindern, daß Müll herumliegt und beim Start Ärger macht. Das File `/usr/spool/lp/SCHEDLOCK` ist ein sogenanntes **Lockfile**, das beim Starten des Spoolers erzeugt wird, nichts enthält und allein durch sein Vorhandensein darauf hinweist, daß in dem System bereits ein Spooler läuft. Es dürfen nicht mehrere Spooler gleichzeitig arbeiten. Als nächstes werden etwaige Aufträge in den Warteschlangen für die jeweiligen Drucker gelöscht.

Mittels des Kommandos `stty(1)` werden die seriellen Drucker-Schnittstellen (Multiplexer-Ports) konfiguriert. Diese Zeilen sind eine Wiederholung von Zeilen aus dem Shellscript `/etc/rc`, das beim Systemstart (Booten) ausgeführt wird. Die Bedeutung der Argumente findet sich außer bei `stty(1)` auch unter `termio(7)`.

Schließlich werden die Drucker mit dem Kommando `/usr/lib/lpadmin(1M)` wieder installiert. Dieses Kommando erwartet hinter der Option `-p` den Namen des Druckers, unter dem er von den Benutzern angesprochen wird. Auf die Option `-v` folgt der Name des zugeordneten Druckers, wie er im Verzeichnis `/dev` eingetragen ist. Dieser braucht nicht mit dem erstgenannten übereinzustimmen. Es können einem physikalischen Drucker (Hardware) mehrere logische Drucker (Namen) zugeordnet werden. Der Spooler legt für jeden logischen Drucker eine eigene Warteschlange an. Falls mehrere Warteschlangen über ein physikalisches Gerät gleichzeitig herfallen, gibt es ein Durcheinander. Unter LINUX dient das Kommando `lpc(8)` der Verwaltung der Drucker.

Hinter der Option `-m` wird das **Modell-File** angegeben, ein Shellscript oder

kompiliertes Programm, das den zu druckenden Text bearbeitet, Druckersteuersequenzen ergänzt und das Ganze zum Drucker schickt. Hier bringt der System-Manager örtliche Besonderheiten unter. Die Modell-Files finden sich im Verzeichnis `/usr/spool/lp/model`. Sie sind zunächst nur eine unverbindliche Sammlung von Shellscripts oder Programmen; erst das `lpadmin(1M)`-Kommando ordnet einem logischen Drucker ein Modell-File zu, das dazu in das Verzeichnis `/usr/spool/lp/interface` kopiert und dann **Interface-File** genannt wird.

Mit `/usr/lib/accept(1M)` wird die Auftragsannahme wieder geöffnet (Gegenstück zu `reject(1M)`). Das Kommando `/usr/bin/enable(1)` aktiviert die Drucker. Mit `disable(1)` könnte man einen Drucker vorübergehend unterbrechen ohne die Auftragsannahme zu schließen, um beispielsweise Papier nachzulegen.

Zu guter Letzt startet `/usr/lib/lpsched(1M)` den Spooler wieder, und er beginnt mit der Abarbeitung der Warteschlangen. `lpstat(1)` mit der Option `-t` zeigt zur Kontrolle den Status des gesamten Spoolsystems an. Mit dem Kommando `lp(1)` übergeben nun die Benutzer ihre Aufträge an den Spooler.

Auf unserer Maschine haben wir ein lokales Druckmenü `p` geschrieben, das das Drucken von Textfiles für die Benutzer weiter vereinfacht. Es baut aus den Antworten des Benutzers das UNIX-Kommando `lp(1)` mit den entsprechenden Optionen und Argumenten auf. Die Optionen werden von dem angesprochenen Modell-File in Steuersequenzen für den Drucker umgesetzt. Das Menü und das Modell-File arbeiten Hand in Hand. Da die Druckausgabe unterschiedlich gestaltet werden kann, müssen Sie Ihren System-Manager fragen.

Laserdrucker gehobener Preisklassen bieten heute meist eine Möglichkeit zum unmittelbaren Anschluß an ein Netz (Ethernet). Sie erhalten dann eine eigene IP-Adresse im Internet und einen Namen wie ein Computer. 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 beispielweise die Ausgabe von kompilierten Programmen oder unsinnigen Steuerzeichen abfängt. Aus mancherlei Gründen gehören Druckerstörungen in einem heterogenen Netz leider zum täglichen Brot der System-Manager.

2.7.13 Memo Writer's Workbench

- Zeichen werden im Computer durch Nummern 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 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 und HTML.

- 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 noch mehr machen als mit Wortprozessoren, bei entprechendem Lernaufwand.
- Neben Editoren und Formatierern enthält UNIX eine Vielzahl kleinerer Werkzeuge zur Textbearbeitung (`grep(1)`, `sort(1)`, `diff(1)`, `awk(1)` usw.).
- Die Zeilenstruktur eines Textes wird in UNIX, in MS-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.

2.7.14 Übung Writer's Workbench

Anmelden wie gewohnt. Schreiben Sie mit dem Editor `vi(1)` einen knapp zweiseitigen Text mit einer Überschrift und einigen Absätzen. Das Textfile 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 Shellscript `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. Das File 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 Shellscript 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`.

2.8 Programmer's Workbench

Unter der *Werkbank des Programmierers* werden Werkzeuge zusammengefaßt, die beim Programmieren benötigt werden. Auf UNIX-Anlagen, die nicht zur Programmentwicklung eingesetzt werden, können sie fehlen.

2.8.1 Nochmals die Editoren

Editoren wurden bereits im Abschnitt 2.7 *Writer's Workbench* 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 gerät der Text schnell an den rechten Seitenrand. Es empfiehlt sich, in dem entsprechenden Verzeichnis ein File `.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.

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**[21] 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. 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:

```
cc -g source.c -lm
```

Die Option `-g` veranlaßt den Compiler, zusätzliche Informationen für den symbolischen Debugger zu erzeugen. Der Quelltext des C-Programmes steht im File `source.c`, das einen beliebigen Namen tragen kann, nur sollte der Name mit der Kennung `.c` enden. Die abschließende Option `-lm` fordert den Linker auf, die mathematische Bibliothek einzubinden. Weitere Optionen sind:

- `-v` (verbose) führt zu etwas mehr Bemerkungen beim Übersetzen,
- `-o` (output) benennt das ausführbare File 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 ein File `mon.out`, das mit dem Profiler `prof(1)` ausgewertet werden kann, um Zeitinformationen zum Programm zu erhalten,

[21] Linker werden auch Binder, Mapper oder Loader genannt. Manchmal wird auch zwischen Binder und Loader unterschieden, soll uns hier nicht beschäftigen.

- `-O` optimiert das ausführbare Programm oder auch nicht.

Speichermodelle wie unter MS-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 source.c
```

Ferner gibt es 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 das ursprüngliche File `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 Files 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. Auch für Manuskripte ist `make(1)` zu gebrauchen. Eigentlich läßt sich mit Makefiles fast alles erledigen, was man auch mit Shellscripts macht, die Stärke von `make(1)` liegt jedoch im Umgang mit Files unter Beachtung der Zeitstempel. Umgekehrt kann man auch mit Shellscripts fast alles bewältigen, was `make(1)` leistet, nur umständlicher.

Man lege für das Projekt ein eigenes Unterverzeichnis an, denn `make(1)` sucht zunächst im Arbeits-Verzeichnis. Das `makefile` beschreibt die Abhängigkeiten 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
```

Programm 2.26 : Einfaches make-File

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 dem include-File `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 demselben include-File 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 mit dem Ziel beginnt und rückwärts bis zu den Quellen geht.

`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 das include-File `incl.h` ausdrücklich genannt? Der Compiler weiß doch auf Grund einer entsprechenden Zeile im Quelltext, daß dieses File einzubinden ist? Richtig, aber `make(1)` muß das auch wissen, denn das include-File 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 der jüngsten Änderungen. Unveränderliche include-Files 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
```

Programm 2.27 : 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 File bezweckt. Ein Dummy-Ziel ist immer out-of-date, die zugehörigen Kommandos werden immer ausgeführt. Ein weiteres Beispiel für `make(1)` findet sich in Abschnitt `??` *Arrays von Funktionen*.

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
make
make install
make clean
```

wobei `make install` Schreibrechte in den betroffenen Verzeichnissen erfordert, also meist Superuserrechte. Die häufigsten Überraschungen beim Einrichten von GNU-Software sind:

- Fehlende include-Files oder Funktionsbibliotheken, irgendwoher beschaffen,
- die Files sind zwar vorhanden, liegen aber im falschen Verzeichnis (in diesem Fall Links anlegen),
- es ist zwar alles an Ort und Stelle, 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.

Ein allgemeines Rezept läßt sich nicht angeben. Gelegentlich hatten wir 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, und die GNU-Software ist den Versuch der Einrichtung allemal wert. Zudem kann man einiges über das Programmieren portabler Software und die Struktur von Programmen lernen.

2.8.4 Debugger (xdb)

Programme sind Menschenwerk und daher fehlerhaft[22]. 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.

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 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 ein unbrauchbares.

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

[22]Es irrt der Mensch, so lang er strebt. Goethe, Faust. Oder *errare humanum est*, wie wir Lateiner sagen. Noch etwas älter: $\alpha\mu\alpha\rho\tau\omega\lambda\alpha\iota\ \varepsilon\nu\ \alpha\nu\vartheta\rho\omega\pi\omega\iota\sigma\iota\nu\ \varepsilon\pi\sigma\nu\tau\alpha\iota\ \vartheta\nu\eta\tau\sigma\iota\varsigma$. 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 die älteste Dokumentation obiger Weisheit gilt.

Quellcode – und ist somit für die meisten Aufgaben wenig geeignet. Ein **symbolischer Debugger** wie der `sdb(1)` oder der `xdb(1)` bezieht sich auf die jeweilige Stelle im Quelltext[23]. Debugger sind mächtige und hilfreiche Werkzeuge. Manche Programmierer gehen so weit, daß sie das Schreiben eines Programms als Debuggen eines leeren Files 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 das `README`-File, das 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!

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 Abschnitt **??** *Ein Herz für Pointer* 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 ein File `gmon.out`, das mit `gprof(1)` betrachtet wird. Besser noch lenkt man die Ausgabe von `gprof(1)` in ein File um, das sich lesen und editieren läßt:

[23] Real programmers don't use source language debuggers.

```
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_constant
  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 Files (Texte, Grafiken) in gleicher Weise zusammenfassen. Dann spricht man allgemeiner von **Archiven**. Außer den Files 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 in Abschnitt **??** *Funktions-Bibliotheken* 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 indexsequentieller Files, 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 Fileformat 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;
}
```

Programm 2.28 : 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;
}
```

Programm 2.29 : 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;
}
```

Programm 2.30 : 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 das `include`-File `stdio.h` und gegebenenfalls durch Bibliotheksfunktionen kommen 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.

2.8.8 Programmverwaltung mit RCS und SCCS

Größere Projekte werden von zahlreichen, unter Umständen wechselnden Programmierern gemeinsam bearbeitet. Es hat auch schon Projekte gegeben, deren Programmierer über alle Kontinente und verschiedene Firmen verstreut waren. In der Regel werden die so entstandenen Programmpakete über Jahre hinweg weiterentwickelt und vielleicht auf mehrere Systeme portiert. Das von Walter F. Tichy entwickelte **Revision Control System RCS** ist ein Werkzeug, um bei der Entwicklung von Programmen 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 Quelltexten.
- Es ermöglicht, ältere Versionen wiederherzustellen, ohne daß diese vollständig gespeichert zu werden brauchen (Differenzen).
- Es verhindert gleichzeitige schreibende Zugriffe mehrerer Benutzer auf einen Quelltext.

Sowie es um mehr als Wegwerfprogramme geht, sollte man `make(1)` und RCS einsetzen. Arbeiten mehrere Programmierer an einem Projekt, kommt man um RCS oder Ähnliches nicht herum. Beide Werkzeuge sind auch für Manuskripte oder WWW-Files zu verwenden. RCS ist in den meisten LINUX-Distributionen enthalten. Man beginnt folgendermaßen:

- Unterverzeichnis anlegen, hineingehen.
- Mit einem Editor die erste Fassung des Quelltextes schreiben. Irgendwo im Quelltext - z. B. im Kommentar - sollte `$Header$` vorkommen, siehe unten. Dann übergibt man mit dem Kommando `ci filename` (check in) das File dem RCS. Dieses ergänzt das File durch Versionsinformationen und macht ein nur lesbares RCS-File (444) mit der Kennung `,v` daraus. Das ursprüngliche File löschen.
- Mit dem Kommando `co filename` (ohne `,v`) (check out) bekommt man eine Kopie seines Files zurück, und zwar nur lesbar. 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 das RCS-File 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, z. B. Grund und Umfang der Änderung.

Das ist für den Anfang alles. Die RCS-Kommandos lassen sich in Makefiles verwenden. Die vom RCS vergebenen Zugriffsrechte können von UNIX-Kommandos überrannt werden, aber das ist nicht Sinn der Sache. Der Einsatz von RCS setzt voraus, daß sich die Beteiligten an die Disziplin halten. Hier ein Makefile mit RCS-Kommandos für das nachstehende Sortierprogramm:

```
# makefile zu mysort.c, im RCS-System
# $Header: makefile,v 1.5 95/07/04 14:56:09 wualex1 Exp $

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
```

Programm 2.31 : Makefile zum Sortierprogramm mysort.c

Da dieses Beispiel sich voraussichtlich zu einer kleinen Familie von Quelltexten ausweiten wird, legen wir ein privates include-File mit unseren eigenen, für alle Teile gültigen Werten an:

```
/* myheader.h zum Sortierprogramm, RCS-Beispiel
   W. Alex, Universitaet Karlsruhe, 04. Juli 1995
*/

/* $Header: myheader.h,v 1.5 95/07/04 14:58:41 wualex1 Exp $ */

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 */
```

Programm 2.32 : Include-File zum Sortierprogramm mysort.c

Nun das Hauptprogramm, das die Verantwortung trägt, aber sonst nicht viel tut. Hier ist der Platzhalter `$Header$` 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
   W. Alex, Universitaet Karlsruhe, 04. Juli 1995
*/

static char rcsid[] =
      "$Header: mysort.c,v 1.9 95/07/04 14:18:37 wualex1 Exp $";

#include &#60;stdio.h&#62;
#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;
}
```

Programm 2.33 : 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: bubble.c,v 1.23 95/07/04 18:11:04 wualex1 Exp $ */

#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;
}
```

Programm 2.34 : C-Funktion Bubblesort

Bubblesort eignet sich für kleine Sortieraufgaben bis zu etwa hundert Elementen. Kopieren Sie sich die Bausteine in ein eigenes Verzeichnis und entwickeln Sie das Programm unter Verwendung des RCS weiter. Weiteres siehe `rcsintro(5)`.

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 einem einzigen SCCS-File 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.

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, `make(1)`, Analysewerkzeuge, Debugger und Versionskontrollsysteme unter einer einheitlichen Oberfläche – hier X Window System und Motif - zusammengefaßt werden. Damit zu arbeiten ist die moderne Form des Programmierens und kann effektiv sein.

2.8.9 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.

- 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 Files) 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 herum, vor allem dann, wenn mehrere Personen beteiligt sind. Das Lernen kostet Zeit, die aber beim Ringen mit dem Chaos mehr als gutgemacht wird.
- CASE-Tools vereinigen die einzelnen Werkzeuge unter einer gemeinsamen Benutzeroberfläche. Der Programmierer braucht gar nicht mehr zu wissen, was ein Compiler ist.

2.8.10 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: Es wird eine Ganzzahl-Variable d deklariert,
   die nicht gebraucht wird. 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);
```

Programm 2.35 : 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[24] 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 das ausführbare File `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` oder im Anhang nachlesen). Wir ersetzen also `a` durch `&a`. Das Compilieren erleichtern wir uns durch `make(1)`. Wir schreiben ein File namens `makefile` mit folgenden Zeilen:

```
fehler: fehler.c
    cc fehler.c -o fehler
```

[24] Real programmers can read core dumps.

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 Eingabe 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 einem ausführbaren File alles heraussucht, was nach String aussieht. Das Werkzeug `nm(1)` gibt eine Liste aller Symbole aus, die lang werden kann. `strip(1)` wirft aus einem ausführbaren File die nur für den Debugger, nicht aber für die Ausführung wichtigen Informationen heraus und verkürzt dadurch das File. Abmelden mit `exit`.

Das Programmieren vollzieht sich in mehreren Stufen parallel zur Zeitachse:

- Aufgabenstellung
- Aufgabenanalyse
- Umsetzung in eine Programmiersprache
- Testen
- Dokumentieren
- vorläufige Freigabe
- endgültige Freigabe

Des weiteren wird ein Programm 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 und Version darstellen.

2.9 Grafikers Atelier

2.9.1 Grundbegriffe

Es gibt keine UNIX-Grafik. Das heißt, 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, die nicht standardisiert sind und gelegentlich Geld kosten. Das Gleiche gilt auch 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 (z. B. serielle 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.
- Die Vielfalt grafischer Objekte (Form, Farbe, Beleuchtung, Perspektive) ist weit größer als die von Zeichen, von denen es in Europa nur wenige hundert und selbst in Fernost nur etwa zehntausend gibt.
- Die Vielfalt grafischer Operationen ist ebenfalls größer als die der Zeichenoperationen.

Heute ist die Bearbeitung grafischer und akustischer Daten mit durchschnittlicher Hardware möglich, aber über das Wie und Womit besteht noch keine Einigkeit. Auch über die Schnittstelle der Werkzeuge zum Menschen ist noch nicht alles gesagt, während bei den Zeichen die Schreibmaschine Vorarbeit geleistet hat. Die Lage ist nicht ganz so schlimm. Insbesondere in LINUX-Distributionen sind viele Grafikwerkzeuge enthalten, und es werden laufend mehr. Aber – wie gesagt – Einigkeit darf man nicht erwarten.

Hier sollen zunächst die Grundbegriffe der Verarbeitung von Grafiken erläutert werden. Die Aufgaben lassen sich in zwei Gruppen einteilen:

- die Erzeugung (**Synthese**) und anschließende Weiterverarbeitung von grafischen Objekten (CAD, Finite Elemente, Simulationen),
- die Verarbeitung (**Analyse**) von grafischen Objekten, die außerhalb des Computers entstanden sind (Schrifterkennung, Mustererkennung, Bildanalyse).

Wir befassen uns nur mit dem ersten Punkt.

Alle Grafikgeräte arbeiten entweder nach dem Raster- oder dem Vektorverfahren. Beim **Vektorverfahren** bestehen die Grafiken aus ununterbrochenen 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 (Bitmap). Beide Verfahren haben ihre Vor- und Nachteile.

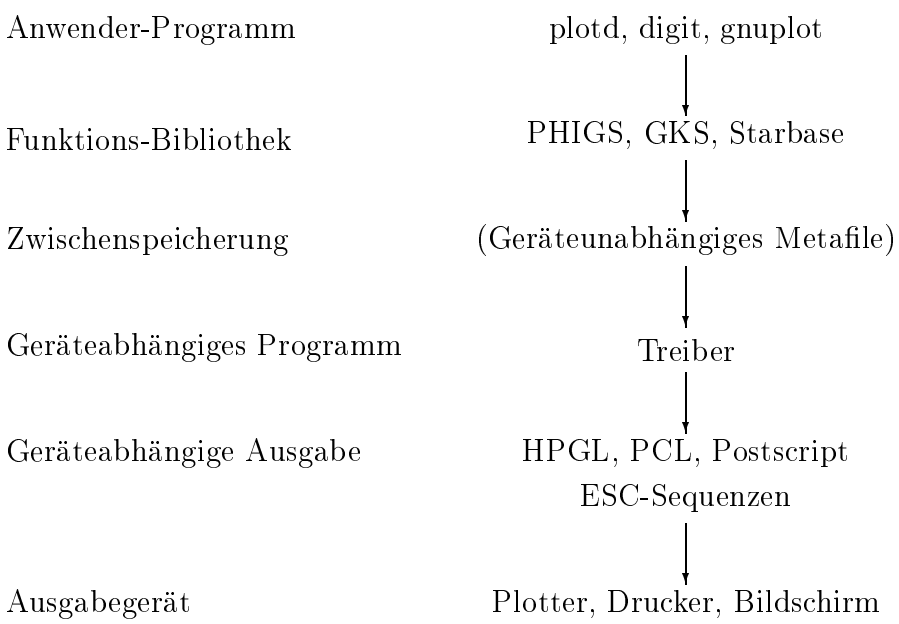

Abb. 2.9: Grafik von der Anwendung zur Ausgabe

Ausgabegeräte sind grafische Bildschirme, Plotter und grafikfähige Drucker. Die Eingabe ist das Ergebnis eines Programmes oder stammt von einem Scanner oder Digitalisiertablett. Die Ausgabegeräte werden in einer bestimmten **Steuersprache** angesprochen. Viele Plotter und manche Drucker verstehen die Hewlett-Packard Graphics Language (HPGL). Diese Sprachen enthalten elementare Befehle wie `select pen`, `pen up`, `pen down`, ziehe Linie von A nach B, `page feed`. Grafische Bildschirme und manche Drucker verlangen Escape-Sequenzen. Der Benutzer könnte ein File in dieser Sprache schreiben und zum Ausgabegerät schicken. Dieser Weg ist mühsam und dem Programmieren in Assembler vergleichbar. Deshalb gehören zu einer Grafikbibliothek auch Unterprogramme und Treiber, die dem Benutzer die Verwendung höherer Befehle ähnlich wie in FORTRAN oder C ermöglichen.

Farbe, Grafik-File-Formate

2.9.2 Diagramme (gnuplot)

`gnuplot(1)` ist ein Programm zum Zeichnen von Diagrammen, das GNU-üblich als Quellcode vorliegt, aber nicht aus dem GNU-Projekt stammt. Ausgangspunkt 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 Script gesteuert.

Die Ausgabe geht in ein File oder auf ein Gerät. Treiber für einige Terminals

und den HP Laserjet gehören dazu, ebenso die Möglichkeit, Postscript-, LaTeX- oder HPGL-Files zu erzeugen. Hier ein einfaches Beispiel. Wir schreiben ein Script `plotscript`:

```
set term latex            # Ausgabe im LaTeX-Format
set output "plot.tex"     # Ausgabe nach File plot.tex
plot sin(x)/x             # zu zeichnende Funktion
```

Programm 2.36 : gnuplot-Script zum Zeichnen der Funktion y = (sin x)/x, Ausgabe im LaTeX-Format auf File plot.tex

und rufen `gnuplot(1)` mit dem Script 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 Abb. 2.10.

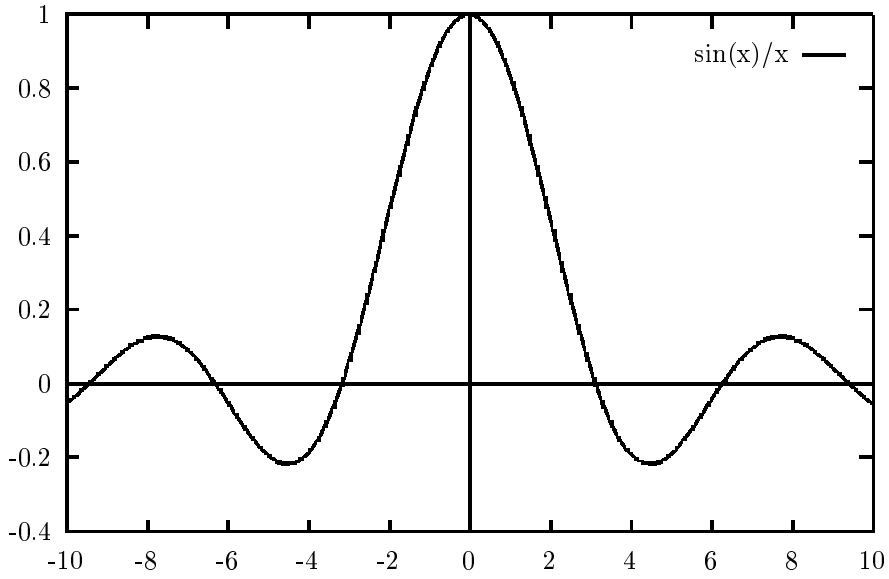

Abb. 2.10: Diagramm von (sin x)/x, erzeugt mit gnuplot

Nun wollen wir zu einer Menge von Wertepaaren eine Regressionsgerade berechnen, mittels `gnuplot(1)` in einem Diagramm darstellen und dieses in eine WWW-Seite einbinden.

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)`.

2.9.3 Zeichnungen (xfig, xpaint)

`xfig(1)` und `xpaint(1)` sind Werkzeuge, die unter dem X Window System laufen. Sie machen von dessen Funktionen Gebrauch und sind infogedessen netzfähig.

2.9.4 Funktions-Bibliotheken

2.9.4.1 GNU Graphics Library ()

Zur Verarbeitung von Zahlen braucht man Zahlenfunktionen wie Addition, Logarithmus, Regula falsi. Grafikfunktionen sind Verschieben (Translation), Drehen (Rotation), Spiegeln, Verzerren.

2.9.4.2 Starbase

Starbase ist eine Grafik-Bibliothek von Hewlett-Packard und nicht auf anderen Fabrikaten zu finden. Die Grundzüge sind jedoch ähnlich wie den übrigen Bibliotheken, so daß wir hier an einem Beispiel den Gebrauch von Grafik-Funktionen lernen können.

2.9.4.3 Graphical Kernel System (GKS)

Ein weit verbreitetes und vom American National Standards Institute (ANSI) genormtes Grafikpaket ist das **Graphical Kernel System** (GKS). 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 compilierte C- oder FORTRAN-Funktionen für eine Reihe von Prozessoren an. Die Sammlung enthält Funktionen:

- zur Ausgabe grafischer Grundelemente,
- 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.

2.9.5 Memo Grafik

- Für die Verarbeitung grafischer Daten gibt es keinen Standard (oder zuviele, was auf dasselbe hinausläuft).

- Unter UNIX gibt es keine Standard-Werkzeuge oder -Funktionen, wohl aber mehrere, teils freie Grafik-Pakete.
- `gnuplot(1)` ist ein interaktives Werkzeug zur Erzeugung von Diagrammen, ausgehend von Wertetabellen oder Funktionsgleichungen.
- `xfig(1)` und `xpaint(1)` sind Werkzeuge zum Herstellen und Bearbeiten von Zeichnungen, die auf dem X Window System aufbauen und in LINUX-Distributionen enthalten sind.
- Das Graphical Kernel System (GKS) ist eine von mehreren Bibliotheken mit Grafikfunktionen. Eine Alternative ist Starbase von Hewlett-Packard.
- Auch das X Window System (X11) enthält Grafikfunktionen, der Schwerpunkt liegt jedoch in der Gestaltung von Fenstern, die über das Netz gehen.

2.9.6 Übung Grafik

- Noch nichts.

2.10 Kommunikation

2.10.1 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 Kapitel 3 *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, so daß 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).

2.10.2 Mail (mail, mailx, elm)

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 Files (komprimierte Files, kompilierte Programme). Hat man binäre Files per Mail zu übertragen, muß man sie erst in Textfiles 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 (das File `/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 einem File `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 eines Files `$HOME/.mailrc`. Auf vielen Systemen ist auch das bildschirmorientierte und benutzerfreundlichere Mailkommando `elm(1)` vorhanden. Es setzt die richtige Terminalbeschreibung (TERM, `terminfo` oder `termcap`) voraus, fragt nach den notwendigen Informationen, ruft zum Schreiben den gewohnten Editor auf und läßt sich durch ein File `$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. Die Signatur sollte nicht länger sein als vier Zeilen, sonst macht man sich unbeliebt. `elm(1)` ist mit `mail(1)` verträglich, man kann sie durcheinander benutzen. Zu einem Zeitpunkt darf immer nur ein Mailprogramm aktiv sein, sonst gerät die Mailverwaltung durcheinander. Wird durch Lockfiles geregelt.

Es empfiehlt sich, einen logischen Benutzer namens `postmaster` mit einem Sternchen als Passwort in `/etc/passwd(4)` einzurichten und seine Mail an den System-Manager oder eine andere vertrauenswürdige Person weiterzuleiten, die täglich ihren Briefkasten leert. 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 Abschnitt 3.11 *Electronic Mail*.

Merke: Mail kann man nur an einen Benutzer schicken, nicht an eine Maschine.

2.10.3 Neuigkeiten (news)

Neuigkeiten oder **News** sind Mitteilungen, die jedermann schreiben und lesen darf. Die Files 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, siehe Abschnitt 3.12 *Neuigkeiten*.

Das File `.news_time` im Home-Verzeichnis hält die Zeit der letzten News-Anzeige fest, so daß man im Regelfall nur neue Mitteilungen zu lesen bekommt.

Das Kommando `news(1)` im File `/etc/profile` sorgt dafür, daß bei jeder Anmeldung die Neuigkeiten angezeigt werden. Sie können es aber auch gesondert eingeben. Mittels `news -a` werden alle, auch alte Nachrichten angezeigt.

2.10.4 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.

2.10.5 Ehrwürdig: UUCP

UUCP heißt *unix-to-unix-copy* und ist ein Programmpaket zur Übertragung von Files zwischen UNIX-Anlagen über serielle Kabel oder Modemstrecken, eine Alternative aus der Frühzeit der Netze 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 Modemstrecken über 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 die 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 Files in Textfiles 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.

2.10.6 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 Benutzern 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.

2.10.7 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, als letze Zeichen 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 Systemaufrufe

2.11.1 Was sind Systemaufrufe?

Dem Programmierer stehen zwei Hilfsmittel zur Verfügung, um seine Wünsche auszudrücken:

- die Schlüsselwörter (Wortsymbole) der Programmiersprache,
- die Systemaufrufe des Betriebssystems.

Die **Schlüsselwörter** (keyword, mot-clé) der Programmiersprache (zum Beispiel C) sind auch unter verschiedenen Betriebssystemen (MS-DOS, OS/2 oder UNIX) dieselben. Sie gehören zur Programmiersprache bzw. 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.

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.

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 File-System,
- Sicherheitsvorkehrungen.

Nach außen – Sie erinnern sich an das Bild mit dem Häuschen – 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, kann man das UNIX-Shell-Kommando nicht verwenden, sondern muß auf den Systemaufruf `time(2)` zurückgreifen. Es ist aber *nicht* so, daß sich grundsätzlich Shell-Kommandos und Systemaufrufe entsprechen, es sind nur einige Shell-Kommandos in C-Programme verpackte Systemaufrufe.

In 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 in Abschnitt **??** *Parameterübergabe*.

Falls Sie mit der Programmiersprache C nicht vertraut sind, sollten Sie jetzt zuerst das Kap. **??** *Programmieren in C/C++* überfliegen.

2.11.2 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 Quarz- oder 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)
```

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 verlangt ein Argument vom Typ Pointer auf long integer. Wir müssen also den Returnwert von `time(2)` in einen Pointer umwandeln (referenzieren). Der Returnwert ist vom Typ Pointer auf eine Struktur namens `tm`. Diese Struktur ist im include-File `time.h` definiert. Die include-Files 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;     /* nonzero if daylight saving t. */
}
```

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 */
/* Filename zeit.c, 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);
}
```

Programm 2.37 : 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[25]. 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
```

Programm 2.38 : FORTRAN-Programm zur Anzeige der Systemzeit

Die **ALIAS-Anweisung** ist als Erweiterung zu FORTRAN 77 in vielen 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` =

[25] FORTRAN 90 kennt Strukturen.

`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.

2.11.3 Beispiel File-Informationen (access, stat, open)

In einem weiteren Beispiel wollen wir mithilfe von Systemaufrufen Informationen über ein File 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 des Files als Argument übernehmen, wie wir es von UNIX-Kommandos her kennen. Dazu ist ein bestimmter Formalismus vorgesehen:

```
int main(argc, argv, envp)
int argc;
char *argv[], *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. 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 des gefragten Files. 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 ein File und hat die Syntax

```
int access(path, mode)
char *path;
int mode;
```

Der Systemaufruf erwartet als erstes Argument einen String, nämlich den Namen des Files. 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 Fileinformationen 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 Filename, das zweite der Name eines Puffers zur Aufnahme einer Struktur, die die Informationen enthält. Diese Struktur vom Typ `stat` ist in dem include-File `/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-Filesystemen enthält jedes File am Anfang eine **Magic Number**, die über die Art des Files Auskunft gibt (`man magic`). Mittels des Systemaufrufs `open(2)` wird das fragliche File 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 das File 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 ist unbekannt (Magic Number)");
                lseek(fildes, 0L, 0);
        }
    }
    else {
        puts("Probleme mit dem Filepointer");
    }

}
close(fildes);
}
```

Programm 2.39 : C-Programm zum Abfragen von Informationen über ein File

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
```

```
File heisst:        a.out
File existiert
File darf ausgefuehrt werden.
File darf nicht beschrieben werden.
File darf gelesen werden.

Device:             13
Inode-Nr.:          43787
File Mode:          33216

File ist regulaer

Links:              1
Owner-ID:           101
Group-ID:           20
Device-ID:          102536
Filegroesse:        53248
Letzter Zugriff:    24. 1. 91
Letzte Modifik.:    24. 1. 91
Letzte Stat.Ae.:    24. 1. 91
File ist executable
```

Die Bedeutung von `File Mode` finden Sie bei `mknod(2)`. Es handelt sich um ausführliche Informationen über die Zugriffsrechte usw.

2.11.4 Memo Systemaufrufe

- Systemaufrufe sind die Verbindungen des Betriebssystems nach oben, zu den Anwendungsprogrammen.
- Systemaufrufe haben vorwiegend mit Prozessen, den Filesystemen und der Ein- und Ausgabe zu tun.
- UNIX-Systemaufrufe sind C-Funktionen, die sich in keiner Weise von anderen C-Funktionen unterscheiden.
- C-Standardfunktionen gehören zum C-Compiler.
- Ein FORTRAN-Programmierer auf einem UNIX-System ist auf die UNIX-Systemaufrufe angewiesen, nicht aber auf die C-Standardfunktionen.

2.11.5 Übung Systemaufrufe

Schreiben Sie in einer Programmiersprache Ihrer Wahl (wir empfehlen C) ein Programm, daß

- ein File 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 das File schreibt. `fprintf(3)` finden Sie unter `printf(3)`.

Schreiben Sie ein Programm ähnlich `who(1)`. Sie brauchen dazu `getut(3)` und `utmp(4)`.

2.12 Systemverwaltung

Ein Betriebssystem wie UNIX läßt sich von drei Standpunkten aus betrachten, von dem

- des Benutzers,
- des System-Managers,
- des System-Entwicklers.

Der **Benutzer** möchte eine möglichst komfortable und robuste Oberfläche für die Erledigung seiner eigenen Aufgaben (Anwenderprogramme, Textverarbeitung, Information Retrieval, Programmentwicklung) vorfinden. 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), 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.

Zum Teil braucht auch der gewöhnliche Benutzer eine ungefähre Vorstellung von den Aufgaben des System-Managers, zum Teil muß er – vor allem auf kleineren Anlagen – diese Tätigkeiten selbst durchführen. Ein System-Manager kommt um das gründliche Studium der Handbücher nicht herum.

Die Systempflege ist die Aufgabe des **System-Managers**. Er braucht dazu die Vorrechte des **Superusers**. Beide Begriffe werden oft synonym gebraucht. Der Begriff System-Manager ist jedoch von der Aufgabe her definiert und daher treffender. Bei großen Anlagen findet man noch den **Operator**. Er ist unmittelbar für den Betrieb zuständig, überwacht die Anlage, beseitigt Störungen, wechselt Datenträger, hat aber weniger Aufgaben in Planung, Konfiguration oder Programmierung.

2.12.1 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 Wahrscheinlichkeit *sehr* hoch, daß etwas schiefgeht und man erst nach mehreren Versuchen Erfolg hat. 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 File-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 Files in wenigen Verzeichnissen (`/usr/local/bin`, `/usr/local/etc`, `/usr/local/config` 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 Files übernommen wird. Gerade im `/etc`-Verzeichnis sind viele Konfigurations-Files 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 Files 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 File-Systeme gemountet hat. Falls man nach der Länge der Filenamen gefragt wird, sollte man sich für lange Namen (maximal 255 Zeichen) entscheiden.

Im weiteren Verlauf werden viele Files auf die Platte kopiert, zwischendurch auch einmal gebootet und erforderlichenfalls der Datenträger gewechselt. Die Files 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. 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.

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.

2.12.2 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 weiteren Prozesse ist.

Der `init`-Prozess liest das File `/etc/inittab(4)` und führt die dort aufgelisteten Tätigkeiten aus. Dazu gehören die im File `/etc/rc` genannten Shell-Kommandos und die Initialisierung der Terminals. Im File (Shellscript) `/etc/rc` 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 dem File `/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 dem File `/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 das File `/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 einem File `.profile` (der Punkt kennzeichnet das File 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`-Files 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 dem File `/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)` sofort 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.

2.12.3 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 Files besitzt,

- Benutzer mit begrenzten Rechten wie `gast`, `ftp` oder Benutzer, die statt der Shell gleich ein bestimmtes Anwendungsprogramm bekommen, das sie nicht verlassen können,
- Normale Benutzer,
- Benutzer mit Superuser-Rechten wie `root`.

Formal ist der Eintrag in `/etc/passwd(4)` entscheidend. Deshalb ist dieses File so wichtig und eine Schwachstelle der System-Sicherheit.

Zur Einrichtung eines neuen Benutzers trägt der System-Manager den Benutzernamen in die Files `/etc/passwd(4)`:

```
wualex1:*:101:20:W. Alex:/mnt1/homes/wualex:/usr/bin/ksh
```

und `/etc/group(4)` ein:

```
users::20:root,wualex1,ig03,gebern1,ig05
```

Dann richtet er ein Home-Verzeichnis ein, übereignet es dem Benutzer und linkt 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**. Als **Passwort** wird zunächst die Kombination Komma-Punkt-Punkt eingetragen, die den Neuen beim ersten Einloggen dazu veranlaßt, sich ein Passwort zu geben. Das sollte unverzüglich erfolgen, da dieser Account vorübergehend jedermann offen steht. Ein Stern im Passwortfeld führt dazu, daß man sich unter dem zugehörigen Namen nicht anmelden kann. Nur `root` kann dann das Passwort ändern. Das braucht man für Dämonen wie `lp`, die als Filebesitzer auftreten, sowie bei Maßnahmen gegen unbotmäßige oder verschollene Benutzer. Das dritte und vierte Feld speichern die Benutzer- und Gruppennummer. Fünftens folgt das Kommentar- oder GECOS-Feld (GECOS = General Electric Comprehensive Operating System, ein historisches Relikt) mit Kommentar, der von Kommandos wie `finger(1)` ausgewertet wird. Schließlich das Home-Verzeichnis und die Sitzungsshell. Letztere darf kein weicher Link sein, sonst gibts Probleme. 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 das File von dieser Gruppe an nicht mehr gelesen wird.

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 als Sicherheitslücke, wegen des fehlenden Passwortes. Falls man sie dennoch einrichtet, soll man darauf achten, daß der aufrufende Benutzer keine Möglichkeit hat, das Kommando zu verlassen oder abzubrechen.

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. Das File `/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 dem File 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 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. Windows NT dagegen beschäftigt eine Schar von subalternen Managern unter dem Administrator.

2.12.4 Geräteverwaltung

2.12.4.1 Terminals

Alle Peripheriegeräte (Platten, Terminals, Drucker) werden von UNIX als Files 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.

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, so daß der System-Manager die Terminalbeschreibung für das `terminfo`-Verzeichnis selbst in die Hände nehmen muß, zumindest beim ersten Mal mit Nachdenken verbunden.

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 (früher in `/etc/termcap`), und zwar in einer compilierten 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)` compiliert, anschließend werden die Zugriffsrechte in der `terminfo` auf 644 gesetzt, damit die Menschheit auch etwas davon hat.

2.12.5 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 Shellscript `/etc/rc` samt Unterscripts oder in `.profile`. Die Shellscripts 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 fast aller übrigen 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 Filesystem zu tun hat.
- `syncer(1M)` ruft periodisch – in der Regel alle 30 Sekunden – den Systemaufruf `sync(2)` auf und bringt das File-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 Files 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, so daß Filesysteme 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.

2.12.6 Störungen und Fehler

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:

- Ursache und Auswirkung eines Fehlers können meilenweit auseinander liegen, bildlich gesprochen.
- Die meisten Fehlermeldungen sagen nichts oder führen in die Irre. Nur daß etwas faul ist, darf man glauben.
- Viel lesen. Manchmal findet sich ein Hinweis unter einer Überschrift, die dem Anschein nach nichts mit dem Fehler zu tun hat.
- Nachdem man gelesen hat, darf man auch fragen. Vielleicht hat ein Leidensgenosse schon mit dem gleichen Fehler gerungen.
- Sich niemals auf eine einzige Fehlerursache versteifen.

2.12.7 Pflege des File-Systems

Das **File-System** kann durch Stromausfälle und ähnliche Unregelmäßigkeiten fehlerhaft (korrupt) werden und wird mit Sicherheit nach einiger Zeit überflüssige Daten enthalten. Nahezu volle File-Systeme geben leicht Anlaß zu Störungen. Deshalb ist eine Pflege notwendig.

Den Füllstand der File-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 im File `/etc/checklist(4)` aufgelisteten File-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 File-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 Files erfordert eine regelmäßige, wenigstens wöchentliche Beobachtung, wobei der `cron(1M)` hilft. Files mit Namen wie

`core(4)`, `a.out(4)` oder `*.bit` werden üblicherweise nicht für eine längere Zeit benötigt und sollten automatisch gelöscht werden. Files, auf die seit einem Monat nicht zugegriffen wurde, gehören nicht auf die kostbare Platte; mit

```
find -atime +32 -print
```

aufspüren und die Benutzer bitten, sich ein anderes Medium zu suchen. 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 Files und Verzeichnisse überwachen. Einige Protokollfiles 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 Files ermitteln, deren Größe über n Blöcken liegt, mit folgendem Script 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`
```

Programm 2.40 : Shellscript zur Ermittlung der Größe aller Home-Verzeichnisse

Mittels `du(1)` kann man auch in dem Script `/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)`.

2.12.8 Weitere Dienstleistungen

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), das Schreiben oder Herbeischaffen (GNU!) von Werkzeugen in `/usr/local/bin`, 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 und Trösten der Benutzer.

Eine Anlage 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)` oder ein eigenes Programm unter Verwendung des Systemaufrufs `pstat(2)` an.

2.12.9 Accounting System

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)`, im File `/etc/rc` in Gang gesetzt. Das Gegenstück `/usr/lib/acct/shutacct(1M)` ist Teil des Shellscripts `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 die Abrechnung 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 ein File 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)`; das `crontab`-File 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
```

Das File wird mit dem Kommando `crontab(1)` compiliert. Im einzelnen bewirken die Zeilen

- `runacct(1M)` ist ein Shellscript, das täglich ausgeführt wird und den Tagesverlauf in verschiedenen Files 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 einem File `/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.

2.12.10 Sicherheit

Zur Sicherheit gehören drei Bereiche:

- die **Betriebssicherheit** oder Verfügbarkeit des Systems,
- der Schutz der Daten vor Verlust oder Beschädigung (**Datensicherheit** oder Datenintegrität),
- der Schutz personenbezogener oder sonstwie vertraulicher Daten vor Mißbrauch (**Datenschutz** oder Datenvertraulichkeit).

In allen drei Bereichen sind Maßnahmen auf der Hard- und der Softwareseite erforderlich. Bei hohen Anforderungen erstrecken sich die Maßnahmen auch auf Gebäude, Personen und Organisationsstrukturen. Besonderen Wert lege man auf die Auswahl und Pflege der System-Manager.

2.12.10.1 Betriebssicherheit

Maßnahmen zur Betriebssicherheit stellen das ordnungsgemäße Funktionieren des Systems sicher. Dazu gehören:

- Schaffung von Ausweichmöglichkeiten bei Hardwarestörungen
- Sicherung der Stromversorgung
- Vermeidung von Staub in den EDV-Räumen, Rauchverbot
- Vermeidung von elektrostatischen Aufladungen
- Klimatisierung der EDV-Räume, zumindest Temperierung
- vorbeugende Wartung der Hardware (Filter!) und der Klimaanlage
- vorbeugendes Auswechseln hochbeanspruchter Teile (Festplatten)
- Vorbereitung von Programmen oder Skripts zur Beseitigung von Softwarestörungen
- regelmäßiges (z. B. wöchentliches) Rebooten der Anlage
- Protokollieren und Analysieren von Störungen

- Überwachung des Zugangs zu kritischen Systemteilen
- Überwachung des Netzverkehrs, Beseitigung von Engpässen
- Kenntnisse der Systemverwaltung bei mehreren, aber nicht zu vielen Mitarbeitern

Wir setzen teilweise PC-Hardware für zentrale Aufgaben ein und achten darauf, daß die Gehäuse geräumig sind und mehrere Lüfter haben. Einfache Lüfter tauschen wir gegen kugelgelagerte Lüfter aus dem Schwarzwald. Ferner entstauben wir Computer und Drucker mindestens einmal im Jahr (Osterputz). Festplatten, die im Dauerbetrieb laufen, wechseln wir nach drei Jahren aus. Dann sind sie auch technisch überholt; in weniger beanspruchten Computern tun sie noch einige Zeit Dienst. Ein Sorgenkind sind die winzigen Lüfter auf den Prozessoren. Eine Zeitlang haben wir sie gegen große statische Kühlkörper getauscht. Das setzt voraus, daß man sich mittels Temperaturmessungen über die Erwärmung Klarheit verschafft. Hohe Temperaturen führen sowohl bei mechanischen wie bei elektronischen Bauteilen zunächst nicht zum Ausfall, sondern setzen die Lebensdauer herab und verursachen gelegentliche Störungen, die unangenehmer sind als ein klarer Defekt.

Bei ausgereiften Einzelsystemen erreicht man heute eine Verfügbarkeit von etwa 99 %. Das sind immer noch drei bis vier Fehltage im Jahr. Nach MURPHY's Law sind das die Tage, an denen man den Computer am dringendsten braucht. Höhere Verfügbarkeit erfordern besondere Maßnahmen wie mehrfach vorhandene, parallel arbeitende Hard- und Software. Damit erreicht man zu entsprechenden Kosten Verfügbarkeiten bis zu etwa 99,999 % gleich fünf Fehlminuten pro Jahr[26].

2.12.10.2 Datensicherheit

Auch bei einer gut funktionierenden Anlage gehen Daten verloren. Häufigste Ursache sind Fehler der Benutzer wie versehentliches Löschen von Files.

Versehentliches Löschen Das Löschkommando `rm(1)` fragt nicht, sondern handelt. Kopierkommandos wie `cp(1)` löschen oder überschreiben stillschweigend das Zielfile. Eine Rücknahme des Kommandos mit `undelete` oder ähnlichen Werkzeugen wie vom PC bekannt gibt es nicht. Will man vorsichtig sein, ruft man es grundsätzlich interaktiv auf, Option `-i`. Traut der System-Manager seinen Benutzern nicht, benennt er das Original um und verpackt es mit besagter Option in ein Shellscript oder Alias namens `rm`. Man kann auch aus dem Löschen ein Verschieben in ein besonderes Verzeichnis machen, dessen Files nach einer bestimmten Frist vom `cron` endgültig gelöscht werden, aber das kostet Platz auf der Platte. Gehen dennoch Daten verloren, bleibt die Hoffnung aufs Backup.

Passwörter Die Passwörter sind der Schlüssel zum System. Da die Benutzernamen öffentlich zugänglich sind, sind jene der einzige Schutz vor Mißbrauch. Besonders reizvoll ist das Passwort des System-Managers, der bekanntlich den

[26] Welche Verfügbarkeit haben Organe wie Herz oder Hirn?

Namen `root` führt. Das `root`-Passwort sollte nie unverschlüsselt über ein Netz laufen, sondern nur auf der Konsole eingegeben werden. Die einfachsten und daher häufigsten Angriffe zielen auf Passwörter ab, und oft haben sie wegen Schlamperei im Umgang mit den Passwörtern Erfolg. Eine internationale Gruselgeschichte baut darauf auf, daß ein Hacker das Passwort der `root` erraten hat.

Ein Passwort darf nicht zu einfach aufgebaut und leicht zu erraten sein. UNIX verlangt sechs bis acht Zeichen, davon mindestens zwei Buchstaben und eine Ziffer oder ein Satzzeichen. Ferner kann der System-Manager einen Automatismus einrichten, der die Benutzer zwingt, sich in regelmäßigen Zeitabständen ein neues Passwort zu geben (password aging). Das hat aber auch Nachteile. So habe ich noch meine vor Jahren gültigen Passwörter im Kopf, selten jedoch meine neuesten.

Ein Passwort soll nicht aus allgemein bekannten Eigenschaften des Benutzers oder des Systems erraten werden können oder ein Wort aus einem Wörterbuch oder der bekannteren Literatur sein. Folgende Passwörter sind leicht zu erraten, inzwischen ohne Mühe per Programm, und daher verpönt:

- bürgerlicher Vor- oder Nachname (w_alex), Übernamen (oldstormy),
- Namen von nahen Verwandten (rainer),
- Namen von Haustieren (hansi),
- Namen von Freunden, Freundinnen, Kollegen, Sportkameraden (steffig),
- Namen bekannter Persönlichkeiten (thmann),
- Namen von bekannten Romanfiguren (hamlet, winnetou, slartibartfast),
- Namen von Betriebssystemen oder Computern (Unix, pclinux),
- Telefonnummern, Autokennzeichen, Postleitzahlen, Jahreszahlen (1972),
- Geburtstage, historische Daten (issos333),
- Wörter aus Wörterbüchern, insbesondere englischen oder deutschen,
- Benutzernamen oder Gruppennamen auf Computern (owner, student),
- Orts-, Pflanzen- oder Tierbezeichnungen (karlsruhe, drosophila),
- Substantive jeder Sprache,
- nebeneinanderliegende Zeichenfolgen der Tastatur (qwertyu),
- einfache, wenn auch sinnlose, Kombinationen aus Buchstaben, Ziffern oder Satzzeichen (aaAAbbBB),
- etwas aus obiger Aufzählung mit vorangestellter oder angehängter Ziffer (3anita),
- etwas aus obiger Aufzählung rückwärts oder abwechselnd groß und klein geschrieben.

Ein zufällig zusammengewürfeltes Passwort läßt sich nicht merken und führt dazu, daß man einen Zettel ans Terminal klebt oder unter die Schreibunterlage legt. Eine brauchbare Methode ist, einen Satz (eine Gedichtzeile oder einen Bibelvers) auswendig zu lernen und das Passwort aus den Anfangsbuchstaben der Wörter zu

bilden oder zwei Wörter mit einer Primzahl zu mischen. Auch der Roman *Finnegans Wake* von JAMES JOYCE gibt gute Passwörter her, weil ihn kaum jemand liest. Ein Passwort wie *gno596meosulphidosalamermauderman* – die Ziffern sind die Seitenzahl – treibt jeden Cracker in den Wahnsinn. Und schließlich beherzige man den Rat unserer Altvordern[27]

> Selber wisse mans,
> nicht sonst noch jemand,
> das Dorf weiß, was drei wissen.

Bedenken Sie, es sind nicht nur Ihre Daten, die gefährdet sind, sondern die ganze Maschine ist kompromittiert und möglicherweise die Bresche für weitere Angriffe, falls Sie ein zu einfaches Passwort verwenden. Es gibt Programme wie `crack` oder `satan`, die einfache Passwörter herausfinden und so dem System-Manager helfen, leichtsinnige Benutzer zu ermitteln.

Einige Versuche, an Passwörter zu gelangen, sind in die Literatur eingegangen. Der simpelste Trick ist, einem Benutzer beim Einloggen zuzuschauen. Auch in Papierkörben findet man Zettel mit Passwörtern. Einem intelligenten UNIX-Liebhaber angemessener sind Programme, die als **Trojanische Pferde**[28] bekannt sind. Vom Chaos Computer Club, Hamburg soll folgende Definition stammen: Ein Trojanisches Pferd ist ein Computerprogramm, welches in einen fremden Stall (Computer) gestellt wird und bei Fütterung mit dem richtigen Kennwort alle Tore öffnet.

Ein solches Programm startet man aus seiner Sitzung. Es schreibt die übliche Aufforderung zum Einloggen auf den Bildschirm und wartet auf ein Opfer. Dieses tippt nichtsahnend seinen Namen und sein Passwort ein, das Pferd schreibt beides in ein File und verabschiedet sich mit der Meldung `login incorrect`. Daraufhin meldet sich der ordnungsgemäße `getty`-Prozess, und das Opfer – in dem Glauben, sich beim erstenmal vertippt zu haben – wiederholt seine Anmeldung, diesmal mit Erfolg. Ein Trojanisches Pferd ist einfach zu schreiben:

---

[27] Leider können wir Ihnen hier nur die Übersetzung von FELIX GENZMER bieten, der Suche nach der originalen Fassung war noch kein Erfolg vergönnt, aber wir sind dank `http://www.lysator.liu.se/runeberg/eddais/on-02.html` nahe dran. Vermutlich liegen dem Ratschlag folgende Zeilen zu Grunde:

> Einn vita
> né annar skal,
> thjóth veit ef thrír eru.

[28] Die Definition des ursprünglichen Trojanischen Pferdes ist nachzulesen in HOMERS *Odyssee* im 8. Gesang (hier in der Übertragung von JOHANN HEINRICH VOSS):

> Fahre nun fort und singe des hölzernen Rosses Erfindung,
> Welches Epeios baute mit Hilfe der Pallas Athene
> Und zum Betrug in die Burg einführte der edle Odysseus,
> Mit bewaffneten Männern gefüllt, die Troja bezwangen.
> ...

```
/* Trojanisches Pferd */
/* Filename horse.c, Compileraufruf cc -o horse horse.c */

#define PROMPT0 "UNIX\n"
#define PROMPT1 "HP-login: "
#define PROMPT2 "Passwort: "

#define CLEAR   "\033H\033J"      /* Escapes fuer HP */
#define INVIS   "\033&dS\033*dR"
#define VISIB   "\033&d@\033*dQ"

#include <stdio.h>
#include <sys/ioctl.h>
#include <signal.h>

main()
{
char name[32], pwort[32], zucker[8];
unsigned long sleep();

signal(SIGINT, SIG_IGN);
signal(SIGQUIT, SIG_IGN);

printf(CLEAR);
printf(PROMPT0);

while (strlen(name) == 0) {
      printf(PROMPT1);
      gets(name);
}

printf(PROMPT2);
printf(INVIS);
gets(pwort);
printf(VISIB);

sleep((unsigned long) 2);

printf("\nIhr Name ist %s.\n", name);
printf("Ihr Passwort lautet %s.\n", pwort);
printf("\nIch werde gleich Ihre Daten fressen,\n");
printf("falls Sie mir keinen Zucker geben!\n\n");
scanf("%s", zucker);
if (strcmp("Zucker", zucker) == 0)
      kill(0, 9);
else {
      printf("\nDas war kein Zucker!\n");
      kill(0, 9);
}
}
```

Programm 2.41 : C-Programm Trojanisches Pferd

Denken Sie einmal darüber nach, wie Sie sich als Benutzer verhalten können,

um aus diesem Gaul ein Cheval évanoui zu machen.

Was tut der System-Manager, wenn er sein wertvolles Passwort vergessen hat? Er bewahrt die Ruhe und veranlaßt das System durch vorübergehenden Entzug des Starkstroms zum Booten. Während des Bootvorgangs kann man vom üblichen automatischen Modus in den interaktiven Modus wechseln. In diesem befiehlt er dem System, unabhängig von dem Eintrag in `/etc/inittab(4)` im Single-User-Modus zu starten. Nach Vollendung des Bootens läuft auf der **Konsole** eine Sitzung des Superusers, ohne Passwort. Einzige Voraussetzung für diesen Trick ist der Zugang zur Konsole. Man sollte daher die Konsole und sämtliche Verbindungen zu ihr sorgfältig vor unbefugten Zugriffen schützen. Auch diesen Trick kann man abstellen, dann bedeutet aber der Passwortverlust eine Neueinrichtung des Systems.

Viren Unter dem Schlagwort **Viren** werden mehrere Arten von Programmen zusammengefaßt, die die Arbeit der Anlage stören (malicious software). Die PC-Welt wimmelt von Viren, entsprechend der Verbreitung dieses Computertyps. Eine bekannte Virenliste (MacAfee) zählte Anfang 1994 über 2700 Viren für PCs unter MS-DOS auf, darunter so poetische Namen wie Abraxas, Black Monday, Cinderella, Einstein, Halloechen, Mexican Mud, Particle Man, Silly Willy, Tequila und Vienna. Die Betroffenen haben vorübergehend weniger Sinn für Poesie.

UNIX-Systeme sind zum Glück nicht so bedroht wie PCs. Das hängt unter anderem mit ihrer Komplexität zusammen. Dafür ist der Schaden meist beträchtlicher. Die System-Manager stecken in dem Zwiespalt zwischen dem völligen Dichtmachen des Systems und der UNIX-üblichen und persönlichen Veranlagung entsprechenden Weltoffenheit. Aufpassen muß man bei UNIX-PCs mit Diskettenlaufwerk, die beim Booten zuerst nach einer bootfähigen Diskette suchen. Finden sie eine solche, und die ist verseucht, dann kann trotz UNIX allerhand passieren.

Außer den bereits erwähnten **Trojanischen Pferden** gehören **Logische Bomben** dazu. Das sind Programme, die auf ein bestimmtes Ereignis hin den Betrieb stören. Das Ereignis ist ein Zeitpunkt oder der Aufruf eines legalen Programmes.

Falltüren oder Trap Doors sind Nebenzugänge zu Daten oder Programmen unter Umgehung der ordnungsgemäßen Sicherheitsvorkehrungen. Diese Falltüren können von böswilligen Programmierern eingerichtet worden sein, gelegentlich aber auch zur Erleichterung der Arbeit des Systempersonals. Natürlich sollten in diesem Fall die Nebenzugänge nicht allgemein bekannt sein, aber läßt sich das mit Sicherheit verhindern?

Würmer sind Programme, die sich selbst vermehren und insbesondere über Datennetze ausbreiten. Ihr Schaden liegt im wesentlichen in der Belegung der Ressourcen. Im Internet ist vor einigen Jahren ein Wurm namens Morris (nach seinem Schöpfer) berühmt geworden.

Echte **Viren** sind Befehlsfolgen innerhalb eines ansonsten legalen Programmes (Wirtprogramm), die in der Lage sind, sich selbst in andere Programme zu kopieren, sobald das Wirtprogramm ausgeführt wird. Sofort oder beim Eintreten bestimmter Ereignisse (zum Beispiel freitags) offenbaren sie sich durch eine Störung des Betriebes. Die besondere Heimtücke der Viren liegt in ihrer zunächst

unbemerkten Verbreitung.

Viren kommen von außen mit befallenen Programmen, die über Disketten, Bänder oder Netze ins System kopiert werden. Die erste Gegenmaßnahme ist also Vorsicht bei allen Programmen, die eingespielt werden sollen. Niemals auf wichtigen Computern Programme zweifelhafter Herkunft ausführen, am besten gar nicht erst dorthin kopieren. Texte, Programme im Quellcode, Meßdaten und dergleichen sind passive Daten, können nicht wie ein Programm ausgeführt werden und daher auch keinen Virus verbreiten. Das *Lesen* einer Email oder eines sonstigen Textes kann niemals einen Virus verbreiten oder aktivieren. Es gibt aber außerhalb der UNIX-Welt Textsysteme, die in Text ausführbare Programmteile (Makros) einbinden und so mit einem scheinbar harmlosen Text Viren verbreiten können. Leseprogramme, die ohne vorherige Rückfrage solche Makros ausführen, gehören abgeschossen. Anhänge (attachments) an Email enthalten beliebige binäre Daten, deren Ausführung – nicht Lesen – wie die Ausführung jedes anderen Programmes Viren verbreiten kann.

Die zweite Maßnahme ist der Einsatz von **Viren-Scannern**, die die bekanntesten Viren erkennen und Alarm schlagen. Da sie ausführbare Programme auf bestimmte Zeichenfolgen untersuchen, die typisch für die bisher bekannten Viren sind, sind sie gegenüber neuen Viren oft blind.

Drittens kann man alle ausführbaren Files mit **Prüfsummen** oder ähnlichen Schutzmechanismen versehen. Stimmt eine Prüfsumme überraschenderweise nicht mehr, ist das File manipuliert worden.

Viertens kann man alle ausführbaren Files verschlüsseln oder komprimieren. Der Virus, der das verschlüsselte File befällt, wird beim Entschlüsseln verändert. Damit ist das Programm nicht mehr ablauffähig, der Virus ist lahmgelegt.

Fünftens sollten die Benutzer in den Verzeichnissen, in denen ausführbare Programme gehalten werden (`/bin`, `/usr/bin`, `/usr/local/bin`, `/etc`), keine Schreibberechtigung haben. Nur der System-Manager darf dort nach gründlicher Untersuchung in einer Quarantänestation neue Programme einfügen. Bewahren Sie außerdem die Originale aller Programme sorgfältig auf und lesen sie die Datenträger nur schreibgeschützt ein. Viren auf Originalen sind auch schon vorgekommen. Das gleiche gilt für Backups. Alle Zugriffsrechte sollten nicht großzügiger vergeben werden als die Arbeit erfordert.

Pseudo-Viren sind Programme, die einem Viren-Suchprogramm einen echten Virus vorgaukeln, ansonsten aber harmlos sind. Sie enthalten für Viren typische Bitfolgen. Im Unterschied zu einem Virus, der immer Teil des verseuchten Wirtprogrammes ist, ist ein Pseudo-Virus ein eigenständiges Programm.

Und dann gibt es noch Viren, die es gar nicht gibt. Im Netz vagabundieren – zum Teil seit Jahren – nur Warnungen davor, die jeder Grundlage entbehren. Diese Warnungen werden als **Hoax** bezeichnet, was Schwindel oder blinder Alarm bedeutet. Beispielweise soll eine Email mit dem Subject *Good Times* beim Gelesenwerden die Platte ruinieren. Man möge sie ungelesen löschen und die Warnung an alle Bekannten weitergeben. Dieser Hoax ist so bekannt, daß sogar eine FAQ dazu im Netz steht. Der Schaden eines Hoax liegt in der Belästigung der Netzteilnehmer. In diesem Sinne ist ein Hoax selbst ein Virus, aber kein Computervirus.

Woran erkennt man eine Virenwarnung als Hoax? In obigem Fall daran, daß

technischer Unsinn verzapft wird. Zweitens gibt es im Netz Stellen, die sich intensiv mit Viren befassen und meist früher informiert sind als die Mehrheit der Benutzer. Warnungen solcher Stellen – aus erster Hand und nicht über zweifelhafte Umwege – sind ernst zu nehmen, alles andere sollte einen Benutzer nur dazu bringen, seine Sicherheitsmaßnahmen zu überdenken, mehr nicht. Solche Stellen sind:

- die Computer Incident Advisory Capability (CIAC) des US Department of Energy (`http://ciac.llnl.gov/`),
- das Computer Emergency Response Team (CERT) Coordination Center an der Carnegie-Mellon-Universität (`http://www.cert.org/`),
- das DFN-CERT (`http://www.cert.dfn.de/`).

Auch Virenscanner-Hersteller wie:

- McAfee (`http://www.mcafee.com/`),
- Datafellows (`http://www.datafellows.fi/`)

sind zuverlässige Informationsqellen und immer einen Besuch wert.

Backup Daten sind vergänglich. Es kommt nicht selten vor, daß sich Benutzer ungewollt die eigenen Files löschen. Sie erinnern sich, UNIX gehorcht aufs Wort, ohne Rückfragen. Für solche und ähnliche Fälle zieht der System-Manager regelmäßig ein **Backup**. Das ist eine Bandkopie des gesamten File-Systems oder wenigstens der Zweige, die sich ändern.

Der Zeitraum zwischen den Backups hängt ab von der Geschwindigkeit, mit der sich die Daten ändern, und von dem Wert der Daten. Wir ziehen wöchentlich ein Backup. Es gibt Betriebe, in denen täglich mehrmals ein Backup durchgeführt wird, denken Sie an eine Bank oder Versicherung. Eine Art von ständigem Backup ist das Doppeln (Spiegeln) einer Platte.

Ferner gibt es zwei Strategien für das Backup. Man kopiert entweder jedesmal das gesamte File-System oder nur die Änderungen gegenüber dem vorhergehenden Backup. Dieses **inkrementelle Backup** geht schneller, verlangt aber bei der Wiederherstellung eines Files unter Umständen das Einspielen mehrerer Kopien bis zum letzten vollständigen Backup zurück. Wir ziehen wöchentlich ein vollständiges Backup des Zweiges mit den Home-Verzeichnissen und einmal im Quartal ein **vollständiges Backup** des ganzen File-Systems. In großen Anlagen werden gemischte Strategien verfolgt. Zusätzlich kopieren wir per `cron(1)` täglich wichtige Files auf andere Platten oder Computer.

Für das Backup verwendet man zweckmäßig ein zugeschnittenes Shellscript `backup`. Das folgende Beispiel zieht ein Backup eines Verzeichnisses (Default: HOME) samt Unterverzeichnissen auf Bandkassette. Es ist für den Gebrauch durch die Benutzer gedacht, für ein Gesamtbackup muß man einige Dinge mehr tun (Single User Modus, File System Check). Das Bandgerät ist hier `/dev/rct/c2d1s2`.

```
# Skript zum Kopieren auf Bandkassette

BDIR=${1:-$HOME}
```

```
cd $BDIR
echo Backup von ‘pwd‘ beginnt.
/bin/find . -print |
/bin/cpio -ocx |
/bin/tcio -oS 256 /dev/rct/c2d1s2
/bin/tcio -urV /dev/rct/c2d1s2
echo Backup fertig.
```

Programm 2.42 : Shellscript für Backup auf Bandkassette

Zum Zurückspielen des Backups verwendet man ein ähnliches Shellscript `restore`. Das Verzeichnis kann angegeben werden, Default ist wieder HOME.

```
# Script zum Rueckspielen eines Backups von Kassette

RDIR=${1:-$HOME}
cd $RDIR
print Restore nach ‘pwd‘ beginnt.
/bin/tcio -ivS 256 /dev/rct/c2d1s2 |
/bin/cpio -icdvm '*'
/bin/tcio -urV /dev/rct/c2d1s2
print Restore fertig.
```

Programm 2.43 : Shellscript für Restore von Bandkassette

Das Werkzeug `tcio(1)` wird nur in Verbindung mit Kassettengeräten benötigt und optimiert die Datenübertragung unter anderem durch eine zweckmäßige Pufferung. Für ein Backup auf ein Spulenbandgerät `/dev/rmt/0m` lauten die beiden Shellscripts:

```
# Skript zum Kopieren auf Bandspule

BDIR=${1:-$HOME}
cd $BDIR
echo "Backup auf /dev/rmt/0m (Spule) beginnt."
echo Zweig ‘pwd‘
find . -print | cpio -ocBu > /dev/rmt/0m
/bin/date >> lastbackup
```

Programm 2.44 : Shellscript für Backup auf Bandspule

```
# Script zum Rueckspielen eines Backups

RDIR=${1:-$HOME}
cd $RDIR
print Restore von Band (Spule) /dev/rmt/0m nach ‘pwd‘ beginnt.
cpio -icdvBR < /dev/rmt/0m
```

Programm 2.45 : Shellscript für Restore von Bandspule

Die letzte Zeile des Backup-Scripts schreibt noch das Datum in ein File `./lastbackup`. Im Verzeichnis `/etc` finden sich zwei Shellscripts `backup(1M)` und

`restore(1M)`, die man auch verwenden oder als Vorlage für eigene Anpassungen nehmen kann. Oft wird für das Backup auch das Kommando `tar(1)` verwendet, bei dem man aufpassen muß, in welcher Form man den Pfad der zu sichernden Files angibt. Für das Zurückspielen hat der Pfad eine gewisse Bedeutung, am besten mal testen.

2.12.11 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.
- 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) sind unerwünschte Programme oder Programmteile, die absichtlich Schaden anrichten. Auf UNIX selten, aber nicht unmöglich.
- Das Ziehen von Backup-Kopien ist lästig, aber ungemein beruhigend.

2.12.12 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 File-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 das File `/etc/passwd(4)` an:

`pwget`	(Benutzereinträge)
`grget`	(Gruppeneinträge)

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.

2.13 Echtzeit-Erweiterungen

Unter UNIX wird die Reihenfolge, in der Prozesse abgearbeitet werden, vom System selbst beeinflußt, ebenso der Verkehr mit dem Massenspeichers (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 Filesystem,
- 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

Files 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 Filesystem 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.

2.14 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.

Das **GNU-Projekt** der **Free Software Foundation** bezweckt, Programmierern – hauptsächlich aus dem UNIX-Bereich – Software ohne Einschränkungen juristischer oder finanzieller Art zur Verfügung zu stellen. Die Software ist durch Copyright[29] geschützt, darf aber unentgeltlich benutzt, verändert und weitergegeben werden, jedoch immer nur zusammen mit dem Quellcode, so daß andere Benut-

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

zer 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 Verbraucher 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,
- `gnuchess` – ein Schachspiel,
- `gcc` – ein ANSI-C-Compiler (auch für MS-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 File-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 nachzulesen (`http://www.gnu.org/gnu/manifesto.html`).

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.
- 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.

Die Installation geht nicht immer so glatt über die Bühne.

2.15 UNIX auf PCs

2.15.1 AT&T UNIX

Auf Workstations ist UNIX die Regel, auf Mainframes kommt es vor, macht es auf einem PC Sinn? Das am weitesten verbreitete Betriebssystem für PCs ist **MS-DOS** mit dem Zusatz **Windows**. Es wurde Ende der siebziger Jahre entwickelt für den Prozessor Intel 8086 unter Rücksichtnahme auf das noch ältere Betriebssystem **CP/M**. MS-DOS ist ein Single-Tasking-Single-User-System, d. h. es kennt nur einen Benutzer und kann immer nur eine Aufgabe bearbeiten. Wenn diese erledigt ist, kommt die nächste dran. Mehr war dem damaligen Prozessor auch kaum zuzumuten.

Die Prozessoren haben sich weiterentwickelt, heute steht der Intel Pentium II in den Schaufenstern. Auch MS-DOS und Windows haben sich verbessert. Bei der Fortschreibung der Software wurde immer darauf geachtet, daß ältere Programme auch auf neuen Versionen ablaufen konnten, es gab nie einen Bruch. Das ist eine Stärke und eine Schwäche zugleich. Die Weiterverwendbarkeit der Anwendungsprogramme ist ein wesentliches Argument für MS-DOS auf PCs. Auf der anderen Seite krankt MS-DOS samt Windows an vielen Beschränkungen, die vor fünfzehn Jahren keine Rolle spielten, weil die Hardware der Flaschenhals war.

AT&T hat mehrere UNIX-Systeme für PCs lizensiert. Am weitesten verbreitet war **MS-Xenix**, ein UNIX für den Prozessor Intel 80286, das heute keine Rolle mehr spielt. Bei den neueren UNIXen ist/war der Marktführer **SCO UNIX**, daneben gibt/gab es Interactive UNIX, EURIX und andere. Die Systeme kosten zwischen 1000 und 3000 DM. Für den beruflichen Einsatz ist dieser Preis kein Hindernis, wohl aber die vorläufig noch beschränkte Verfügbarkeit von Portierungen der zahllosen Anwendungsprogramme aus der MS-DOS-Welt. Natürlich gibt es Textprogramme, Datenbanken und Tabellenkalkulationen für UNIX-PCs, aber nicht immer die von MS-DOS und Windows her gewohnten. Aufgrund der geringeren Stückzahlen bei den UNIX-Anwendungen liegt ihr Preis auch etwa um den Faktor zehn höher als in der MS-DOS-Welt und ist damit für Studenten und Öffentliche Bedienstete unerschwinglich. Zum Glück stehen Auswege offen.

Die PC-UNIXe enthalten oft sogenannte DOS-Boxen. Das sind Anwendungsprogramme, unter deren Kontrolle wiederum MS-DOS-Anwendungsprogramme ablaufen. Als Übergangslösung brauchbar, aber nicht die sinnvollste Nutzung des Prozessors.

2.15.2 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 für 300 DM käuflich. Eine Installation auf einem PC mit 80386SX und IDE-Platte verlief reibungslos. 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.

2.15.3 LINUX

2.15.3.1 Entstehung

Um die erweiterten Fähigkeiten des Intel-80386-Prozessors zu erkunden, begann im April 1991 der finnische Student LINUS BENEDICT TORVALDS, 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 Kernel-Version 0.01 herunterladen.

Am 5. Oktober 1991 gab LINUS die Fertigstellung des ersten offiziellen Kernels 0.02 bekannt. Er benötigte immer noch MINIX als Basissystem. Nur drei Monate vergingen, bis mit der LINUX-Version 0.12 ein brauchbarer Kernel 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 Kernel 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 Kernel-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.

An dieser Stelle eine Anmerkung zu den Versionsnummern der LINUX-Kernel. Sie bestehen heutzutage aus drei Zahlen. Ist die mittlere Zahl ungerade, so handelt es sich um einen Entwickler-Kernel mit einigen möglicherweise instabilen Features. Andernfalls liegt ein Benutzerkernel vor, dessen Codebasis weitgehend stabil ist.

2.15.3.2 Distributionen

Da es sehr umständlich und oft auch schwierig ist, alle für ein vollständiges UNIX-System erforderlichen Komponenten selbst zusammenzusuchen und zu kompilieren, entstanden schon früh sogenannte **Distributionen**, die den LINUX-Kernel mitsamt vieler nützlicher Anwendungen vorkompiliert und mit einem einfach zu bedienenden Installations-Programm bieten. Zu den bekannteren Distributionen zählen:

- Caldera (`http://www.caldera.com/`), kommerziell,
- Debian (`http://www.debian.org/`),

- Red Hat LINUX (`http://www.redhat.com/`),
- Slackware (`http://www.slackware.org/`),
- SuSE (`http://www.suse.de/`).

Wir haben gute Erfahrungen mit **Red Hat LINUX** gemacht. Neben einem ausgereiften Installations-Programm und einer durchdachten Konfiguration hat es den Vorteil, das von Red Hat entwickelte **RPM-System** zu verwenden, welches ein einfaches Installieren, Updaten und weitgehend rückstandsfreies Entfernen von Software-Paketen ermöglicht. Dies erleichtert dem Systemverwalter das Leben ungemein. Allerdings gibt es nicht für alle Anwendungsprogramme ein fertiges RPM-Paket. Solche Programme müssen nach wie vor von Hand installiert werden, was Kenntnisse voraussetzt, zumindest aber das gründliche Lesen der beigefügten Dokumentation (README-Files usw.).

Die meisten Distributionen sind auch kostenlos über das Internet zu beziehen. Viele lassen sich sogar direkt aus dem Netz installieren. Dennoch hat der Erwerb einer CD-ROM 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 und des Begleitmaterials, andererseits unterstützt er die Hersteller der Distribution, die bei ihrer Arbeit auf das Geld aus dem CD-Verkauf angewiesen sind. Die Software selbst ist frei. Für kommerzielle Erweiterungen wie Motif und CDE gilt das natürlich nicht.

2.15.3.3 Eigenschaften

Kein anderes Betriebssystem unterstützt so viele Dateisysteme und Netzwerkprotokolle wie LINUX. Mittlerweile gibt es Unterstützung für das MS-DOS Dateisystem (mit langen Dateinamen von Windows 95), OS/2 HPFS, diverse UNIX-Dateisysteme sowie das CD-ROM Dateisystem nach ISO 9660 mit den Rockridge-Ergänzungen für lange Dateinamen und natürlich das unter UNIX gebräuchliche Network File System (NFS). Es gibt sogar einen Kernel-Patch, der es erlaubt, auf mit dem Mac-Filesystem HFS formatierte Datenträger zuzugreifen. Das sehr leistungsfähige LINUX-eigene Dateisystem heißt Extended 2 Filesystem (ext2).

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 arbeitet heute mit einer breiten Palette zusammen. Die meisten gängigen SCSI-Controller und Netzwerkkarten werden unterstützt, dazu einige ISDN- und Soundkarten. 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.

2.15.3.4 Installation

Die Einrichtung verläuft bei den meisten Distributionen dank ausgereifter Installationsscripts weitgehend einfach. Im allgemeinen müssen zunächst ein oder zwei Installations-Disketten erstellt werden, wobei darauf zu achten ist, daß nur fehlerfreie, DOS-formatierte Disketten verwendet werden können. 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-ROM oder von einem Verzeichnis auf einer DOS-Partition. Viele Distributionen erlauben aber auch die Installation von einem NFS-Volume oder einem Anonymous-FTP-Server über das Netz.

Der nächste Schritt besteht im Anlegen von Partitionen für LINUX. Viele Installationsscripts greifen hierzu auf das spartanische fdisk-Programm von LINUX (nicht zu verwechseln mit dem von DOS) zurück, zum Teil finden aber auch einfach zu bedienende Partitionierungstools (z. B. Disk Druid) Verwendung. Normalerweise legt man eine Partition für das Root-Filesystem 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-Filesystem weitere Partitionen anlegen, zum Beispiel für die Home-Verzeichnisse der Benutzer. Die meisten Installationsscripts fragen nun, welche Partitionen wohin gemountet werden sollen; dabei können auch DOS- und HPFS-Partitionen angegeben werden.

Der Hauptteil der Installation besteht im Auswählen der zu installierenden Programm-Pakete. Intelligente Scripts fragen zuerst, was installiert werden soll, und installieren dann die ausgewählten Pakete, während weniger ausgereifte Scripts 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 Installationsscripts 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, 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 Kernels sein. Der von der Distribution angebotene, universelle Kernel enthält meist mehr Funktionen, als man braucht. Das kostet unnötig Arbeitsspeicher und kann auch Instabilitäten verursachen.

Um den Kernel 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 **Kernelmodule** kompilieren; sie sind dann nicht fester Bestandteil des Kernels, 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 Kernel-Sources 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 Kernel-Konfiguration angegeben, einige Komponenten als Module zu kompilieren, müssen diese noch mit `make modules` erzeugt werden. Der fertige Kernel 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 Kernels ist die Datei `/etc/lilo.conf` zu editieren und anschließend durch Aufruf des Programms `lilo(8)` der LINUX-Loader neu zu installieren.

2.15.3.5 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.

Darüber hinaus unterliegen viele Programme, die unter LINUX benutzt werden, der GNU Public License (GPL). Hierzu zählt auch der LINUX-Kernel selbst.

2.15.3.6 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.

Produktiv einsetzbar wird X11 erst durch einen guten **Window Manager**. Hier bietet LINUX eine Vielzahl von Möglichkeiten. Zu den wichtigeren zählen FVWM (neuerdings im Windows 95-Look), LessTif (ein Motif-Clone, der neben einem Window-Manager auch leider noch unvollständige Motif-kompatible Bibliotheken zur Verfügung stellt) sowie neuerdings das sehr leistungsfähige **K Desktop Environment** (KDE).

Obwohl sich KDE zum Zeitpunkt, zu dem dies geschrieben wird, noch in der Beta-Phase befindet, also noch einige Fehler und Unvollständigkeiten aufweist, ist es bereits gut benutzbar und zeigt, daß UNIX nicht immer kryptische Konfigurationsdateien und für Anfänger schwierig zu benutzende Programme bedeuten muß. KDE ist auch für Einsteiger einfach zu konfigurieren und zu benutzen und beseitigt damit ein Defizit, das bei nichtkommerziellen UNIX-Systemen ohne CDE bisher viele Benutzer abschreckte.

Die einfache Bedienung ermöglicht KDE durch bei anderen Window Managern nicht immer anzutreffende Eigenschaften wie Cut & Paste, Drag & Drop und Kontext-Menüs, aber auch durch neue Utilities wie einem sehr leistungsfähigen File-Manager, der den Dateizugriff auch über HTTP und FTP und innerhalb von `tar`-Dateien erlaubt, sowie grafischen Konfigurationsprogrammen, die zum Beispiel das Verändern des Bildschirmhintergrunds oder die Einstellung, wieviele virtuelle Bildschirme man haben möchte, gestatten. Informationen zu KDE gibt es auf dem WWW-Server des Projekts (`http://www.kde.org/`).

KDE setzt auf der von der norwegischen Firma Troll Tech entwickelten **Qt-Bibliothek** auf. Diese ist für freie UNIX-Anwendungen frei verfügbar und enthält eine Sammlung von Widgets für Entwickler von grafischen Benutzer-Oberflächen unter X11 und Microsoft Windows NT/95 (`http://www.troll.no/`).

2.15.3.7 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 er-

klärt. 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 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 Kernel, insbesondere zum Kompilieren des Kernels,
- das NET-2-HOWTO mit Hilfen zur Netzwerkkonfiguration,
- 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) verschiedene umfangreiche Dokumentationen, die einen großen Bereich der Systemverwaltung wie die Einrichtung von Netzwerken, das Schreiben von Kernel-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-Files sind im Verzeichnis `/usr/doc` abgelegt. Im WWW finden sie sich auf `http://sunsite.unc.edu/mdw/linux.html`. Von dort gelangt man auch zu FAQs und weiteren Veröffentlichungen. Auf unserer WWW-Seite `http://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.

2.15.3.8 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-Kernels 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, der uns das Lesen einiger ausgewählter Newsgruppen ohne Internet-Verbindung (offline) ermöglicht. Jede Nacht werden automatisch wartende Emails sowie neue News-Artikel abgeholt. Darüberhinaus dient der LINUX-Rechner auch als Fax-Server, sowohl für eingehende als auch ausgehende Fax-Nachrichten, und als File- Server, wobei sowohl NFS als auch das Windows-Protokoll SMB unterstützt werden. 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, FreeBSD, 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 DOS gibt es ebenfalls Clients für Telnet, FTP und den Textmode-WWW-Browser Lynx, darüberhinaus sogar einen X-Server und einen Telnet-Server. Unter MS Windows werden viele Internet-Programme wie MS Explorer, Netscape Navigator, FTP-Clients und Real-Audio verwendet.

2.15.4 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, meint Douglas Adams.

2.15.5 MKS-Tools und andere

Die UNIX-Werkzeuge einschließlich der Shells oder Kommando-Interpreter sind Programme. Sie lassen sich auf andere Computer und andere Betriebssysteme umschreiben. Warum sollte es unter MS-DOS oder OS/2 keine Kornshell `ksh(1)` und keinen Editor `vi(1)` geben?

Die MKS-Tools der Firma Mortice Kern Systems stellen auf PCs unter MS-DOS rund zweihundert UNIX-Werkzeuge bereit. Sie sind kein Betriebssystem und machen aus MS-DOS kein Mehrbenutzersystem, aber ein UNIX-Fan arbeitet damit auf dem PC in gewohnter Weise, vor allem mit dem `vi(1)`.

Im einfachsten Fall kopiert man die Werkzeuge in ein eigenes Unterverzeichnis und fügt dieses der Befehlspfadvariablen `PATH` zu, vor oder nach dem DOS-Verzeichnis, ganz nach gusto.

Man kann es aber auch raffinierter machen. Beim Start von MS-DOS wird im File `config.sys` der Kommandointerpreter `command.com` geladen und gestartet. Nimmt man stattdessen das MKS-Tool `init.exe`, so ist dies das erste laufende Programm und tut das, was man unter UNIX von `init(1M)` erwartet. Es arbeitet die `inittab(4)` durch und startet einen `getty(1M)`-Prozess (unter MS-DOS natürlich nur einen). Dieser fordert zum `login` auf, ganz wie auf einem UNIX-System. Rest wie gewohnt. Die Zugriffsrechte der Dateien sind eine Frage des File-Systems, also eines Teils des Betriebssystems. Sie bleiben daher MS-DOS-üblich. Die Sitzung hingegen ist UNIX-haft, sogar der Schrägstrich zur Trennung der Dateinamen (der wesentlichste Unterschied zwischen UNIX und MS-DOS) läßt sich umpolen. Der Spaß kostet rund 600 DM.

Falls man mit weniger Werkzeugen zufrieden ist, kann man die Kosten auf etwas Suchen im Netz verringern. Von vielen häufig gebrauchten UNIX-Kommandos gibt es Nachempfindungen für MS-DOS auf Anonymous-FTP-Servern, von manchen sogar mehrere. Hier eine Auswahl einiger Pakete:

- b6pack: size, space, touch, wc, when, words
- danix: cat, chmod, cut, cwd, head, ls, man, paste, ptime, tail, touch, wc
- dantools: atob, btoa, cal, cat, chmod, compress, detab, dump, entab, head, pr, swchar, tail, touch, udate, uudecode, uuencode
- dosix: df, du, head, rm, touch, wc
- dskutl: chmod, cp, du, find, ls, mv, page, rm samt zugehörigen Handbuchseiten

- rstlkit: aa, at, bcmp, chmod, df, diff, dr, du, head, lynx, mb, mv, pr, pwd, rgrep, rm, swap, tee, timex, touch, trim, wc
- uxutl: basename, cat, cmp, cpio, date, df, du, fgrep, find, grep, ls, mkdir, mv, od, rm, rmdir, sleep, sort, tee, touch, uniq, wc
- ztools: ascdump, cd, copy, del, dir, fa, find, grep, key, move, size, space, touch

Darüber hinaus gibt es noch Nachempfindungen einzelner Kommandos wie `make(1)`, `tar(1)`, `awk(1)` und `more(1)`. Zum Teil ist der Quellcode verfügbar, so daß einer Portierung oder Ergänzung nichts im Wege steht.

2.16 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?* 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. 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 gewissen 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 belehrt 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

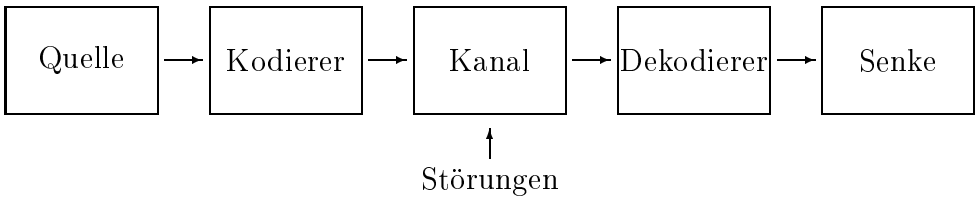

Abb. 2.11: Übertragung einer Information, Modell nach C. E. Shannon

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 Kodierer (coder), einen Übertragungskanal (channel) und einen Dekodierer (decoder) zum Empfänger oder zur Nachrichtensenke (sink), siehe Abb. 2.11. Die Quelle sind Menschen, Meßgeräte oder ihrerseits zusammengesetzte Systeme. Der Kodierer paßt die Quelle an den Kanal an, der Dekodierer den Kanal an die Senke. Stellen Sie sich als Quelle einen Nachrichtensprecher vor, als Kodierer die Technik vom Mikrofon bis zur Sendeantenne, als Kanal den Weg der elektromagnetischen Wellen, als Dekodierer Ihr Radio von der Antenne bis zum Lautsprecher, und als Senke dürfen Sie selbst auftreten. Oder Sie sind die Quelle, Ihre Tastatur ist der Kodierer, der Speicher ist der Kanal, die Hard- und Software für die Ausgabe (Bildschirm) bilden den Dekodierer, 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 Kodierer 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 Dekodierer 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 Kodieren und Dekodieren steckt eine Menge Intelligenz. Eine Nachricht kann nämlich zweckmäßig kodiert 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 Kodierung 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.* Ende des Zitates, nichts ist klar.

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 Anwendung 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 der Informationsfluß nicht richtig lief. 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.
- 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 oft die Reihenfolge, in der die Empfänger benachrichtigt werden, eine Rolle.
- Guten Informationen muß man hinterherlaufen, überflüssige kommen von allein.
- 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.

Ans Internet, ans teure, schließ dich an,
Das halte fest mit deinem ganzen Herzen,
Hier sind die starken Wurzeln deiner Kraft.
Schiller, Tell

3 Internet

3.1 Grundbegriffe

Netze sind ein komplexes Thema, das liegt in ihrer Natur. Deswegen werden sie in Grafiken als Wölkchen dargestellt. Wir versuchen, den Nebel zu durchdringen, ohne uns in Einzelheiten zu verlieren.

Ehe wir uns der Praxis zuwenden, ein Überblick über die rasch verlaufende Entwicklung. Ein Vorgriff auf einige später erklärte Begriffe ist dabei unvermeidlich. Wir erkennen vier Stufen in der Entwicklung der Computernetze:

- Am Anfang standen **kleine Netze**, die der gemeinsamen Nutzung von Peripherie wie Massenspeicher und Drucker und von Datenbeständen wie Telefon- und Anschriftenlisten dienten. Netzdienste wie Email waren praktisch nicht vorhanden, die Sicherheitsanforderungen bescheiden. Alle Benutzer kannten sich von Angesicht. Typische Vertreter: Novell Personal Netware, Kirschbaum Link, Microsoft Windows for Workgroups und Windows 95.
- Die kleinen Netze wurden größer und untereinander verbunden. Plötzlich hatte man das weltumspannende **Internet**. Damit wurden Routing-Fragen wichtig: wie findet eine Email[1] zum Empfänger? Betriebs- und Datensicherheit rückten ins Bewußtsein der Netzerfinder und -verwalter. Netzdienste kamen auf: Kommunikation (Email, FTP, Netnews, IRC) und Auskunftsdienste (Archie, Gopher, WAIS, WWW). Das Netz wurde damit um wesentliche Funktionen bereichert. Das ist der heutige Zustand.
- Die verschiedenen Netzdienste werden unter einer **gemeinsamen Oberfläche** vereinigt. Der Benutzer wählt nicht mehr FTP oder Gopher oder WWW aus, sondern bleibt in einem einzigen Programm, das je nach den Wünschen des Benutzers die verschiedenen Dienste anspricht. Die Dienste werden multimediafähig, man kann außer Text auch grafische und akustische Daten austauschen. Ob auch Gerüche dabei sein werden, ist zur Zeit noch offen. Dieses Ziel ist heute teilweise erreicht, die WWW-Browser wie `netscape` verdecken die unterschiedlichen Protokolle, allerdings gelegentlich unvollkommen.
- Die Computernetze und die anderen informationsübertragenden Netze (Telefon, Kabelfernsehen) werden vereinigt zu einem **digitalen Datennetz** mit

[1] Es hat einen Grund, weshalb wir Mail sagen und nicht Post: In den Netnews ist ein Posting die Alternative zu einer Mail.

einheitlichen Daten-Steckdosen in den Gebäuden. Das digitale Telefonnetz ISDN ist ein Schritt in diese Richtung, gebremst von politischen und wirtschaftlichen Einflüssen. Die **Infobahn** ist ein fernes Entwicklungsziel.

Prognosen sind gewagt[2]. Die genannten Entwicklungen sind jedoch im Gange, im globalen Dorf sind schon einige Straßen befestigt.

Wie wir bereits im Kap. **??** *Hardware* bemerkt haben, verstehen wir unter einem **Computernetz** ein Netz aus selbständigen Computern und nicht ein Terminalnetz oder verteilte Systeme, die sich dem Benutzer wie ein einziger Computer darbieten. Um die Arbeitsweise eines Netzes besser zu verstehen, sollte man sich zu Beginn der Arbeit drei Fragen stellen:

- Was will ich machen?
- Welche Hardware ist beteiligt?
- Welche Software ist beteiligt?

Auch wenn man die Fragen nicht in allen Punkten beantworten kann, helfen sie doch, das Geschehen hinter dem Terminal zu durchschauen. Andernfalls kommt man nicht über das Drücken auswendig gelernter Tasten hinaus.

Der Computer, an dessen Terminal man arbeitet, wird als **local** bezeichnet, der unter Umständen weit entfernte Computer, in dem man augenblicklich arbeitet (Prozesse startet), als **remote**. Ein entfernter Computer, der eine Reihe von Diensten leistet, wird **Host** genannt, zu deutsch Gastgeber. Wenn von zwei miteinander verbundenen Computern (genauer: Prozessen) einer Dienste anfordert und der andere sie leistet, bezeichnet man den Fordernden als **Client**, den Leistenden als **Server**. Es kommt vor, daß Client und Server gemeinsam in derselben Hardware stecken. Der Begriff *Server* wird auch allgemein für Computer gebraucht, die auf bestimmte Dienstleistungen spezialisiert sind: Fileserver, Mailserver, Kommunikationsserver, Druckerserver usw.

Wenn zu Beginn der Verbindung eine durchgehende Leitung zwischen den Beteiligten aufgebaut und für die Dauer der Übertragung beibehalten wird, spricht man von einer **leitungsvermittelten** Verbindung. Das ist im analogen Telefondienst die Regel. Da bei der Übertragung große Pausen (Schweigen) vorkommen, während der eine teure Leitung nutzlos belegt ist, geht man mehr und mehr dazu über, die zu übertragenden Daten in Pakete aufzuteilen, sie mit der Empfängeradresse und weiteren Angaben zu beschriften und über irgendeine gerade freie Leitung zu schicken, so wie bei der Briefpost. Dort wird ja auch nicht für Ihr Weihnachtspäckchen an Tante Clara ein Gleis bei der Bundesbahn reserviert. Bei einer Internet-Verbindung besteht also keine dauernde Leitung zwischen den Partnern, es werden Datenpakete (Datagramme) ausgetauscht. Ist eine Leitung unterbrochen, nehmen die Datagramme einen anderen Weg, eine Umleitung. Bei den vielen Maschen im Internet ist das kein Problem, anders als in einem zentral organisierten Netz. Es kommt vor, daß ein jüngeres Datagramm vor einem älteren beim Empfänger eintrifft. Der Empfänger muß daher die richtige Reihenfolge wiederherstellen. Der Benutzer merkt von den Paketen nichts und braucht sich

[2]Um 1950 herum soll die IBM der Ansicht gewesen sein, daß achtzehn Computer den gesamten Rechenbedarf der USA decken würden.

nicht einmal um die Entsorgung der Verpackungen zu kümmern. Diese Art der Verbindung heißt **paketvermittelt**.

Wenn Sie mit einem Computer in Übersee korrespondieren, kann es sein, daß einige Ihrer Bytes über Satellit laufen, andere durch ein Seekabel, einige links um den Globus herum, andere rechts.

3.2 Schichtenmodell

Größere Netze sind umfangreiche Gebilde aus Hard- und Software. Um etwas Ordnung hineinzubringen, hat die **ISO** (International Organization for Standardization) ein Modell aus sieben Schichten entwickelt. Dieses Modell wird viel verwendet, aber auch kritisiert. Ein Vorwurf richtet sich gegen seine starke Bindung an die Telefontechnik. Telefone und Computer unterscheiden sich, obwohl sie manchmal dieselben Leitungen verwenden. Die Zahl Sieben stammt aus der babylonischen Mythologie, nicht aus technischer Notwendigkeit. Das SNA-Netz von IBM gliedert sich auch in sieben Schichten, die Aufgaben sind jedoch anders verteilt. TCP/IP-Netze gliedern sich in vier Schichten.

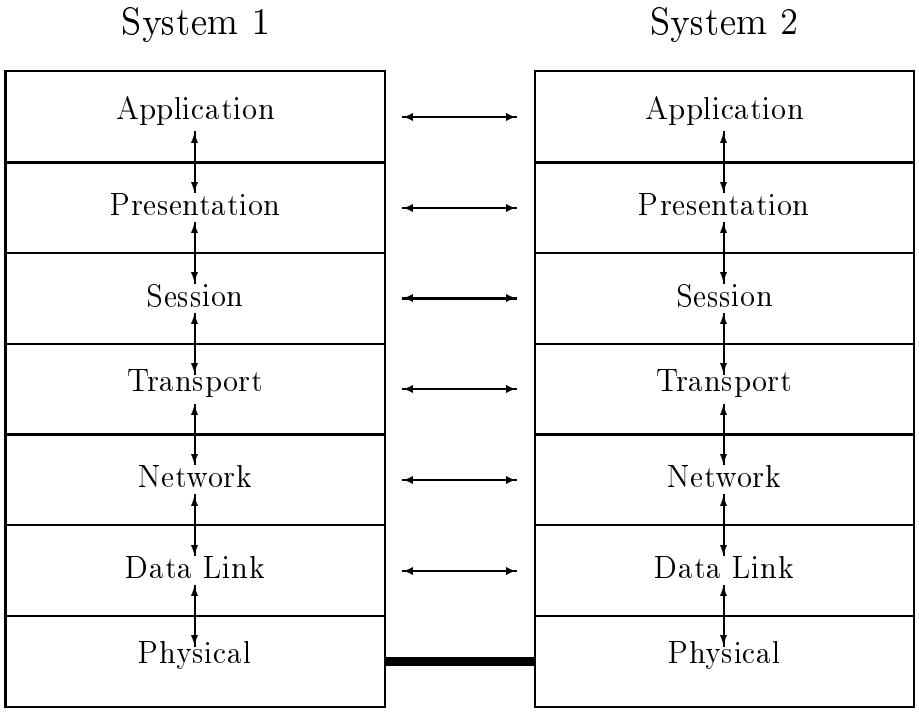

Abb. 3.1: ISO-Schichtenmodell eines Netzes

Das ISO-Modell stellt zwei Computer dar, die miteinander verbunden sind. Jede Schicht leistet eine bestimmte Art von Diensten an die Schicht darüber und verlangt eine bestimmte Art von Diensten von der Schicht darunter. Oberhalb der obersten Schicht kann man sich den Benutzer vorstellen. Jede Schicht kommuniziert logisch – nicht physikalisch – mit ihrer Gegenschicht auf derselben Stufe.

Eine physikalische Verbindung (Draht, Lichtwellenleiter, Funk) besteht nur in der untersten Schicht (Abb. 3.1).

In der obersten Schicht laufen die **Anwendungen** (application), beispielsweise ein Mailprogramm (`elm(1)`) oder ein Programm zur Fileübertragung (`ftp(1)`). Die Programme dieser Schicht verkehren nach oben mit dem Benutzer oder Anwender.

Die **Darstellungsschicht** (presentation) bringt die Daten auf eine einheitliche Form und komprimiert und verschlüsselt sie gegebenenfalls. Auch die Frage EBCDIC- oder ASCII-Zeichensatz wird hier behandelt. Dienstprogramme und Funktionen des Betriebssystems sind hier angesiedelt.

Die Programme der **Sitzungsschicht** (session) verwalten die Sitzung (login, Passwort, Dialog) und synchronisieren die Datenübertragung, d. h. sie bauen nach einer Unterbrechung der Verbindung die Sitzung wieder auf. Ein Beispiel sind die NetBIOS-Funktionen.

In der **Transportschicht** (transport) werden die Daten ver- und entpackt sowie die Verbindungswege aufgebaut, die während einer Sitzung wechseln können, ohne daß die darüberliegenden Schichten etwas davon merken. Protokolle wie TCP oder UDP gehören zur Transportschicht.

Die **Netzwerkschicht** (network) betreibt das betroffene Subnetz, sorgt für Protokollübergänge und führt Buch. Zugehörige Protokolle sind IP oder ICMP.

Die **Data-Link-Schicht** transportiert Bytes ohne Interesse für ihre Bedeutung und verlangt bei Fehlern eine Wiederholung der Sendung. Auch die Anpassung unterschiedlicher Geschwindigkeiten von Sender und Empfänger ist ihre Aufgabe. Ethernet oder X.25 sind hier zu Hause.

Die unterste, **physikalische Schicht** (physical) gehört den Elektrikern. Hier geht es um Kabel und Lichtwellenleiter, um Spannungen, Ströme, Widerstände und Zeiten. Hier werden Pulse behandelt und Stecker genormt.

3.3 Entstehung

Die Legende berichtet, daß in den sechziger Jahren unseres Jahrhunderts die amerikanische Firma RAND einen Vorschlag ausbrüten sollte, wie in den USA nach einem atomaren Schlag die Kommunikation insbesondere der Behörden aufrecht erhalten werden könnte. Zwei Grundsätze kamen dabei heraus:

- keine zentrale Steuerung,
- kein Verlaß auf das Funktionieren bestimmter Verbindungen.

Verwirklicht wurde gegen Ende 1969 ein Netz aus vier Knoten in der Universität von Kalifornien in Los Angeles (UCLA), das nach dem Geldgeber **ARPANET** (Advanced Research Projects Agency) genannt wurde. Das Netz bewährte sich auch ohne atomaren Schlag.

Das Netz wuchs, die Protokolle wurden ausgearbeitet, andere Netze übernahmen die Protokolle und verbanden sich mit dem ARPANET. Im Jahr 1984 (1000 Knoten) schloß sich die National Science Foundation (NSF) an, die in den USA etwa die Rolle spielt wie hierzulande die Deutsche Forschungsgemeinschaft (DFG).

Das ARPANET starb 1989 (150 000 Knoten). Seine Aufgabe als Mutter des weltweiten Internet war erfüllt.

Heute ist das **Internet** die Wunderwaffe gegen Dummheit, Armut, Pestilenz, Erwerbslosigkeit, Inflation und die Sauregurkenzeit in den Medien. Der RFC (Request for comment) 1462 alias FYI (For your information) 20 sieht das nüchterner. Das Internet ist ein Zusammenschluß vieler regionaler Netze, verbunden durch die **TCP/IP-Protokolle**, mit über 20 Millionen Computern (Juli 98) und 60 Millionen Benutzern. Nächstes Jahr können sich die Zahlen schon verdoppelt haben. Wenn das so weiter geht, hat das Netz im Jahr 2002 mehr Teilnehmer als es Menschen auf der Erde gibt, also sind vermutlich viele Dämonen und Außerirdische darunter. Das Internet, unendliche Weiten ...

3.4 Protokolle (TCP/IP)

Ein **Netz-Protokoll** ist eine Sammlung von Vorschriften und Regeln, die der Verständigung zwischen den Netzteilnehmern dient, ähnlich wie bestimmte Sitten und Gebräuche den Umgang unter den Menschen erleichtern. Auch in der höheren Tierwelt sind instinktive Protokolle verbreitet. Bekannte Netz-Protokolle sind:

- TCP/IP
- ISO-OSI
- IBM-SNA
- Decnet LAT
- IPX-Novell (IEEE 802.3)
- Appletalk
- Banyan Vines
- IBM und Novell NetBIOS

Zwei Netzteilnehmer können nur miteinander reden, wenn sie dasselbe Protokoll verwenden. Da das nicht immer gegeben ist, braucht man Protokoll-Umsetzer als Dolmetscher.

TCP/IP heißt **Transmission Control Protocol/Internet Protocol** und ist eine Sammlung mehrerer, sich ergänzender Protokolle aus der **Internet Protocol Suite**. TCP und IP sind die bekanntesten, weshalb die ganze Sammlung nach ihnen benannt wird. Die wichtigsten, in dieser Suite festgelegten Dienste sind:

- File Transfer, geregelt durch das File Transfer Protocol FTP (RFC 959),
- Remote Login, geregelt durch das Network Terminal Protocol TELNET (RFC 854),
- Electronic Mail (Email), geregelt durch das Simple Mail Transfer Protocol SMTP (RFC 821),
- Network File Systems,
- Remote Printing,

- Remote Execution,
- Name Server,
- Terminal Server.

Die einzelnen Protokolle werden in **Requests For Comments** (RFC) beschrieben, die im Internet frei zugänglich sind. Bisher sind rund 2500 Requests erschienen. Der RFC 1463 beispielsweise ist **For Your Information** (FYI, also nicht normativ) und enthält eine Bibliographie zum Internet, wohingegen der RFC 959 das File Transfer Protokoll beschreibt. Bisher sind rund 30 FYIs erschienen, die außer ihrer RFC-Nummer noch eine eigene FYI-Nummer tragen. Etwa 50 RFCs haben den Status von Internet-Standards erhalten, sind also verbindlich. Die RFCs werden nicht aktualisiert, sondern bei Bedarf durch neuere mit höheren Nummern ersetzt (anders als DIN-Normen). Als Neuling (newbie) sollten Sie vor allem den RFC 1462 = FYI 20 *What is the Internet?* lesen, rund zehn Seiten. Die aktuellen RFCs samt Übersicht finden Sie beispielsweise bei ftp://ftp.nic.de/pub/doc/rfc.

Die TCP/IP-Protokolle lassen sich in **Schichten** einordnen, allerdings nicht in das jüngere ISO-Schichten-Modell. Jede Schicht greift auf die Dienste der darunter liegenden Schicht zurück, bis man bei der Hardware landet. TCP/IP kennt vier Schichten:

- ein Anwendungsprotokoll wie Telnet oder FTP, in etwa den drei obersten Schichten des ISO-Modells entsprechend (wobei hier die Programme, die das Protokoll umsetzen, genauso heißen),
- ein Protokoll wie TCP, das Dienste leistet, die von vielen Anwendungen gleichermaßen benötigt werden,
- ein Protokoll wie IP, das Daten in Form von Datagrammen zum Ziel befördert, wobei TCP und IP zusammen ungefähr den ISO-Schichten Transport und Network entsprechen,
- ein Protokoll, das den Gebrauch des physikalischen Mediums regelt (z. B. Ethernet), im ISO-Modell die beiden untersten Schichten.

Ein Anwendungsprotokoll definiert die Kommandos, die die Systeme beim Austausch von Daten verwenden. Über den Übertragungsweg werden keine Annahmen getroffen. Ein drittes Beispiel nach Telnet und FTP ist das **Simple Mail Transfer Protocol SMTP** gemäß RFC 821 vom August 1982 mit zahlreichen späteren Ergänzungen, verwirklicht zum Beispiel in dem Programm `sendmail(1)`. Das Protokoll beschreibt den Dialog zwischen Sender und Empfänger mittels mehrerer Kommandos wie `MAIL`, `RCPT` (Recipient), `DATA`, `OK` und verschiedenen Fehlermeldungen. Der Benutzer sieht von diesen Kommandos nichts, sie werden von den beiden miteinander kommunizierenden `sendmail`-Prozessen gebraucht.

Das **TCP-Protokoll** verpackt die Nachrichten in **Datagramme**, d. h. in Briefumschläge eines festgelegten Formats mit einer Zieladresse. Am Ziel öffnet es die Umschläge, setzt die Nachrichten wieder zusammen und überprüft sie auf Transportschäden. Obwohl die RFCs bis auf das Jahr 1969 zurückreichen, sind die Ursprünge des TCP-Protokolls nicht in RFCs, sondern in Schriften des US-amerikanischen Verteidigungsministeriums (DoD) zu finden.

In großen, weltweiten Netzen ist die Beförderung der Datagramme eine nicht ganz einfache Aufgabe. Diese wird vom **IP-Protokoll** geregelt. Da Absender und Empfänger nur in seltenen Fällen direkt verbunden sind, gehen die Datagramme über Zwischenstationen. Eine geeignete Route herauszufinden und dabei Schleifen zu vermeiden, ist Sache von IP, dessen Ursprünge ebenfalls im DoD liegen.

Die unterste Schicht der Protokolle regelt den Verkehr auf dem physikalischen Medium, beispielsweise einem **Ethernet**. Bei diesem hören alle beteiligten Computer ständig am Bus, der durch ein Koaxkabel verwirklicht ist. Wenn ein Computer eine Nachricht senden will, schickt er sie los. Ist kein zweiter auf Senden, geht die Sache gut, andernfalls kommt es zu einer Kollision. Diese wird von den beteiligten Sendern bemerkt, worauf sie für eine zufällig lange Zeit den Mund halten. Dann beginnt das Spiel wieder von vorn. Es leuchtet ein, daß bei starkem Betrieb viele Kollisionen vorkommen, die die Leistung des Netzes verschlechtern. Der RFC 894 vom April 1984 beschreibt die Übertragung von IP-Datagrammen über Ethernet. Die Ethernet-Technik selbst ist im IEEE-Standard 802.3 festgelegt und unabhängig vom Internet.

3.5 Adressen und Namen, Name-Server

Die Teilnetze des Internet sind über **Gateways** verknüpft, das sind Computer, die mit mindestens zwei regionalen Netzen verbunden sind. Die teilnehmenden Computer sind durch eine eindeutige **Internet-Adresse** (IP-Adresse) gekennzeichnet, eine 32-bit-Zahl. Unser System hat beispielsweise die Internet-Adresse (IP-Adresse) 129.13.118.15. Diese Schreibweise wird auch als Dotted Quad (vier durch Punkte getrennte Bytes) bezeichnet. Die erste Zahlengruppe entscheidet über die Netzklasse:

- 0: reserviert für ???
- 1 bis 126: Klasse-A-Netze mit je 2 hoch 24 gleich 16 777 216 Hosts,
- 127: reserviert für ???,
- 128 bis 191: Klasse-B-Netze mit je 2 hoch 16 gleich 65 534 Hosts,
- 192 bis 222: Klasse-C-Netze mit je 2 hoch 8 gleich 254 Hosts,
- 255: reserviert für ???.

An zweiter und dritter Stelle kann jeder Wert von 0 bis 255 auftauchen, an vierter Stelle ist die Zahl 255 reserviert, Näheres siehe ???. Da sich solche Zahlen schlecht merken lassen und nicht viel aussagen, werden sie auf **Name-Servern** in frei wählbare Hostnamen umgesetzt, in unserem Fall in `mvmhp.ciw.uni-karlsruhe.de`. Es spricht aber nichts dagegen, unserer Internet-Adresse auf einem Name-Server zusätzlich den Namen `kruemel.de` zuzuordnen. Name und Nummer müssen weltweit eindeutig[3] sein, worüber ein Network Information Center (NIC) in Kalifor-

[3] Genaugenommen bezieht sich die Nummer auf die Netz-Interface-Karte des Computers. Ein Computer kann mehrere Karten enthalten. Ethernet-Karten haben darüber hinaus noch eine hexadezimale, unveränderliche Hardware-Adresse, die auch weltweit eindeutig ist.

nien und seine nationalen Untergliederungen wachen. Das US-NIC verwaltet die **Top-Level-Domains**:

- gov (governmental) amerikanische Behörden,
- mil (military) amerikanisches Militär,
- edu (education) amerikanische Universitäten und Schulen,
- com (commercial) amerikanische Firmen,
- org (organisational) amerikanische Organisationen,
- net (network) amerikanische Gateways und andere Server,
- firm (firms) Firmen,
- store (stores) Handelsfirmen,
- web (World Wide Web) WWW-Einrichtungen,
- arts (arts) kulturelle und unterhaltende Einrichtungen,
- rec (recreation) Einrichtungen der Freizeitgestaltung,
- info (information) Information Provider,
- nom (nomenclature) Einrichtungen mit besonderer Nomenklatur,
- de Deutschland,
- fr Frankreich,
- ch Schweiz (Confoederatio Helvetica),
- at Österreich (Austria),
- fi Finnland,
- jp Japan usw.

Daneben finden sich noch einige Exoten wie nato, uucp und bitnet.

Eine **Domain** ist ein Adressbereich[4], der in einem Glied der Adresse übereinstimmt. Alle Adressen der Top-Level-Domain `de` enden auf ebendiese Silbe und bezeichnen Computer, die physikalisch oder logisch in Deutschland beheimatet sind.

Die nächste Domain ist Sache der nationalen Netzverwalter. Hierzulande sorgt das Network Information Center für Deutschland (DE-NIC) – das nationale Standesamt – am Rechenzentrum der Universität Karlsruhe für Ordnung und betreibt den Primary Name Server (ns.nic.de, 193.196.32.1)[5]. Der Universität Karlsuhe ist die Domain `uni-karlsruhe.de` zugewiesen. Sie wird vom Primary Name Server der Universität `netserv.rz.uni-karlsruhe.de` (129.13.64.5) im Rechenzentrum verwaltet, bei dem jeder Computer auf dem Campus anzumelden ist, der

[4]Eine Windows-NT-Domäne ist etwas völlig anderes, nämlich eine Menge von Computern mit gemeinsamer Benutzerverwaltung, unter UNIX einer NIS-Domain entsprechend.

[5]Dieser kennt nicht etwa alle deutschen Knoten, sondern nur die ihm unmittelbar unter- und übergeordneten Name-Server. Es hat also keinen Zweck, ihn als Default-Name-Server auf dem eigenen Knoten einzutragen.

am Netz teilnimmt. Innerhalb der Universität Karlsruhe vergibt das Rechenzentrum die Nummern und Namen, und zwar im wesentlichen die Namen fakultätsweise (`ciw` = Chemieingenieurwesen) und die Nummern gebäudeweise (`118` = Gebäude 30.70), was mit der Verkabelung zusammenhängt. Innerhalb der Fakultäten oder Gebäude geben dann subalterne Manager wie wir die Nummern weiter und erfinden die Namen der einzelnen Computer. In der Regel ist die numerische Adresse mit der Hardware (Netzkarte) verknüpft, die alphanumerische Adresse (Name) mit der Funktion eines Netzcomputers. Unsere beiden Hosts `mvmpc2.ciw.uni-karlsruhe.de` und `ftp.ciw.uni-karlsruhe.de` sind beispielsweise hardwaremäßig identisch, die Namen weisen auf zwei Aufgaben der Kiste hin. Der Benutzer im Netz bemerkt davon kaum etwas; es ist gleich, ob er FTP mit `ftp` oder `mvmpc2` macht. Da die Name-Server für das Funktionieren des Netzes unentbehrlich sind, gibt es außer dem Primary Name Server immer mehrere Secondary Name Server, die die Adresslisten spiegeln und notfalls einspringen. `ns.nic.de` in Karlsruhe wird von Dresden und Stuttgart unterstützt.

Vergibt man Namen, ohne seinen Primary Name Server zu benachrichtigen, so sind diese Namen im Netz unbekannt, die Hosts sind nur über ihre numerische IP-Adresse erreichbar. Verwendet man IP-Adressen oder Namen innerhalb einer Domain mehrfach – was möglich ist, der Name-Server aber nicht akzeptiert – schafft man Ärger.

Auf UNIX-Systemen trägt man in das File `/etc/resolv.conf` die IP-Adressen (nicht die Namen) der Nameserver ein, die man zur Umsetzung von Namen in IP-Adressen heranziehen möchte, zweckmäßig Server in der Nähe. Bei uns steht an erster Stelle ein institutseigener Secondary Name Server, dann der Primary Name Server unserer Universität und an dritter Stelle ein Name Server der Universität Heidelberg.

Welchen Weg die Nachrichten im Netz nehmen, bleibt dem Benutzer verborgen, genau wie bei der Briefpost oder beim Telefonieren. Entscheidend ist, daß vom Absender zum Empfänger eine lückenlose Kette von Computern besteht, die mit Hilfe der Name-Server die Empfänger-Adresse so weit interpretieren können, daß die Nachricht mit jedem Zwischenglied dem Ziel ein Stück näher kommt. Es braucht also nicht jeder Internet-Computer eine Liste aller Internet-Teilnehmer zu halten. Das wäre gar nicht möglich, weil sich die Liste laufend ändert. Mit dem Kommando `traceroute(8)` und einem Hostnamen oder einer IP-Adresse als Argument ermittelt man den gegenwärtigen Weg zu einem Computer im Internet, beispielsweise von meiner Linux-Workstation zu einem Host in Freiburg:

```
/usr/sbin/traceroute ilsebill.biologie.uni-freiburg.de

 1  mv01-eth7.rz.uni-karlsruhe.de (129.13.118.254)
 2  rz11-fddi3.rz.uni-karlsruhe.de (129.13.75.254)
 3  belw-gw-fddi1.rz.uni-karlsruhe.de (129.13.99.254)
 4  Karlsruhe1.BelWue.DE (129.143.59.1)
 5  Freiburg1.BelWue.DE (129.143.1.241)
 6  BelWue-GW.Uni-Freiburg.DE (129.143.56.2)
 7  132.230.222.2 (132.230.222.2)
 8  132.230.130.253 (132.230.130.253)
```

```
 9  ilsebill.biologie.uni-freiburg.de (132.230.36.11)
```

Es geht zwar über erstaunlich viele Zwischenstationen, aber nicht über den Großen Teich. Die Nummer 1 steht bei uns im Gebäude, dann geht es auf den Karlsruher Campus, ins BelWue-Netz und schließlich auf den Freiburger Campus.

3.6 BelWue

BelWue versteht sich als ein Zusammenschluß der baden-württembergischen Hochschulen und Forschungseinrichtungen zur Förderung der nationalen und internationalen Telekooperation und Nutzung entfernt stehender DV-Einrichtungen unter Verwendung schneller Datenkommunikationseinrichtungen. BelWue ist ein organisatorisches Teilnetz im Rahmen des Deutschen Forschungsnetzes. Unbeschadet der innerorganisatorischen Eigenständigkeit der neun Universitätsrechenzentren ist das Kernziel die Darstellung dieser Rechenzentren als eine einheitliche DV-Versorgungseinheit gegenüber den wissenschaftlichen Nutzern und Einrichtungen. Soweit der Minister für Wissenschaft und Kunst von Baden-Württemberg.

Das Karlsruher Campusnetz **KLICK**, an das fast alle Einrichtungen der Universität Karlsruhe angeschlossen sind, ist ein BelWue-Subnetz. BelWue ist – wie oben verkündet – ein Subnetz des Deutschen Forschungsnetzes DFN. Das DFN ist ein Subnetz des Internet. Durch das BelWue-Netz sind miteinander verbunden

- die Universitäten Freiburg, Heidelberg, Hohenheim, Kaiserslautern, Karlsruhe, Konstanz, Mannheim, Stuttgart, Tübingen und Ulm,
- die Fachhochschulen Aalen, Biberach, Esslingen, Furtwangen, Heilbronn, Karlsruhe, Konstanz, Mannheim, Offenburg, Pforzheim, Reutlingen, Stuttgart (3), Ulm und Weingarten (Württemberg),
- die Berufsakademien Karlsruhe, Mannheim, Mosbach, Ravensburg und Stuttgart,
- das Ministerium für Wissenschaft und Forschung, Stuttgart.

Einige Netzadressen sind im Anhang **??** *Netzadressen* zu finden. Weiteres in der Zeitschrift IX Nr. 5/1993, S. 82 - 92.

3.7 Netzdienste im Überblick

Ein Netz stellt Dienstleistungen zur Verfügung. Einige nimmt der Benutzer ausdrücklich und unmittelbar in Anspruch, andere wirken als Heinzelmännchen im Hintergrund. Die wichtigsten sind:

- Terminal-Emulationen (das eigene System wird zum Terminal eines entfernten Systems, man führt einen Dialog) bis hin zu netzorientierten Window-Systemen (X Window System),
- Remote Execution (zum Ausführen von Programmen auf einem entfernten Host, ohne Dialog),

- File-Transfer (zum Kopieren von Files zwischen dem eigenen und einem entfernten System, Dialog eingeschränkt auf die zum Transfer notwendigen Kommandos),
- Electronic Mail (zum Senden und Empfangen von Mail zwischen Systemen),
- Netzgeschwätz (Echtzeit-Dialog mehrerer Benutzer),
- Nachrichtendienste (Neuigkeiten für alle),
- Informationshilfen (Wo finde ich was?),
- Navigationshilfen (Wo finde ich jemand?)
- Netzwerk-File-Systeme,
- Name-Server (Übersetzung von Netz-Adressen),
- Zeit-Server (einheitliche, genaue Zeit im Netz),
- Drucker-Server (Remote Printing, Drucken auf einem entfernten Host),
- Cookie-Server, Backgammon-Server usw. (weniger wichtig).

Das Faszinierende am Netz ist, daß Entfernungen fast keine Rolle spielen. Der Kollege in Honolulu ist manchmal besser zu erreichen als der eigene Chef eine Treppe höher. Die Kosten sind – verglichen mit denen der klassischen Kommunikationsmittel – geringer, und einen Computer braucht man ohnehin. Eine allzu eingehende Beschäftigung mit dem Netz kann allerdings auch – wie übermäßiger Alkoholgenuß – die eigene Leistung gegen Null gehen lassen.

Im Netz hat sich so etwas wie eine eigene Subkultur entwickelt, siehe *The New Hacker's Dictionary* oder das Jargon-File. Die Benutzer des Netzes sehen sich nicht bloß als Teilnehmer an einer technischen Errungenschaft, sondern als Bürger oder Bewohner des Netzes (netizen).

3.8 Terminal-Emulatoren (telnet, rlogin, ssh)

`telnet(1)` emuliert ein VT100-Terminal gemäß dem telnet-Protokoll in TCP/IP-Netzen (Internet). `tn3270(1)` bildet ein VT100-Terminal auf eine IBM-3270-Emulation ab, so daß man mit einem echten oder emulierten VT 100 mit IBM-Großrechnern wie der IBM 3090 einen Dialog führen kann.

Mittels **Remote Login**, Kommando `rlogin(1)`, meldet man sich als Benutzer auf dem entfernten Computer (Host) an. Hat man dort keine Benutzerberechtigung, wird der Zugang verweigert. Darf man, wird eine Sitzung eröffnet, so als ob man vor Ort säße. Ist der lokale Computer ein PC, so muß dieser ein Terminal emulieren, das mit dem Host zusammenarbeitet (oft ein VT 100). Der Unterschied zwischen `telnet(1)` und `rlogin(1)` besteht darin, daß das erstere Kommando ein Internet-Protokoll realisiert und daher auf vielen Systemen verfügbar ist, während die r-Dienstprogramme von Berkeley nur auf UNIX-Systemen laufen.

Das Programmpaar `ssh(1)` (Secure Shell Client) und `sshd(1)` (Secure Shell Daemon) ermöglichen eine Terminalverbindung zu einem entfernten Computer ähnlich wie `telnet(1)` oder `rlogin(1)`. Die Daten gehen jedoch verschlüsselt über

die Leitung und können zwar abgehört, aber kaum von Unberechtigten verwendet werden.

Netzwerkorientierte Window-Systeme ermöglichen es, aufwendige grafische Ein- und Ausgaben über das Netz laufen zu lassen. Ein Beispiel dafür ist das **X Window System**. Näheres siehe Abschnitt 2.6.2 *X Window System*. Innerhalb des X Window Systems lassen sich dann wieder Terminal-Emulatoren starten – auch mehrere gleichzeitig – so daß man auf einem Bildschirm verschiedene Terminal-Sitzungen mit beliebigen X-Window-Clienten im Netz abhalten kann.

In größeren Anlagen sind die Terminals nicht mehr unmittelbar mit dem Computer verbunden, weil auch vorübergehend nicht benutzte Terminals einen wertvollen Port belegen würden. Sie sind vielmehr mit einem **Terminal-Server** verbunden, der nur die aktiven Terminals zum Computer durchschaltet. Der Terminal-Server ist ein kleiner Computer, der nur ein Protokoll wie Telnet fährt. Der Terminal-Server kann an mehrere Computer angeschlossen sein, so daß jedes Terminal gleichzeitig mehrere Sitzungen auf verschiedenen Anlagen geöffnet haben kann. Wenn ein Benutzer dann einen **Session Manager** zur Verwaltung seiner offenen Sitzungen braucht, ist er auf der Höhe der Zeit. Terminal in Karlsruhe, Daten in Stuttgart, Prozesse in Bologna und Druckerausgabe in Fort Laramy, alles möglich!

3.9 File-Transfer (kermit, ftp, fsp)

Um im vorigen Beispiel zu bleiben, nehmen wir an, daß unser PC ein Terminal emuliert und wir eine Sitzung auf dem entfernten Computer (Host) eröffnet haben. Jetzt möchten wir ein File von dem Host auf unseren PC übertragen, eine Aufgabe, die zwischen einem echten Terminal und einem Computer keinen Sinn macht, weil das echte Terminal keinen Speicher hat, in das ein File kopiert werden könnte. Dasselbe gilt auch für die umgekehrte Richtung. Wir brauchen also neben der Emulation ein Programm für die File-Übertragung. Im einfachsten Fall sind das Kopierprogramme ähnlich `cat(1)` oder `cp(1)`, die zum Computerausgang schreiben bzw. vom Computereingang (serielle Schnittstellen) lesen, und zwar muß auf dem sendenden und auf dem empfangenden Computer je eines laufen.

Bei der Übertragung treten Fehler auf, die unangenehmer sind als ein falsches Zeichen auf dem Bildschirm, außerdem spielt die Geschwindigkeit eine Rolle. Man bevorzugt daher gesicherte **Übertragungsprotokolle**, die die zu übertragenden Daten in Pakete packen und jedes Paket mit einer Prüfsumme versehen, so daß der Empfänger einen Übertragungsfehler mit hoher Wahrscheinlichkeit bemerkt und eine Wiederholung des Paketes verlangt. Beispiele gesicherter Protokolle sind `kermit`, `xmodem`, und `zmodem`. Sie gehören *nicht* zu den Internet-Protokollen. Wir verwenden oft `kermit(1)`. Es ist zwar angejahrt, aber verbreitet, Original Point of Distribution `kermit.columbia.edu`. Das für viele Systeme verfügbare kermit-Programm enthält auch eine Terminal-Emulation, erledigt also zwei Aufgaben.

Bei einem File Transfer mittels `ftp(1)` kopiert man ein File von einem Computer zum anderen und arbeitet dann mit seiner lokalen Kopie weiter. **FTP** geht in beide Richtungen, senden und empfangen. Es ist ein Internet-Protokoll und wird

im RFC 959 beschrieben. Unter FTP stehen mehrere Dutzend FTP-Kommandos zur Verfügung, die beim File-Transfer gebraucht werden. Man kann also nicht wie beim Remote Login auf der entfernten Maschine arbeiten, die Eingabe von UNIX-Kommandos führt zu einem Fehler. Einige FTP-Kommandos haben dieselben Namen wie DOS- oder UNIX-Kommandos, aber nicht alle. Ein Trick, um sich kleine Textfiles (`readme`) doch gleichsam on-line anzuschauen:

```
get readme |more
```

Das FTP-Kommando `get` erwartet als zweites Argument den lokalen Filenamen. Beginnt dieser mit dem senkrechten Strich einer Pipe, unmittelbar gefolgt von einem UNIX-Kommando, so wird das übertragene File an das UNIX-Kommando weitergereicht. Eine andere Möglichkeit ist, das File zu übertragen, FTP mittels eines Ausrufezeichens vorübergehend zu verlassen, auf Shellebene mit dem File zu arbeiten und nach Beenden der Shell FTP fortzusetzen. Beide Verfahren belegen zwar keine Übertragungswege (da paketvermittelt), aber auf den beteiligten Computern einen FTP-Port, und deren Anzahl ist begrenzt.

Bei der Übertragung zwischen ungleichen Systemen (UNIX – MS-DOS – Macintosh) ist zwischen Textfiles und binären Files zu unterscheiden. Textfiles unterscheiden sich – wir sprachen in Abschnitt 2.7.11 *Textfiles aus anderen Welten* darüber – in der Gestaltung des Zeilenwechsels. Die Übertragungsprogramme übersetzen stillschweigend den Zeilenwechsel in die Zeichenkombination des jeweiligen Zielcomputers. Alle anderen Files gelten als binär und sind zu übertragen, ohne auch nur ein Bit zu ändern. Bei der Übertragung zwischen zwei UNIX-Systemen braucht man den Unterschied nicht zu beachten. Auch Postscript-Files und gepackte Textfiles müssen binär übertragen werden. Überträgt man ein Textfile binär, kann man mit einem einfachen Filter den Zeilenwechsel wieder hinbiegen. Ist ein Binärfile im Textmodus von FTP übertragen worden, ist es Schrott.

Arbeitet man hinter einer Firewall-Maschine, so kann der FTP-Dialog zwischen Client und Server mißlingen. Normalerweise verlangt nach Beginn des Dialogs der Server vom Client die Eröffnung eines Kanals zur Datenübertragung. Die Firewall sieht in dem Verlangen einen hereinkommenden Aufruf an einen unbekannten Port, eine verdächtige und daher abzublockende Angelegenheit. Schickt man nach Herstellung der Verbindung, jedoch vor der Übertragung von Daten das FTP-Kommando `pasv` an den Server, so wird die Datenverbindung vom Client aus aufgebaut, und die Firewall ist beruhigt. Nicht alle FTP-Server unterstützen jedoch dieses Vorgehen.

Das **File Service Protocol FSP** dient dem gleichen Zweck wie FTP, ist etwas langsamer, aber dafür unempfindlich gegenüber Unterbrechungen. Manche Server bieten sowohl FTP wie auch FSP an.

3.10 Anonymous-FTP

In Universitäten ist es Brauch, den Netzteilnehmern Informationen und Software unentgeltlich zur Verfügung zu stellen. Was mit öffentlichen Mitteln finanziert worden ist, soll auch der Öffentlichkeit zugute kommen. Einige Organisationen

und Firmen haben sich ebenfalls dem Netzdienst angeschlossen. Zu diesem Zweck wird auf den Anlagen ein Benutzer namens `anonymous` (unter vielen Systemen auch `ftp`) eingerichtet, der wie `gast` kein Passwort benötigt. Es ist jedoch üblich, seine Email-Anschrift als Passwort mitzuteilen. Nach erfolgreicher Anmeldung auf einer solchen Anlage kann man sich mit einigen FTP-Kommandos in den öffentlichen Verzeichnissen (oft `/pub`) umsehen und Files auf die eigene Anlage kopieren (download). Eine Anonymous-FTP-Verbindung mit der Universität Freiburg im schönen Breisgau verläuft beispielsweise so:

```
ftp ftp.uni-freiburg.de
anonymous                          (Login-Name)
ig03@mvmhp.ciw.uni-karlsruhe.de    (eigener Name als Passwort)
dir                                (wie UNIX-ls)
ascii                              (Textmodus)
get README                         (File README holen)
cd misc
dir
quit
```

Anschließend findet man das Freiburger File README in seinem Arbeits-Verzeichnis. Die Geschwindigkeit der Verbindung liegt bei 600 Bytes/s. Allerdings ist diese Angabe infolge der geringen Filegröße ungenau. Auf diese Weise haben wir uns den *Hitchhikers Guide to Internet* besorgt.

Die Verbindung funktioniert nicht nur im Ländle[6], sondern sogar bis zum anderen Ende der Welt. Mit

```
ftp ftp.cc.monash.edu.au
wulf.alex@ciw.uni-karlsruhe.de
dir
cd pub
dir
quit
```

schaut man sich im Computer Center der Monash University in Melbourne in Australien um. Die Geschwindigkeit sinkt auf 40 Bytes/s. Man wird sich also nicht megabytegroße Dokumente von dort holen. Grundsätzlich soll man immer zuerst in der Nachbarschaft suchen. Viele Files werden nämlich nicht nur auf ihrem Ursprungscomputer (original point of distribution, OPD) verfügbar gehalten, sondern auch auf weiteren Hosts. Manche FTP-Server kopieren sogar ganze Verzeichnisbäume fremder Server. Eine solche Kopie wird **Spiegel** (mirror) genannt. Ein Spiegel senkt die Kosten und erhöht die Geschwindigkeit der Übertragung. Weiterhin gebietet der Anstand, fremde Computer nicht zu den dortigen Hauptverkehrszeiten zu belästigen.

Da der Mensch seit altersher mit einem starken Sammeltrieb ausgestattet ist, stellt Anonymous-FTP für den Anfänger eine Gefahr dar. Zwei Hinweise. Erstens:

[6] Für Nicht-Badener: Das Ländle ist Baden, seine Einwohner heißen Badener und nicht etwa Badenser. Die Frankfurter nennen sich ja auch nicht Frankfurtser.

Man lege ein Verzeichnis `aftp` an (der Name `ftp` wird meist für die FTP-Software benötigt). In diesem richte man für jeden FTP-Server, den man anzapft, ein Unterverzeichnis an. In jedem Unterverzeichnis schreibe man ein Shellscript namens `aftp` mit folgender Zeile:

```
ftp ftp-servername
```

`ftp-servername` ist der Name, notfalls die numerische Internet-Adresse des jeweiligen FTP-Servers. Das Shellscript mache man les- und ausführbar (750). Dann erreicht man in dem augenblicklichen Verzeichnis mit dem Kommando `aftp` immer den zugehörigen Server und weiß, woher die Files stammen. Weiter lege man für wichtige Programme, deren Herkunft man bald vergessen hat, in dem Verzeichnis `aftp` einen Link auf das zum FTP-Server gehörige Unterverzeichnis an. So hat man einen doppelten Zugangsweg: über die Herkunft und den Namen. Bei uns schaut das dann so aus:

```
...
emacs -> unimainz
unimainz
    aftp
    ...
    emacs-20.2
    emacs-20.2.tar.gz
...
```

Sie dürfen sich gern ein anderes Ordnungsschema ausdenken, aber ohne Ordnung stehen Sie nach vier Wochen Anonymous-FTP im Wald.

Zweitens: Man hole sich nicht mehr Files in seinen Massenspeicher, als man in nächster Zukunft verarbeiten kann, andernfalls legt man nur eine Datengruft zum Wohle der Plattenindustrie an. Für alle weiteren Schätze reicht eine Notiz mit Herkunft, Namen, Datum und Zweck. Files ändern sich schnell. Mit überlagerten Daten zu arbeiten ist Zeitvergeudung.

Das **Gutenberg-Projekt** hat sich zur Aufgabe gesetzt, bis zum Jahr 2001 eine Vielzahl englischsprachiger Texte als ASCII-Files zur Verfügung zu stellen. Die Bibel, William Shakespeare's Gesammelte Werke und die Verfassung der USA gibt es schon. Versuchen Sie FTP mit `mrcnext.cso.uiuc.edu` in der University of Illinois. Unser Exemplar der Zahl π auf eine Million Stellen (`ftp.ciw: /pub/misc/pi/pimil10.txt`) stammt von dort, beeindruckend.

Die kostenfreie **GNU-Software** kommt von `prep.ai.mit.edu` in den USA, kann aber auch von mehreren Servern (mirrors) in Europa abgeholt werden. Inzwischen gibt es auch eine Liste der deutschen Mirrors. Die SIMTEL-Archive werden von `ftp.uni-paderborn.de` gespiegelt. Man muß fragen und suchen, das Internet kennt keine zentrale Verwaltung.

Die Einrichtung eines eigenen **FTP-Servers** unter UNIX ist nicht weiter schwierig, siehe die man-Seite zu `ftpd(1M)`. Man muß achtgeben, daß anonyme Benutzer nicht aus dem ihnen zugewiesenen Bereich im Filesystem herauskönnen, sofern man überhaupt Anonymous FTP zulassen will. Bei uns greifen FTP-Server und WWW-Server auf denselben Datenbestand zu, das hat sich als zweckmäßig

erwiesen. Andersherum: unser FTP-Server kann inzwischen auch WWW, unser WWW-Server kann FTP, und der Datenbestand des einen ist eine Kopie des anderen, ein Backup.

3.11 Electronic Mail (Email)

3.11.1 Grundbegriffe

Electronic Mail, Email oder Computer-Mail ist die Möglichkeit, mit Benutzern im Netz zeitversetzt Nachrichten auszutauschen, in erster Linie kurze Texte. Die Nachrichten werden in der Mailbox[7] des Empfängers gespeichert, wo sie bei Bedarf abgeholt werden. Die **Mailbox** ist ein File oder ein Unterverzeichnis auf dem Computer des Empfängers.

PCs unter MS-DOS und ähnliche Rechner haben hier eine Schwierigkeit. Sie sind oftmals ausgeschaltet oder mit anderen Arbeiten beschäftigt, jedenfalls nicht bereit, Mail entgegenzunehmen. Eine größere UNIX-Anlage dagegen ist ständig in Betrieb und vermag als Multitasking-System Post zu empfangen, während sie andere Aufgaben bearbeitet. Die Lösung ist, die Nachrichten auf zentralen **Mailservern** zu speichern und von dort – möglichst automatisch – abzuholen, sobald der eigene Computer bereit ist. Das Zwischenlager wird manchmal als **Maildrop** bezeichnet. Hierzu wird das **Post Office Protocol** (POP) nach RFC 1460 verwendet; der POP-Dämon ist in `/etc/services` und in `/etc/inetd.conf` einzutragen, ist also ein Knecht des inet-Dämons. Auf dem PC oder Mac läuft ein POP-fähiges Mailprogramm wie *Eudora*, das bei Aufruf Kontakt zum Mailserver aufnimmt.

Im Internet wird der Mailverkehr durch das **Simple Mail Transfer Protocol** SMTP nach RFC 821 in Verbindung mit RFC 822 geregelt. Eine Alternative ist die CCITT-Empfehlung X.400, international genormt als ISO 10021. Es gibt Übergänge zwischen den beiden Protokollwelten, Einzelheiten siehe im RFC 1327. Im Internet wird Mail sofort befördert und nicht zwischengelagert wie in einigen anderen Netzen (UUCP). Die Adresse des Empfängers muß hundertprozentig stimmen, sonst kommt die Mail als unzustellbar zurück. Eine gültige Benutzeradresse ist:

```
wualex1@mvmhp64.ciw.uni-karlsruhe.de
```

`wualex1` ist ein Benutzername, wie er im File `/etc/passwd(4)` steht. Der Kringel – das ASCII-Zeichen Nr. 64 – wird im Deutschen auch Klammeraffe (commercial at, arobace) genannt und trennt den Benutzernamen vom Computernamen. Falls man Schwierigkeiten beim Eingeben dieses Zeichens hat, kann man es mit `\@` oder `control-v @` versuchen. Der Klammeraffe dient gelegentlich als Steuerzeichen und löscht dann eine Zeile. `mvmhp64` ist der Name des Computers, `ciw` die Subdomain (Fakultät für Chemieingenieurwesen), `uni-karlsruhe` die Domain und `de` die

[7] Das Wort *Mailbox* wird in anderen Netzen auch als Oberbegriff für ein System aus Postfächern und Anschlagtafeln gebraucht, siehe die Liste der Mailboxen in der Zeitschrift c't oder in der Newsgruppe `de.etc.lists`.

Top-level-domain Deutschland. Andere Netze (Bitnet, UUCP) verwenden andere Adressformate, was zur Komplexität von Mailprogrammen wie `sendmail(1M)` und deren Konfiguration beiträgt.

Besagter Benutzer tritt auch noch unter anderen Namen auf anderen Maschinen auf. In den jeweiligen Mailboxen oder Home-Verzeichnissen steht ein `forward`-Kommando, das etwaige Mail an obige Adresse weiterschickt. Keine Mailbox zu haben, ist schlimm, viele zu haben, erleichtert das Leben auch nicht gerade. Da man auf jedem Computer, der ans Netz angeschlossen ist, grundsätzlich eine Mailbox (das heißt eine gültige Mailanschrift) besitzt, hat man selbst für das richtige Forwarding sorgen. Andernfalls kann man jeden Morgen die Menge seiner Mailboxen abklappern. Man muß aber aufpassen, daß man keine geschlossenen Wege erzeugt: Von Computer A nach Computer B, von diesem nach C und von C wieder nach A. Eine Mail gelangt zwar in diesen Ring hinein, kreist dann aber in der Schleife, bis ein Postmaster eingreift.

Da Computer kommen und gehen und mit ihnen ihre Namen, ist es unpraktisch, bei jedem Umzug aller Welt die Änderung der Mailanschrift mitteilen zu müssen. Unser Rechenzentrum hat daher **generische Anschriften** gemäß der CCITT-Empfehlung X.500 eingeführt, die keinen Maschinennamen mehr enthalten:

```
wulf.alex@ciw.uni-karlsruhe.de
```

Ein Server im Rechenzentrum weiß, daß Mail an diese Anschrift zur Zeit an `wualex1@mvmhp64` weitergeleitet werden soll. Bei einem Umzug genügt eine Mitteilung ans Rechenzentrum, für die Außenwelt ändert sich nichts. Die Anschriften mit Maschinennamen bleiben weiterhin bestehen, sollten aber nicht veröffentlicht werden. Im Prinzip könnte eine einmal angelegte X.500-Anschrift lebenslang gültig bleiben, da sie die etwaigen Änderungen der tatsächlichen Email-Anschrift verbirgt.

Die CCITT-Empfehlung **X.500** hat zunächst nichts mit Email zu tun, sondern ist ein weltweites, verteiltes Informationssystem mit Informationen über Länder, Organisationen, Personen usw. Zu jedem Objekt gehören bestimmte Attribute, zu einer Person unter anderem Name, Telefonnummer und Email-Anschriften. Das sind personenbezogene Daten, die unter die Datenschutzgesetze fallen. Die Eintragung der Daten bedarf daher der Zustimmung des Betroffenen. Wer sich nicht eintragen lassen will, ist unter Umständen schwierig zu finden.

Kennt man den Benutzernamen nicht, aber wenigstens den vollständigen Computernnamen, kann man die Mail mit der Bitte um Weitergabe an `postmaster@computername` schicken. Die Postmaster sind Kummer gewöhnt. Jeder Mailserver soll einen haben.

Die Mailprogramme fügen der Mail eine Anzahl von **Kopfzeilen** (Header) hinzu, die folgendes bedeuten (RFC 822, RFC 2045):

- Message-ID: weltweit eindeutige, maschinenlesbare Bezeichnung der Mail
- Date: Zeitpunkt des Absendens
- From: logischer Absender
- Sender: tatsächlicher Absender

- Return-Path: Rückweg zum Absender
- Reply-to: Anschrift für Antworten
- Organization: Organisation des Absenders, z. B. Universität Karlsruhe
- To: Empfänger
- Cc: Zweiter Empfänger (Carbon copy)
- Received: Einträge der Hosts, über die Mail ging
- Subject: Thema der Mail
- Keywords: Schlagwörter zum Inhalt der Mail
- Lines: Anzahl der Zeilen ohne Header
- Precedence: Dringlichkeit wie urgent, normal, bulk
- Priority: Dringlichkeit wie urgent, normal, bulk
- Status: z. B. bereits gelesen, wird vom MDA (`elm(1)`) eingesetzt
- In-Reply-To: Bezug auf eine Mail (Message-ID) des Empfängers
- References: Bezüge auf andere Mails (Message-IDs)
- Resent: weitergleitet
- Expires: Haltbarkeitsdatum der Mail (best before ...)
- Errors-To: Anschrift für Probleme
- Comments: Kommentar
- MIME-Version: MIME-Version, nach der sich die Mail richtet
- Content-Transfer-Encoding: MIME-Codierungsverfahren, Default 7bit
- Content-ID: MIME ID der Mail, weltweit eindeutig
- Content-Description: MIME Beschreibung des Inhalts der Mail
- Content-Type: MIME text, image, audio, video, application usw. Defaultwert text/plain; charset=us-ascii
- Content-Length: MIME Anzahl der Zeichen, ohne Header
- X400-Originator u. a.: Felder nach CCITT-Empfehlung X.400/ISO 10021
- X-Sender: (user defined field)
- X-Mailer: (user defined field)
- X-Gateway: (user defined field)
- X-Priority: (user defined field)
- X-Envelope-To: (user defined field)
- X-UIDL: (user defined field)

Die meisten Mails weisen nur einen Teil dieser Header-Zeilen auf, abhängig vom jeweiligen Mailprogramm. Einige der Header-Zeilen wie *Subject* lassen sich editieren.

Zum Feld `Content-Transfer-Encoding` nach RFC 2045 noch eine Erläuterung. Das Simple Mail Transfer Protocol läßt nur 7-bit-Zeichen und Zeilen mit weniger als 1000 Zeichen zu. Texte mit Sonderzeichen oder binäre Daten müssen daher umcodiert werden, um diesen Forderungen zu genügen. Das Feld weist das die Mail an den Empfänger ausliefernde Programm darauf hin, mit welchem Zeichensatz bzw. welcher Codierung (nicht: Verschlüsselung) die Daten wiederzugeben sind. Übliche Eintragungen sind:

- 7bit (Default, 7-bit-Zeichensatz, keine Codierung),
- 8bit (8-bit-Zeichensatz, keine Codierung),
- binary (binäre Daten, keine Codierung),
- quoted-printable (Oktetts werden in die Form =Hexpärchen codiert, druckbare 7-bit-US-ASCII-Zeichen dürfen beibehalten werden),
- base64 (jeweils 3 Zeichen = 3 Bytes = 24 Bits werden codiert in 4 Zeichen des 7-bit-US-ASCII-Zeichensatzes, dargestellt durch 4 Bytes mit höchstwertigem Bit gleich null),
- ietf-token (Sonderzeichen der IETF/IANA),
- x-token (user defined).

Dieses Feld sagt nichts darüber aus, ob die Daten Text, Bilder, Audio oder Video sind. Das gehört in das Feld Content-Type. Beispiel:

```
Content-Type: text/plain; charset=ISO-8859-1
Content-transfer-endocding: base64
```

kennzeichnet eine Mail als einen Text, urpsrünglich geschrieben mit dem Zeichensatz ISO 8859-1 (Latin-1) und codiert gemäß base64 in Daten, die nur Zeichen des 7-bit-US-ASCII-Zeichensatzes enthalten. Nach Rückcodierung hat man den Text und kann ihn mit einem Ausgabegerät, das den Latin-1-Zeichensatz beherrscht, in voller Pracht genießen.

Nun die entscheidende Frage: Wie kommt eine Mail aus meinem Rechner an einen Empfänger irgendwo in den unendlichen Weiten? Im Grunde ist es ähnlich wie bei der Briefpost. Alle Post, die ich nicht in meinem Heimatdorf selbst austrage, werfe ich in meinen Default-Briefkasten ein. Der Rest ist Sache der Deutschen Post AG. Vermutlich landet mein Brief zuerst in Karlsruhe auf einem Postamt. Da er nach Fatmomakke in Schweden adressiert ist, dieser Ort jedoch in Karlsruhe ziemlich ausländisch klingt, gelangt der Brief zu einer für das Ausland zuständigen zentralen Stelle in Frankfurt (Main) oder Hamburg. Dort ist zumindest Schweden ein Begriff, der Brief fliegt weiter nach Stockholm. Die stockholmer Postbediensteten wollen mit Fatmomakke auch nichts zu tun haben und sagen bloß *Ab damit nach Östersund*. Dort weiß ein Busfahrer, daß Fatmomakke über Vilhelmina zu erreichen ist und nimmt den Brief mit. Schließlich fühlt sich der Landbriefträger in Vilhelmina zuständig und händigt den Brief aus. Der Brief wandert also durch

eine Kette von Stationen, die jeweils nur ihre Nachbarn kennen, im wesentlichen in die richtige Richtung.

Genau so läuft die elektronische Post. Schauen wir uns ein Beispiel an. Die fiktive Anschrift sei `xy@access.owl.de`, der Rechner ist echt, jedoch kein Knoten (Host) im Internet. Das Kommando `nslookup(1)` sagt *No Address*. Mit `host -a access.owl.de` (unter LINUX verfügbar) erfahren wir etwas mehr, nämlich (gekürzt):

```
access.owl.de 86400 IN MX (pri=20) by pax.gt.owl.de
access.owl.de 86400 IN MX (pri=50) by jengate.thur.de
access.owl.de 86400 IN MX (pri=100) by ki1.chemie.fu-berlin.de
access.owl.de 86400 IN MX (pri=10) by golden-gate.owl.de
For authoritative answers, see:
owl.de 86400 IN NS golden-gate.owl.de
Additional information:
golden-gate.owl.de 86400 IN A 131.234.134.30
golden-gate.owl.de 86400 IN A 193.174.12.241
```

Der Rechner `access.owl.de` hat keine Internet-Adresse, es gibt aber vier Internet-Rechner (MX = Mail Exchange), die Mail für `access.owl.de` annehmen. Der beste (pri=10) ist `golden-gate.owl.de`. Dessen IP-Adresse erfährt man mit `nslookup(1)` oder `host(1)`, sofern von Interesse. Wie die Mail von Karlsruhe nach `golden-gate.owl.de` gelangt, ermittelt das Kommando `traceroute golden-gate.owl.de`:

```
 1  mv01-eth7.rz.uni-karlsruhe.de (129.13.118.254)  1.230 ms
 2  rz11-fddi3.rz.uni-karlsruhe.de (129.13.75.254)  2.238 ms
 3  belw-gw-fddi1.rz.uni-karlsruhe.de (129.13.99.254)  4.397 ms
 4  Karlsruhe1.BelWue.DE (129.143.59.1)  2.821 ms
 5  Uni-Karlsruhe1.WiN-IP.DFN.DE (188.1.5.29)  2.682 ms
 6  ZR-Karlsruhe1.WiN-IP.DFN.DE (188.1.5.25)  3.967 ms
 7  ZR-Frankfurt1.WiN-IP.DFN.DE (188.1.144.37)  14.330 ms
 8  ZR-Koeln1.WiN-IP.DFN.DE (188.1.144.33)  19.910 ms
 9  ZR-Hannover1.WiN-IP.DFN.DE (188.1.144.25)  24.667 ms
10  Uni-Paderborn1.WiN-IP.DFN.DE (188.1.4.18)  26.569 ms
11  cisco.Uni-Paderborn.DE (188.1.4.22)  25.23 ms  26.126 ms
12  fb10sj1-fb.uni-paderborn.de (131.234.250.37)  28.262 ms
13  golden-gate.uni-paderborn.de (131.234.134.30)  29.128 ms
```

Station 1 ist das Gateway, das unser Gebäudenetz mit dem Campusnetz verbindet. Mit der Station 5 erreichen wir das deutsche Wissenschaftsnetz, betrieben vom *Verein zur Förderung eines Deutschen Forschungsnetzes (DFN-Verein)*. Diese noch zur Universität Karlsruhe gehörende Station schickt alles, was sie nicht selbst zustellen kann, an ein Default-Gateway in Karlsruhe (Nr. 6). Von dort geht es über Frankfurt, Köln und Hannover (wo der Router offenbar einmal etwas von Paderborn und `owl.de` gehört hat) in die Universität Paderborn, der Heimat des Rechners `golden-gate.owl.de`. Dieser Weg braucht weder physikalisch noch logisch der schnellste zu sein, Hauptsache, er führt mit Sicherheit zum Ziel.

Die Software zum netzweiten Mailen auf einem UNIX-Rechner setzt sich aus zwei Programmen zusammen: einem Internet-Dämon (Mail Transport Agent), oft `sendmail(1M)`, und einem benutzerseitigen Werkzeug (Mail Delivery Agent) wie `elm(1)`. Der Benutzer kann zwar auch mit `sendmail(1M)` unmittelbar verkehren, aber das ist abschreckend und nur zur Analyse von Störfällen sinnvoll. Das Werkzeug `elm(1)` arbeitet mit einfachen Menus und läßt sich den Benutzerwünschen anpassen. Es ist komfortabler als `mail(1)` und textorientiert (ohne MUFF).

Will man zwecks Störungssuche auf der eigenen Maschine unmittelbar mit `sendmail(1)` eine Mail verschicken, geht man so vor:

```
sendmail -v empfaengeradressen
Dies ist eine Testmail.
Gruss vom Mostpaster.
.
```

Der einzelne Punkt beendet die Mail samt Kommando. Auf eine entfernte Maschine greift man per `telnet(1)` zu. Auf Port 25 liegt der Maildämon:

```
telnet entfernte_maschine 25
help
mail from: Mostpaster
rcpt to: empfaengeradresse1
rcpt to: empfaengeradresse2
data
Dies ist eine Testmail.
Gruss vom Mostpaster
.
quit
```

Im Logfile von `sendmail(1M)` erzeugt jedes *from* und jedes *to* eine Eintragung. Zusammengehörige Ein- und Auslieferungen haben diesselbe Nummer. Mit den üblichen Werkzeugen zur Textverarbeitung läßt sich das Logfile auswerten. Diese Wege sind – wie gesagt – nicht für die Alltagspost gedacht.

Die Electronic Mail im Internet ist zum Versenden von Nachrichten gedacht, nicht zum weltweiten Ausstreuen unerbetener Werbung. Diese wird als **Spam** bezeichnet, was auf eine Geschichte zurückgeht, in der *Spiced Ham* eine Rolle spielt. Die Spam-Flut ist momentan ein Problem, da Technik und Gesetzgeber nicht auf diesen Mißbrauch des Netzes vorbereitet sind. Es tut sich aber schon etwas.

Mail dient in erster Linie zum Verschicken von Texten, die – sofern man sicher gehen will – nur die Zeichen des 7-Bit-US-ASCII-Zeichensatzes enthalten dürfen. Will man beliebige binäre Files per Mail verschicken (FTP wäre der bessere Weg), muß man die binären Files umcodieren und beim Empfänger wieder decodieren. Damit lassen sich beliebige Sonderzeichen, Grafiken oder ausführbare Programme mailen. Ein altes Programmpaar für diesen Zweck ist `uuencode(1)` und `uudecode(1)`. Neueren Datums sind `mpack(1)` und `munpack(1)`, die von den *Multipurpose Internet Mail Extensions* (MIME) Gebrauch machen.

Die Mailbenutzer haben einen eigenen Jargon entwickelt. Einige Kürzel finden Sie im Anhang H *Slang im Netz* und im Netz in Files namens `jargon.*` oder ähnlich. Daneben gibt es noch die **Grinslinge** oder **Smileys**, die aus ASCII-Zeichen bestehen und das über das Netz nicht übertragbare Mienenspiel bei einem Gespräch ersetzen sollen. Die meisten sind von der Seite her zu lesen:

- :-) Grinsen, Lachen, bitte nicht ernst nehmen
- :-(Ablehnung, Unlust, Trauer
- %*@:-(Kopfweh, Kater
- :-x Schweigen, Kuß
- :-o Erstaunen
- `+|+-)` schlafend, langweilig
- Q(8-{)## Mann mit Doktorhut, Glatze, Brille, Nase, Bart, Mund, Fortsetzung des Bartes (die Ähnlichkeit mit einem der Verfasser ist verblüffend)

3.11.2 Mailing-Listen

Wozu lassen sich Mailing-Listen (Verteilerlisten) gebrauchen? Zwei Beispiele. In der Humboldt-Universität zu Berlin wird eine Mailing-Liste `www-schulen` geführt. Schüler P. hat diese Liste abonniert (subskribiert), weil er sich für das Medium WWW interessiert und wissen möchte, was sich auf diesem Gebiet in den Schulen so tut. Er hat eine Frage zur Teilbarkeit von Zahlen und schickt sie per Email an die Liste, das heißt an die ihm weitgehend unbekannte Menge der Abonnenten. Die Liste paßt von ihrer Ausrichtung her zwar nicht optimal, ist aber auch nicht gänzlich verfehlt, immerhin haben Zahlen und Schule etwas gemeinsam. Seine Mail wird an alle Mitglieder oder Abonnenten der Liste verteilt. Der ehemalige Schüler T. hat aus beruflichen Gründen die Liste ebenfalls abonniert und noch nicht alles vergessen, was er einst gelernt. Er liest die Mail in der Liste und antwortet an die Liste. Familienvater W. nimmt auch an der Liste teil, findet die Antwort gut, druckt sie aus und legt sie daheim in ein Buch über Zahlentheorie. Schüler P. ist geholfen, Familienvater W. hat etwas gelernt, und der Ehemalige T. freut sich, ein gutes Werk getan zu haben. Aufwand vernachlässigbar, auf herkömmlichen Wegen untunlich.

Zweites Beispiel. Wenn man früher eine Frage zu einer Vorlesung hatte, konnte man den Dozenten gleich nach der Vorlesung oder in seiner Sprechstunde löchern, sofern man Glück hatte. Die Fragen tauchen jedoch meist nachts im stillen Kämmerlein auf, außerdem ist nicht immer der Dozent der geeignetste Ansprechpartner. Heute richtet man zu einer Vorlesung eine lokale Mailing-Liste ein, jeder kann jederzeit schreiben, und der Kreis der potentiellen Beantworter ist weitaus größer. So gibt es zu den beiden Vorlesungen, aus denen dieses Buch entstanden ist, die Liste `wualex-l@rz.uni-karlsruhe.de`, die einzige Möglichkeit, den aus mehreren Fakultäten stammenden Hörerkreis schnell zu erreichen. Umgekehrt erhalten auch die Hörer Antwort, sowie ihr Anliegen bearbeitet ist

und nicht erst in der nächsten Vorlesung. Das Ganze funktioniert natürlich auch in der vorlesungsfreien Zeit, den sogenannten Semesterferien.

Eine Mailing-Liste ist also ein Verteiler, der eine einkommende Mail an alle Mitglieder verteilt, die wiederum die Möglichkeit haben, an die Liste oder individuell zu antworten. Von der Aufgabe her besteht eine leichte Überschneidung mit den Netnews, allerdings sind die Zielgruppen kleiner, und der ganze Verkehr ist besser zu steuern. Beim Arbeiten mit Mailing-Listen sind das Listenverwaltungsprogramm und die Liste selbst zu unterscheiden. Wünsche betreffs Subskribieren, Kündigen und Auskünften *über* die Liste gehen per Email an das Verwaltungsprogramm, Mitteilungen, Fragen und Antworten an die Liste. Wollen Sie unsere Liste subskribieren, schicken Sie eine Email mit der Zeile:

```
subscribe wualex-l Otto Normaluser
```

und weiter nichts im Text (body) an `listserv@rz.uni-karlsruhe.de` und setzen anstelle von `Otto Normaluser` Ihren bürgerlichen Namen ein. Der Listserver schickt Ihnen dann eine Bestätigung. Anschließend können Sie Ihre erste Mail an die Liste schicken. Sie schreiben an die Liste, den virtuellen Benutzer `wualex-l@rz.uni-karlsruhe.de` und fragen, ob UNIX oder Windows-NT das bessere Betriebssystem sei. Bekannte Listenverwaltungsprogramme sind `listserv`, `listproc` und für kleinere Anlagen (lokale Listen) `majordomo`; sie unterscheiden sich für den Benutzer geringfügig in ihrer Syntax.

Die Listenverwaltung `majordomo` stammt aus der UNIX-Welt und ist frei. Sie besteht aus einer Reihe von `perl`-Skripts, einigen Alias-Zeilen für den Mail-Dämon `sendmail` und mehreren Files mit der Konfiguration und den Email-Anschriften der Abonnenten. Das Einrichten von `majordomo` samt erster Liste hat uns etwa einen Tag gekostet – die Beschreibung war älter als das Programm – das Einrichten weiterer Listen je eine knappe Stunde. In unserem Institut setzen wir die Listen für Rundschreiben ein.

Es gibt offene Listen, die jedermann subskribieren kann, und geschlossene, deren Zugang über einen Listen-Manager (List-Owner) führt. Ferner können Listen moderiert sein, so daß jede Einsendung vor ihrem Weiterversand über den Bildschirm des Moderators geht. Im Netz finden sich Verzeichnisse von Mailing-Listen und Suchprogramme, siehe Anhang. Sie können es auch mit einer Mail `lists global` (und weiter nichts) an `listserv@rz.uni-karlsruhe.de` versuchen. Oder erstmal mit folgenden Zeilen:

```
help
lists
end
```

an `major@domo.rrz.uni-hamburg.de`. Die Anzahl der Listen weltweit wird auf einige Zehntausend geschätzt.

3.11.3 Privat und authentisch (PGP, PEM)

Warum sollte man das Programmpaket **Pretty Good Privacy (PGP)** oder das Protokoll **Privacy Enhanced Mail (PEM)** verwenden? Zum einen besteht in

vielen Fällen die Notwendigkeit einer Authentisierung des Urhebers einer Nachricht, zum Beispiel bei Bestellungen. Zum anderen sollte man sich darüber im klaren sein, daß eine unverschlüsselte E-Mail mit einer Postkarte vergleichbar ist: Nicht nur die Postmaster der am Versand beteiligten Systeme können den Inhalt der Nachricht einsehen, sondern auch Bösewichte, die die Nachricht auf ihrem Weg durchs Netz kopieren. Auch Verfälschungen sind machbar, und schließlich könnte der Urheber einer Mail bei bestimmten Anlässen seine Urheberschaft im nachhinein verleugnen wollen. Es geht insgesamt um vier Punkte:

- Vertraulichkeit (disclosure protection, data confidentiality),
- Authentisierung des Absenders (origin authentication),
- Datenintegrität (data integrity),
- Nicht-Verleugnung des Absenders (non-repudiation of origin).

Mit PGP oder PEM verschlüsselte E-Mails bieten sogar mehr Sicherheit als ein eigenhändig unterzeichneter Brief in einem Umschlag. Außerdem wäre es angebracht, wenn alle E-Mails im Internet standardmäßig verschlüsselt würden. Solange dies nur bei wenigen Nachrichten geschieht, fallen diese besonders auf und erregen Mißtrauen. PGP und PEM sind Verfahren oder Protokolle, die in mehreren freien oder kommerziellen Programmpaketen realisiert werden. Der Mailversand erfolgt unverändert mittels der gewohnten Programme wie `elm(1)` und `sendmail(1)`.

PGP verschlüsselt den Klartext zunächst nach dem symmetrischen IDEA-Verfahren mit einem jedesmal neu erzeugten, nur einmal verwendeten, zufälligen Schlüssel. Dieser IDEA-Schlüssel wird anschließend nach einem Public-Key-Verfahren (RSA bei der PGP-Version 2.6.3) mit dem öffentlichen Schlüssel des Empfängers chiffriert. Neben einem deutlichen Geschwindigkeitsvorteil erlaubt diese Vorgehensweise auch, eine Nachricht relativ einfach an mehrere Empfänger zu verschicken. Hierzu braucht nur der IDEA-Schlüssel, nicht die gesamte Nachricht, für jeden Empfänger einzeln chiffriert zu werden.

Um die Integrität einer Nachricht sicherzustellen und den Empfänger zu authentisieren, versieht PGP eine Nachricht mit einer nach der Einweg-Hash-Funktion MD5 (Message Digest 5, RFC 1321) ermittelten Prüfzahl. Diese wird mit dem privaten Schlüssel des Absenders chiffriert. Der Empfänger ermittelt nach der gleichen Einweg-Hash-Funktion die Prüfzahl für den empfangenen Text. Anschließend decodiert er die mitgeschickte, chiffrierte Prüfzahl mit dem öffentlichen Schlüssel des Absenders. Stimmen die beiden Prüfzahlen überein, ist die Nachricht während der Übertragung nicht verändert worden. Hierbei wird die Tatsache ausgenutzt, daß es praktisch unmöglich ist, eine andere Nachricht mit gleicher Prüfzahl zu erzeugen.

Da die Prüfzahl nach der Decodierung mit dem öffentlichen Schlüssel des Absenders nur dann mit der errechneten Prüfzahl übereinstimmt, wenn sie zuvor mit dem dazugehörigen privaten Schlüssel chiffriert wurde, besteht auch Gewißheit über die Identität des Absenders. Somit kann eine digitale Unterschrift (Signatur) erstellt werden.

Der Haken bei diesem Verfahren besteht darin, daß ein Bösewicht ein Schlüsselpaar erzeugen könnte, das behauptet, von jemand anderem zu stammen. Daher

muß ein öffentlicher Schlüssel durch eine zentrale vertrauenswürdige Instanz (Certification Authority, CA) oder durch die digitalen Unterschriften von anderen, vertrauenswürdigen Personen bestätigt werden, bevor er als echt angesehen werden kann. Im Gegensatz zu PEM bietet PGP beide Möglichkeiten.

Bevor eine Nachricht für einen bestimmten Empfänger verschlüsselt werden kann, muß dessen öffentlicher Schlüssel bekannt sein. Mit etwas Glück kann dieser von einem sogenannten Key Server im Internet bezogen werden. Dabei ist zu beachten, daß die Aufgabe der Key Server nur in der Verbreitung, nicht in der Beglaubigung von öffentlichen Schlüsseln besteht.

Zur Beglaubigung von Schlüsseln entstehen in letzter Zeit immer mehr Certification Authorities. Diese zertifizieren einen Schlüssel im allgemeinen nur bei persönlichem Kontakt und nach Vorlage eines Identitätsausweises. Derartige CAs werden u. a. von der Computer-Zeitschrift c't, dem Individual Network und dem Deutschen Forschungsnetz DFN betrieben.

Eine Behinderung bei der Vebreitung von PGP sind die strengen Export-Gesetze der USA, die Verschlüsselungs-Software mit Kriegswaffen gleichsetzen und den Export stark einschränken. Daher gibt es eine US- und eine internationale Version von PGP.

Seit einigen Monaten ist auch außerhalb der Vereinigten Staaten die neue PGP-Version 5.0 erhältlich. Diese verwendet zum Teil andere, mindestens ebenso sichere Algorithmen und kommt unter Microsoft Windows mit einer komfortablen Oberfläche daher, hat sich aber noch nicht überall durchgesetzt. Informationen zu PGP, internationale Fassung, findet man auf `http://www.pgpi.com/`.

Alternativ zu PGP läßt sich PEM einsetzen, ein Internet-Protokoll, beschrieben in den RFCs 1421 bis 1424. Eine Implementation ist Riordan's **Internet Privacy Enhanced Mail (RIPEM)** von Mark Riordan. RIPEM verwendet zur symmetrischen Verschlüsselung den Triple-DES-Algorithmus, zur unsymmetrischen wie PGP den RSA-Algorithmus. Für die Verbreitung und Sicherung der öffentlichen Schlüssel sieht RIPEM mehrere Wege vor. PGP und PEM konkurrieren in einigen Punkten miteinander, in anderen setzen sie unterschiedliche Gewichte. PEM ist ein Internet-Protokoll, PGP weiter verbreitet.

Ähnliche Aufgaben wie bei Email stellen sich auch bei der Veröffentlichung von WWW-Dokumenten. Ohne besondere Maßnahmen könnte ein Bösewicht unter meinem Namen schwachsinnige oder bedenkliche HTML-Seiten ins Netz stellen, oder auch verfälschte Kopien meiner echten Seiten.

3.12 Neuigkeiten (Usenet, Netnews)

Das **Usenet** ist kein Computernetz, keine Organisation, keine bestimmte Person oder Personengruppe, keine Software, keine Hardware, sondern die Menge aller Computer, die die Netnews vorrätig halten. Genauer noch: die Menge der Personen, die mit Hilfe ihrer Computer die Netnews schreiben und verteilen. Diese Menge deckt sich nicht mit der Menge aller Knoten oder Benutzer des Internet. Nicht alle Internet-Hosts speichern die Netnews, umgekehrt gibt es auch außerhalb des Internets (im Bitnet/EARN beispielsweise) Hosts, die die Netnews speichern.

Netnews klingt nach Zeitung im Netz. Diese Zeitung

- wird im Internet und anderen Netzen verbreitet,
- besteht nur aus Leserbriefen,
- erscheint nicht periodisch, sondern stetig,
- hat keine Redaktion,
- behandelt alle Themen des menschlichen Hier- und Daseins.

Das funktioniert und kann so reizvoll werden, daß der davon befallene Leser zumindest vorübergehend zum Fortschritt der Menschheit nichts mehr beiträgt, sondern nur noch liest (*No Netnews before lunch*, dann ist wenigstens der Vormittag gerettet.). Zwei Dinge braucht der Leser (außer Zeit und Sprachkenntnissen):

- ein Programm zum Lesen und Schreiben, einen Newsreader wie `tin(1)` oder `trn(1)`,
- Verbindung zu einem News-Server (bei uns `news.rz.uni-karlsruhe.de`).

Als **Newsreader** setzen wir `tin(1)` für alphanumerische Terminals und `xn(1)` für X-Window-Systeme ein. Pager wie `more(1)` oder Editoren wie `vi(1)` sind zur Teilnahme an den Netnews ungeeignet, weil die Newsreader über das Lesen und Schreiben hinaus organisatorische Aufgaben erfüllen. Der Newsreader wird auf dem lokalen Computer aufgerufen und stellt eine Verbindung zu seinem Default-News-Server her. Üblicherweise arbeitet man immer mit demselben News-Server zusammen, man kann jedoch vorübergehend die Umgebungsvariable NNTPSERVER auf den Namen eines anderen Servers setzen. `tin(1)` spricht dann diesen an. Da `tin(1)` Buch darüber führt, welche Artikel man gelesen hat und sich diese Angaben auf den Default-Server beziehen, der andere aber eine abweichende Auswahl von Artikeln führt, kommt es leicht zu einem Durcheinander. Außerdem verweigern fremde News-Server meist den Zugang, ausprobiert mit `news.univ-lyon1.fr` und `news.uwasa.fi`. Schade. Öffentlich zugänglich sollen unter anderen `news.belwue.de`, `news.fu-berlin.de`, `news.uni-stuttgart.de` und `newsserver.rrzn.uni-hannover.de` sein, teilweise nur zum Lesen, nicht zum Posten.

Woher bezieht der News-Server seine Nachrichten? Wie kommt mein Leserbrief nach Australien? Eine zentrale Redaktion oder Sammelstelle gibt es ja nicht. Von dem lokalen Computer, auf dem der Newsreader oder -client läuft, wandert der Leserbrief zunächst zum zugehörigen News-Server. Dort kann er von weiteren Kunden dieses Servers sofort abgeholt werden. Von Zeit zu Zeit nimmt der News-Server Verbindung mit einigen benachbarten News-Servern auf und tauscht neue Artikel in beiden Richtungen aus. Da jeder News-Server Verbindungen zu wieder anderen hat, verbreitet sich ein Artikel innerhalb weniger Tage im ganzen Usenet. Das Verfahren wird dadurch beschleunigt, daß es doch so etwas wie übergeordnete Server gibt, die viele Server versorgen. Hat ein Artikel einen solchen übergordneten Server erreicht, verseucht er mit einem Schlag ein großes Gebiet. Das Zurückholen von Artikeln ist nur beschränkt möglich und vollzieht sich auf demselben Weg, indem man einen Artikel auf die Reise schickt, der eine Anweisung zum Löschen

des ersten enthält. Wieviele Leser den verunglückten Artikel schon gelesen haben (und entsprechend antworten), ist unergründlich. Also erst denken, dann posten.

Die Netnews sind umfangreich, sie sind daher wie eine herkömmliche Zeitung in Rubriken untergliedert, die **Newsgruppen** heißen. Bezeichnungen wie Area, Arena, Board, Brett, Echo, Forum, Konferenz, Round Table, Special Interest Group meinen zwar etwas Ähnliches wie die Newsgruppen, gehören aber nicht ins Internet. Der News-Server der Universität Karlsruhe hält eine Auswahl von rund 10000 (zehntausend) Gruppen bereit. Die Gruppen sind hierarchisch aufgeteilt:

- mainstream-Gruppen (die Big Eight), sollten überall vorrätig sein)
  - `comp`. Computer Science, Informatik für Beruf und Hobby
  - `humanities`. Humanities, Geisteswissenschaften
  - `misc`. Miscellaneous, Vermischtes
  - `news`. Themen zu den Netnews selbst
  - `rec`. Recreation, Erholung, Freizeit, Hobbies
  - `sci`. Science, Naturwissenschaften
  - `soc`. Society, Politik, Soziologie
  - `talk`. Diskussionen, manchmal end- und fruchtlos
- alternative Gruppen (nicht alle werden überall gehalten)
- deutschsprachige Gruppen (nur im deutschen Sprachraum)
- lokale (z. B. Karlsruher) Gruppen

Mit Hilfe des Newsreaders abonniert oder subskribiert man einige der Gruppen; alle zu verfolgen, ist unmöglich. Ein Dutzend Gruppen schafft man vielleicht. Für den Anfang empfehlen wir:

- `news.announce.newusers`
- `comp.unix.questions`
- `comp.lang.c`
- `de.newusers`
- `de.newusers.questions`
- `de.comm.internet`
- `de.comp.os.unix`
- `de.comp.lang.c`
- `de.sci.misc`

und je nach persönlichen Interessen noch

- `ka.uni.studium`
- `soc.culture.nordic`
- `de.rec.fahrrad`

- `rec.music.beatles`
- `rec.arts.startrek`[8]

Die Auswahl läßt sich jederzeit ändern, in `tin(1)` mit den Kommandos `y`, `s` und `u`. Viele Beiträge sind Fragen nebst Antworten. Sokratische Denkwürdigkeiten sind im Netz so selten wie im wirklichen Leben, das meiste ist Alltag – Dummheit, Arroganz oder böser Wille kommen auch vor. Das Netz sind nicht die Computer, sondern ihre Benutzer.

Nach Aufruf von `tin(1)` erscheint ein Menü der subskribierten Gruppen, man wählt eine aus und sieht dann in einem weiteren Menü die noch nicht gelesenen, neuen Artikel. Die interessierenden Artikel liest man und kann dann verschieden darauf reagieren:

- Man geht zum nächsten Artikel weiter. Der zurückliegende Artikel wird als gelesen markiert und erscheint nicht mehr im Menü. Man kann allerdings alte Artikel, soweit auf dem Server noch vorrätig (Verweilzeit zwei Tage bis vier Wochen), wieder hervorholen.
- Mit `s` (save) wird der Artikel in ein File gespeichert.
- Mit `o` (output) geht der Artikel zum Drucker.
- Man antwortet. Dafür gibt es zwei Wege. Ein **Follow-up** wird an den Artikel bzw. die bereits vorhandenen Antworten angehängt und wird damit veröffentlicht. Man tritt so vor ein ziemlich großes Publikum, unter Nennung seines Namens. Artikel plus Antworten bilden einen **Thread**. Ein **Reply** ist eine Antwort per Email nur an den ursprünglichen Verfasser des Artikels.

Fortgeschrittene (threaded) Newsreader folgen einem Thread und sogar seinen Verzweigungen, einfache unterscheiden nicht zwischen Artikel und Antwort. Bei einem Follow-up ist zu beachten, ob der ursprüngliche Schreiber sein Posting in mehreren Newsgruppen veröffentlicht hat. Die eigene Antwort gehört meist nur in eine. Überhaupt ist das gleichzeitige Posten in mehr als drei Newsgruppen eine Unsitte. Will man selbst einen Artikel schreiben (posten), wählt man in `tin(1)` das Kommando `w` wie write. Man sollte aber erst einmal einige Wochen lesen und die Gebräuche – die **Netiquette** – kennenlernen, ehe man das Netz und seine Leser beansprucht. Für Testzwecke stehen `test`-Newsgruppen bereit, die entweder keine oder eine automatische Antwort liefern und niemand belästigen.

Manche Fragen wiederholen sich, die Antworten zwangsläufig auch. Diese **Frequently Asked Questions** (FAQs; Fragen, Antworten, Quellen der Erleuchtung) werden daher mit den Antworten gruppenweise gesammelt und periodisch in den Netnews veröffentlicht. Außerdem stehen sie auf `rtfm.mit.edu` per FTP zur Verfügung, in Deutschland gespiegelt von `ftp.uni-paderborn.de`. Als erstes wäre *FAQs about FAQs* zu lesen, monatlich veröffentlicht in der Newsgruppe `news.answers` und unter `http://www.faqs.org/faqs/faqs/about-faqs`. Einige FAQs haben wir auch auf `ftp.ciw.uni-karlsruhe.de` kopiert, Sie brauchen also nicht lange zu suchen. Ihr Studium ist dringend anzuraten, man lernt einiges dabei, spart Zeit und schont das Netz.

[8] Erwähnt auf besonderen Wunsch eines Nachwuchs-Informatikers.

3.13 Netzgeschwätz (irc)

Früher trieben die Leute, die Muße hatten, Konversation. Heute treiben die Leute, die einen Internet-Anschluß haben, Kommunikation. Eine Form davon, die der Konversation nahe steht, ist das **Netzgeschwätz** oder der globale Dorftratsch mittels `irc(1)` (Internet Relay Chat). Man braucht:

- einen IRC-Server im Internet (bei uns `irc.rz.uni-karlsruhe.de`),
- einen IRC-Client auf dem eigenen System (bei uns `/usr/local/bin/irc`).

Nach Aufruf des Kommandos `irc(1)` erscheint ein Bildschirm, in dessen unterster Zeile man IRC-Kommandos ähnlich wie bei FTP eingeben kann, beispielsweise `/list`. Auf dem Schirm werden dann rund 1000 Gesprächskreise, sogenannte **Channels**, aufgelistet. Mittels `/join channel-name` schließt man sich einem Kreis an und je nachdem, ob die Teilnehmer ruhen oder munter sind, scrollt die die Diskussion langsamer oder schneller über den Schirm. Die IRC-Kommandos beginnen mit einem Schrägstrich, alle sonstigen Eingaben werden als Beitrag zur Runde verarbeitet. Hat man genug, tippt man `/quit`. Man muß das mal erlebt haben.

Unsere persönliche Haltung zum Netzgeschwätz ist noch unentschieden. Im Internet geht zwar die Sonne nicht unter, aber der Tag hat auch nur vierundzwanzig Stunden. Da unsere Zeit kaum reicht, Email und Netnews zu bewältigen, halten wir uns zurück. Aber vielleicht einmal als rüstige Rentner?

3.14 Suchhilfen: Archie, Gopher, WAIS

Im Netz liegt so viel an Information herum, daß man zum Finden der gewünschten Information bereits wieder einen Netzdienst beanspruchen muß. Sucht man ein bestimmtes File, dessen Namen man kennt, helfen die **Archies**. Das sind Server im Internet, die die Fileverzeichnisse einer großen Anzahl von FTP-Servern halten und Suchwerkzeuge zur Verfügung stellen. Nach Schlag- oder Stichwörtern kann zunächst nicht gesucht werden. Der älteste Archie ist `archie.mcgill.ca` in Kanada. Inzwischen gibt es weitere, auch in Deutschland (Darmstadt), siehe Anhang ?? *Netzadressen*. Auf dem eigenen Computer muß ein Archie-Client eingerichtet sein. Auf die Eingabe

```
archie
```

erhält man Hinweise zum Gebrauch (usage). Der Aufruf:

```
archie -s suchstring
```

führt zu einer Ausgabe aller Filenamen, auf die der Suchstring zutrifft, samt ihrer Standorte nach `stdout`, Umlenkung in ein File empfehlenswert. Die Option `-s` bewirkt die Suche nach einem Substring. Der Archie-Client wendet sich an seinen Default-Archie-Server, falls nicht ein bestimmter Server verlangt wird. Ruft man den Archie-Client interaktiv auf, stehen einige Archie-Kommandos bereit, darunter `whatis` zur Textsuche in Programmbeschreibungen, was einer Suche

nach Schlagwörtern nahe kommt. Archies sind nützlich, aber nicht allwissend: ihre Auskunft ist oft unvollständig, aber man hat meistens eine erste Fährte zu dem gefragten File.

Merke: Archies sagen, wo ein File liegt. Zum Beschaffen des Files braucht man ein anderes Programm (`ftp(1)`).

Nun zu den Gophern, der nächsten Stufe der Intelligenz und Bequemlichkeit. Ein **Gopher** ist mancherlei:

- eine Nadelbaumart, aus deren Holz Noah seine Arche gebaut hat (Genesis 6,14), vielleicht eine Zypresse,
- ein Vertreter des Tierreichs, Unterrreich Metazoa, Unterabteilung Bilateria, Reihe Deuterostomia, Stamm Chordata, Unterstamm Vertebrata, Klasse Mammalia, Unterklasse Placentalia, Ordnung Rodentia, Unterordnung Sciuromorpha, Überfamilie Geomyoidea, Familie Geomyidae, Gattung Geomys, zu deutsch eine Taschenratte[9], kleiner als unser Hamster, in Nord- und Mittelamerika verbreitet. Frißt Wurzeln (System-Manager Vorsicht!),
- ein menschlicher Einwohner des Staates Minnesota (Gopher State),
- ein Informationsdienst im Internet, der an der Universität von Minnesota entwickelt wurde.

Ein Gopher-Server im Internet hilft bei der Suche nach beliebigen Informationen. Auf dem eigenen Computer läuft ein Gopher-Client-Prozess, der sich mit seinem Gopher-Server verständigt. Der Benutzer wird über Menüs geführt. Die Gopher-Server sind intelligent; weiß einer nicht weiter, fragt er seinen Nachbarn – wie in der Schule. Der Benutzer merkt davon nichts. Hat man die gesuchte Information gefunden, beschafft der Gopher auch noch die Files, ohne daß der Benutzer sich mit Kermit, Mail oder FTP auseinanderzusetzen braucht. So wünscht man sich's. Hier die ersten beiden Bildschirme einer Gopher-Sitzung:

```
        Internet Gopher Information Client 2.0 pl7
           Rechenzentrum Uni Karlsruhe - Gopher

 1. Willkommen
 2. Anleitung/
 3. Universitaetsverwaltung/
 4. Zentrale Einrichtungen/
 5. Fakultaeten/
  .
 9. Sonstiges/
10. Mensaplan/
```

Wählen wir Punkt *9. Sonstiges* aus, gelangen wir in folgendes Menü:

```
        Internet Gopher Information Client 2.0 pl7
```

[9] keine Beutelratte (Didelphida). Diese Familie gehört zur Ordnung Marsupialia und ist in Südamerika verbreitet.

```
Sonstiges

1. Wissenschaftsfoerderung (Gopher-Giessen)/
2. Deutsche Kfz.-Kennzeichen (Gopher-Aachen).
3. Deutsche Bankleitzahlen (Gopher-ZIB, Berlin)<?>
4. Deutsche Postleitzahlen (Gopher-Muenchen)/
5. Postgebuehren (Gopher-ZIB, Berlin).
6. Telefonvorwahlnummern (Gopher-Aachen)<?>
7. Weather Images/Meteosat (Gopher-Hohenheim)/
```

Wie man sieht, holt sich der Karlsruher Gopher die Postleitzahlen von seinem Kollegen in München, ohne daß der Benutzer davon etwas zu wissen braucht.

Gopher-Clients für alle gängigen Computertypen liegen frei im Netz herum. Wir haben unseren Client für HP-UX bei `gopher.Germany.EU.net` geholt, UNIX-üblich als Quelle mit Makefile. Der Aufruf:

```
gopher gopher.ask.uni-karlsruhe.de
```

verbindet mit einem Gopher-Server der Universität Karlsruhe; gibt man keinen Namen an, erreicht man seinen Default-Gopher. Der Rest sind Menüs, die sich nach Art der verfügbaren Information unterscheiden. Ende mit `q` für quit. Der Gopher-Dienst ist inzwischen vom WWW stark zurückgedrängt worden.

Veronica ist ein Zusatz zum Gopher-Dienst, der Suchbegriffe auswertet. Eine Veronica-Suche erstreckt sich über eine Vielzahl von Gopher-Servern und liefert bei Erfolg Gopher-Einträge (Menüpunkte), die wie gewohnt angesprochen werden. Der Vorteil liegt darin, daß man sich nicht von Hand durch die Menüs zu arbeiten braucht. Der Veronica-Dienst wird von einigen Gopher-Servern angeboten, erfordert also keinen lokalen Veronica-Client oder ein Veronica-Kommando. Probieren Sie die folgende Gopher-Sitzung aus:

- mittels `gopher gopher.rrz.uni-koeln.de` mit dem Gopher-Server der Universität Köln verbinden,
- Punkt `2. Informationssuche mit ... Veronica ... /` wählen,
- Punkt `2. Weltweite Titelsuche (mit Veronica) <?>` wählen,
- Suchbegriff `veronica` eingeben,
- Ergebnis: 12 Seiten zu je 18 Einträgen (Files) zum Thema Veronica,
- Kaffee aufsetzen, anfangen zu lesen.

Jughead ist ein Dienst ähnlich Veronica, aber beschränkt auf eine Untermenge aller Gopher-Server, zum Beispiel auf eine Universität. Das hat je nach Aufgabe Vorteile. Jughead wird wie Veronica als Punkt eines Gopher-Menüs angesprochen.

WAIS (Wide Area Information Servers) ist ein Informationssystem zur Volltextsuche. Man braucht wieder – wie bei Gopher – einen lokalen WAIS-Client und eine Verbindung zu einem WAIS-Server, anfangs meist zu `quake.think.com`, der ein `directory-of-servers` anbietet, aus dem man sich eine lokale Liste von WAIS-Sources zusammenstellt. Als lokale Clienten kommen `swais(1)`,

`waissearch(1)` oder `xwais(1)` für das X Window System in Frage, erhältlich per Anonymous-FTP und mit den üblichen kleinen Anpassungen zu compilieren.

Nehmen wir an, unser in solchen Dingen nicht ungeübter System-Manager habe alles eingerichtet. Dann rufen wir `swais(1)` oder ein darum herumgewickeltes Shellscript `wais` auf. Es erscheint ein Bildschirm `Source Selection`, aus dem wir eine Informationsquelle auswählen, beispielsweise das uns nahestehende ASK-SISY. Dieses Wissen muß man mitbringen, ähnlich wie bei FTP. Man darf auch mehrere Quellen auswählen. Dann gibt man einige Suchwörter (keywords) ein, beispielsweise `wais`. Nach einiger Zeit kommt das Ergebnis (`Search Results`). An oberster Stelle steht die Quelle, die sich durch die meisten Treffer auszeichnet, was nicht viel über ihren Wert aussagt. Wir bleiben ASK-SISY treu, auch wenn es erst an dritter Stelle auftaucht. Nach nochmals einiger Zeit wird ein Dokument angezeigt (`Document Display`), nach Art und Weise von `more(1)`. Dieses können wir am Bildschirm lesen, in ein File abspeichern oder als Mail versenden. Hier beschreibt das Dokument ein Programm namens `wais`, das bei ASK-SAM per FTP erhältlich ist. Genausogut können Sie sich über die geografischen Fakten von Deutschland aufklären lassen, wozu als Quelle das World-Factbook des CIA auszuwählen wäre.

3.15 WWW – das World Wide Web

3.15.1 Hypertext

Hypertext oder, falls auch stehende oder bewegte Bilder sowie akustische Informationen eingeschlossen sind, **Hypermedia** sind Informationen, die bei den wesentlichen Stichwörtern Verweise (Links) auf weitere Informationen enthalten, die elektronisch auswertbar sind, so daß man ohne Suchen und Blättern weitergeführt wird. Auf Papier dienen Fußnoten, Literatursammlungen, Register, Querverweise und Konkordanzen diesem Zweck. Im ersten Kapitel war von Wallenstein die Rede. Von diesem Stichwort könnten Verweise auf das Schauspiel von Friedrich Schiller, den Roman von Alfred Döblin oder 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 schlimmer. So ist jede Information eingebettet in ein Gespinst oder Netz von Beziehungen zu anderen Informationen, und wir kommen zum World Wide Web.

3.15.2 Hypertext Markup Language (HTML)

Zum Schreiben von Hypertext-Dokumenten ist die **Hypertext Markup Language** (HTML) entworfen worden worden, gegenwärtig in der Version 4 im Netz. Zum Lesen von Hypertexten braucht man HTML-Browser wie `netscape`, `mosaic` oder den Internet-Explorer von Microsoft. Leider halten sich nicht alle Browser an den gültigen HTML-Standard. Sie erkennen nicht alle HTML-Konstrukte und bringen eigene (proprietäre) Dinge mit.

Ein einfache HTML-Dokument, das nicht alle Möglichkeiten von HTML 4.0 ausreizt, ist schnell geschrieben:

```
<HTML>

<HEAD>
<TITLE>Institut fuer Hoeheres WWW-Wesen</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff">

<IMG SRC="http:../logowww.gif" alt="">

<H3>
Fakult&auml;t f&uuml;r Internetingenieurwesen
</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>
Hubert Auriol

<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 10. Jan. 1998
<A HREF="mailto:webmaster@ciw.uni-karlsruhe.de"> webmaster@ciw.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 (`&auml;`) oder durch die Nummer im Latin-1-Zeichensatz (`&#228;`) 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. Im Netz sind mehrere Kurzanleitungen und die ausführliche Referenz zu HTML verfügbar.

3.15.3 Das Web

Das **World Wide Web**, **W3** oder **WWW**, entwickelt von Tim Berners-Lee am CERN in Genf, ist ein Informationssystem, das Dokumente nach dem Client-Server-Schema beschafft und verwertet. Mit dem Kommando `www(1)` landet man in seiner **Startseite**, dem Ausgangspunkt für alles weitere. Der lokale Client wird auch **Browser** genannt, ein Programm zum Betrachten von WWW-Dokumenten, etwas intelligenter als `more(1)`. Wenn der vorliegende Text für das WWW aufbereitet wäre (was mit viel Handarbeit verbunden ist), könnten Sie jetzt das Stichwort *WWW* auswählen und würden zu einer ausführlicheren Information geleitet, die sicher irgendwo im Netz herumliegt. So weit sind wir noch nicht.

Auf unserem System arbeiteten wir anfangs mit einer rustikalen zeilenorientierten Ausführung von `www(1)`. Die Startseite enthält einen allgemeinen Überblick über das Web und bietet unter anderem die Möglichkeit, ein Thema, einen Server oder einen Dienst per Ziffer auszuwählen. Wir entscheiden uns für die Suche per Thema und landen in der **WWW Virtual Library**, die heute achtundachtzig Themen im Angebot hat, darunter Verweise auf weitere Virtual Libraries. Wir entscheiden uns für Punkt 18: `Climate Research` und werden mit der Home Page des Deutschen Klimarechenzentrums verbunden. Dort weckt der Punkt 3: `Klimaforschung` unsere Neugier und unter diesem der Punkt 4: `The Climate of the Next Century`, das wir zu erleben hoffen. Punkt 4 gibt kurze Hinweise und bietet ein File von 1,6 MB Größe an. Wir nehmen das Angebot an und finden nach einigen Sekunden ein File mit der Kennung `.mpeg` in unserem lokalen Filesystem. Hier enden unsere technischen Möglichkeiten, denn dieses File ist ein digitaler Film, zu dessen Wiedergabe unser damaliges serielles, monochromes, alphanumerisches Terminal denkbar ungeeignet war.

WWW-Informationen werden gemäß dem **Hypertext Transfer Protocol HTTP** übertragen, beschrieben im RFC 2068 vom Januar 1997. Die Informationen selbst werden durch einen **Uniform Resource Locator URL** gekennzeichnet, im wesentlichen Protokoll-Host-Filename ähnlich wie eine Email-Anschrift:

```
http://hoohoo.ncsa.uiuc.edu:80/docs/Overview.html
```

Zuerst wird das Protokoll genannt, dann der Name des Hosts. Nach dem Doppelpunkt folgt die Portnummer, die samt Doppelpunkt entfallen kann, wenn der Defaultwert (80) zutrifft. Als letztes Glied kommt der Pfad des gewünschten Files oder Verzeichnisses. Die Browser gestatten das direkte Ansprechen von Informationen, sofern man deren URL kennt. Unter dem X Window System ist der Browser `mosaic(1)` oder `xmosaic(1)` des NCSA verbreitet, der eine Motif-ähnliche Oberfläche mit viel Komfort bietet. Mit `mosaic(1)` hätten wir uns den digitalen Film ansehen und -hören können. Inzwischen haben sich `netscape(1)` und der Internet Explorer von Microsoft verbreitet. Es gibt aber noch mehr Browser, darunter auch sehr einfache wie `lynx(1)` für Textterminals.

Die genannten Netzdienste sind allesamt Versuche, die in elektronischer Form vorliegenden Informationen – das sind heute schon viele, wenngleich nicht so viele wie auf Papier in einer Universitätsbibliothek – netzweit leicht zugänglich zu machen. Leicht heißt vor allem, unter einer einheitlichen Benutzer-Oberfläche, so daß der Benutzer sich nicht mit verschiedenen Such- und Beschaffungsverfahren (Protokollen) herumzuschlagen braucht. Die Versuche sind erst wenige Jahre alt und daher noch im Fluß.

3.16 Navigationshilfen (nslookup, whois, finger)

In den unendlichen Weiten des Netzes kann man sich leicht verirren. Sucht man zu einer numerischen IP-Anschrift den Namen oder umgekehrt, so hilft das Kommando `nslookup(1)` mit dem Namen oder der Anschrift als Argument. Es wendet sich an den nächsten Name-Server, dieser unter Umständen an seinen Nachbarn usw. Eine Auskunft sieht so aus:

```
Name Server:  netserv.rz.uni-karlsruhe.de
Address:  129.13.64.5

Name:    mvmpc2.ciw.uni-karlsruhe.de
Address:  129.13.118.2
Aliases:  ftp.ciw.uni-karlsruhe.de
```

Der Computer mit der IP-Anschrift 129.13.118.2 hat also zwei Namen: `mvmpc2.ciw.uni-karlsruhe.de` und `ftp.ciw.uni-karlsruhe.de`.

Das Kommando `whois(1)` verschafft nähere Auskünfte zu einem als Argument mitgegebenen Hostnamen, sofern der angesprochene Host – beispielsweise `whois.internic.net` oder `whois.nic.de` – diesen kennt:

```
whois -h whois.internic.net gatekeeper.dec.com
```

liefert nach wenigen Sekunden:

```
Digital Equipment Corporation (GATEKEEPER)

   Hostname: GATEKEEPER.DEC.COM
```

```
   Address: 16.1.0.2
   System: VAX running ULTRIX

   Coordinator:
      Reid, Brian K.  (BKR)  reid@PA.DEC.COM
      (415) 688-1307

   domain server

   Record last updated on 06-Apr-92.

To see this host record with registered users, repeat the command
with a star ('*') before the name; or, use '%' to show JUST the
registered users.

The InterNIC Registration Services Host ONLY contains Internet
Information (Networks, ASN's, Domains, and POC's).
Please use the whois server at nic.ddn.mil for MILNET Information.
```

Der Gatekeeper (Torwächter) ist ein gutsortierter FTP-Server von Digital Corporate Research (DEC) und nicht nur für DEC-Freunde von Reiz.

Geht es um Personen, hilft das Kommando `finger(1)`, das die Files `/etc/passwd(4)`, `$HOME/.project` und `$HOME/.plan` abfragt. Die beiden Dotfiles kann jeder Benutzer in seinem Home-Verzeichnis mittels eines Editors anlegen. `.project` enthält in seiner ersten Zeile (mehr werden nicht beachtet) die Projekte, an denen man arbeitet, `.plan` einen beliebigen Text, üblicherweise Sprechstunden, Urlaubspläne, Mitteilungen und dergleichen. Nicht alle Hosts anworten jedoch auf `finger`-Anfragen. Andere hinwiederum schicken ganze Textfiles zurück. Befingern Sie den Autor des LINUX-Betriebssystems:

```
finger torvalds@kruuna.helsinki.fi
```

so erhalten Sie eine Auskunft über den Stand des Projektes (gekürzt):

```
[kruuna.helsinki.fi]
Login: tkol_gr1         Name: Linus B Torvalds
Directory: /home/kruuna3/tkol/tkol_gr1 Shell: /usr/local/bin/expired
last login on klaava Wed Dec  1 15:00:17 1993 on ttyqf from hydra
New mail received Thu Dec  2 21:34:52 1993;
unread since Wed Dec  1 15:00:35 1993
No Plan.

Login: torvalds         Name: Linus Torvalds
Directory: /home/hydra/torvalds      Shell: /bin/tcsh
last login on klaava Fri Mar 18 18:53:52 1994 on ttyq2 from klaava
No unread mail
Plan:
Free UN*X for the 386
```

```
LINUX 1.0 HAS BEEN RELEASED! Get it from:

ftp.funet.fi pub/OS/Linux

and other sites. You'd better get the documentation from there too:
no sense in having it in this plan.
```

Ruft man `finger(1)` nur mit dem Namen einer Maschine als Argument auf, erfährt man Näheres über den Postmaster. Wird ein Klammeraffe (ASCII-Zeichen Nr. 64) vor den Maschinennamen gesetzt, werden die gerade angemeldeten Benutzer aufgelistet:

```
[mvmhp.ciw.uni-karlsruhe.de]
Login       Name           TTY Idle    When    Bldg.          Phone
wualex1  W. Alex           con      Fri 08:05  30.70.003       2404
gebern1  G. Bernoer        1p0 1:15 Fri 08:52  30.70.107       2413
```

Das Werkzeug `netfind(1)` hilft, die genaue Email-Anschrift eines Benutzers zu finden, von dem man nur ungenaue Angaben kennt. `netfind(1)` vereinigt mehrere Suchwerkzeuge und -verfahren, darunter `finger(1)`. Seine Tätigkeit ist einer Telefonauskunft vergleichbar.

3.17 Die Zeit im Netz (ntp)

3.17.1 Aufgabe

Computer – insbesondere im Netz – brauchen für manche Aufgaben die genaue Zeit. Dazu zählen Anwendungen wie `make(1)`, die die Zeitstempel der Files auswerten, Email, Datenbanken, einige Sicherheitsmechanismen wie Kerberos und natürlich Echtzeit-Aufgaben. Abgesehen von diesen Erfordernissen ist es lästig, wenn die Systemuhr zu sehr von der bürgerlichen Zeit abweicht. Die Systemuhr wird zwar vom Systemtakt und damit von einem Quarz gesteuert, dieser ist jedoch nicht auf die Belange einer Uhr hin ausgesucht. Mit anderen Worten: die Systemuhren müssen regelmäßig mit genaueren Uhren synchronisiert werden, fragt sich, mit welchen und wie.

Eine völlig andere Aufgabe ist die Messung von Zeitspannen, beispielsweise zur Geschwindigkeitsoptimierung von Programmen oder auch bei Echtzeit-Aufgaben. Hier kommt es auf eine hohe Auflösung an, aber nicht so sehr auf die Übereinstimmung mit anderen Zeitmessern weltweit. Außerdem benötigt man nur eine Maßeinheit wie die Sekunde und keinen Nullpunkt wie Christi Geburt.

3.17.2 UTC – Universal Time Coordinated

Die **Universal Time Coordinated** (UTC) ist die Nachfolgerin der Greenwich Mean Time (GMT), heute UT1 genannt, der mittleren Sonnenzeit auf dem Längengrad null, der durch die Sternwarte von Greenwich bei London verläuft.

Beide Zeiten unterscheiden sich durch ihre Definition und gelegentlich um Bruchteile von Sekunden. Als Weltzeit gilt seit 1972 die UTC. Computer-Systemuhren sollten UTC haben. Daraus wird durch Addition von 1 h die in Deutschland gültige Mitteleuropäische Zeit (MEZ) abgeleitet, auch Middle oder Central European Time genannt.

Wie kommt man zur UTC? Die besten Uhren (Cäsium- oder Atom-Uhren) laufen so gleichmäßig, daß sie für den Alltag schon nicht mehr zu gebrauchen sind, wie wir sehen werden. Ihre Zeit wird als **Temps Atomique International** (TAI) bezeichnet. In Deutschland stehen einige solcher Uhren in der Physikalisch-Technischen Bundesanstalt (PTB) in Braunschweig. Weltweit verfügen etwa 60 Zeitinstitute über Atom-Uhren. Aus den Daten dieser Uhren errechnet das Bureau International des Poids et Mesures (BIPM) in Paris einen Mittelwert, addiert eine vereinbarte Anzahl von Schaltsekunden hinzu und erhält so die internationale UTC oder UTC(BIPM). Dann teilt das Bureau den nationalen Zeitinstituten die Abweichung der nationalen UTC(*) von der internationalen UTC mit, bei uns also die Differenz $UTC - UTC(PTB)$. Sie soll unter einer Mikrosekunde liegen und tut das bei der PTB auch deutlich. Die UTC ist also eine nachträglich errechnete Zeit. Was die Zeitinstitute über Radiosender wie DCF77 verbreiten, kann immer nur die nationale UTC(*) sein, in Deutschland UTC(PTB). Mit den paar Nanosekunden Unsicherheit müssen wir leben.

Warum nun die Schaltsekunden? Für den Alltag ist die Erddrehung wichtiger als die Schwingung von Cäsiumatomen. Die Drehung wird allmählich langsamer – ein heutiger Tag ist bereits drei Stunden länger als vor 600 Millionen Jahren – und weist auch Unregelmäßigkeiten auf. Die UTC wird aus der TAI abgeleitet, indem nach Bedarf Schaltsekunden hinzugefügt werden, sodaß Mittag und Mitternacht, Sommer und Winter dort bleiben, wohin sie gehören. UTC und UT1 unterscheiden sich höchstens um 0,9 Sekunden, UTC und TAI durch eine ganze Anzahl von Sekunden, gegenwärtig (Ende 1997) um 31. Nicht jede Minute der jüngeren Vergangenheit war also 60 Sekunden lang. Erlaubt sind auch negative Schaltsekunden, jedoch noch nicht vorgekommen.

In den einzelnen Ländern sind nationale Behörden für die Darstellung der Zeit verantwortlich:

- Deutschland: Physikalisch-Technische Bundesanstalt, Braunschweig
- Frankreich:
- Schweiz:
- Österreich:
- England: National Physical Laboratory,
- USA: U. S. Naval Observatory

Außer der Zeit bekommt man von diesen Instituten oft auch Informationen über die Zeit und ihre Messung. In Deutschland wird die Zeit vor allem über den Sender DCF77 bei Frankfurt (Main) auf 77,5 kHz verteilt. Funkuhren, die dessen Signale verarbeiten, sind mittlerweile so preiswert geworden, daß sie kein Zeichen von Exklusivität mehr sind. Auch funkgesteuerte Computeruhren sind erschwinglich, so lange man keine hohen Ansprüche stellt.

3.17.3 Einrichtung

Man könnte jeden Computer mit einer eigenen Funkuhr ausrüsten, aber das wäre doch etwas aufwendig, zumal das Netz billigere und zuverlässigere Möglichkeiten bietet. Außerdem steht nicht jeder Computer an einem Platz mit ungestörtem Empfang der Radiosignale. Das **Network Time Protocol** nach RFC 1305 (Postscript-Ausgabe 120 Seiten) vom März 1992 zeigt den Weg.

Im Netz gibt es eine Hierarchie von Zeitservern. Das Fundament bilden die Stratum-1-Server. Das sind Computer, die eine genaue Hardware-Uhr haben, beispielsweise eine Funkuhr. Diese Server sprechen sich untereinander ab, sodaß der vorübergehende Ausfall einer Funkverbindung praktisch keine Auswirkungen hat. Von den Stratum-1-Servern holt sich die nächste Schicht die Zeit, die Stratum-2-Server. Als kleiner Netzmanager soll man fremde Stratum-1-Server nicht belästigen. Oft erlauben die Rechenzentren den direkten Zugriff auch nicht.

Stratum-2-Server versorgen große Netze wie ein Campus- oder Firmennetz mit der Zeit. Auch sie sprechen sich untereinander ab und holen sich außerdem die Zeit von mehreren Stratum-1-Servern. So geht es weiter bis zum Stratum 16, das aber praktisch nicht vorkommt, weil in Instituten oder Gebäuden die Zeit einfacher per Broadcast von einem Stratum-2- oder Stratum-3-Server verteilt wird. Die Mehrheit der Computer ist als Broadcast-Client konfiguriert.

Man braucht einen Dämon wie `xntp(1M)` samt ein paar Hilfsfiles. Die Konfiguration steht üblicherweise in `/etc/ntp.conf`. Dieses enthält Zeilen folgender Art:

```
server ntp.rz.uni-karlsruhe.de
server servus05.rus.uni-stuttgart.de
server 127.127.1.1
peer   mvmah90.ciw.uni-karlsruhe.de
peer   mvmpc100.ciw.uni-karlsruhe.de

broadcast 129.13.118.255

driftfile /etc/ntp.drift
```

Server sind Maschinen, von denen die Zeit geholt wird, Peers Maschinen, mit denen die Zeit ausgetauscht wird. Der Server 127.127.1.1 ist die Maschine selbst für den Fall, daß sämtliche Netzverbindungen unterbrochen sind. Da `ntp.rz.uni-karlsruhe.de` ein Stratum-1-Server ist, läuft die Maschine mit obiger Konfiguration als Stratum-2-Server. Sie sendet Broadcast-Signale in das Subnetz 129.13.118. Außerdem spricht sie sich mit ihren Kollegen `mvmah90` und `mvmpc100` ab, die zweckmäßigerweise ihre Zeit von einer anderen Auswahl an Zeitservern beziehen. Der Ausfall einer Zeitquelle hat so praktisch keine Auswirkungen. In `/etc/ntp.drift` wird die lokale Drift gespeichert, sodaß die Systemuhr auch ohne Verbindung zum Netz etwas genauer arbeitet. Weitere Computer im Subnetz 129.13.118 sind mit `broadcastclient yes` konfiguriert.

Da die Synchronisation nur bis zu einer gewissen Abweichung arbeitet, ist es angebracht, beim Booten mittels des Kommandos `ntpdate(1M)` die Zeit zu

setzen. Mit dem Kommando `ntpq(1M)` erfragt man aktuelle Daten zum Stand der Synchronisation.

... aber die Daten fehlen, um den ganzen
Nonsens richtig zu überblicken –
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 127. 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

B Zeichensätze

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	≪		è
139	213	≫		ı
140	214			î
141	215	ý		ì
142	216			Ä
143	217	±		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		ÿ
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	∼	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	¡	ˆ	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	,
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	$\emptyset$
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	a	$\bullet$
250	372		o	$\cdot$
251	373	Û	$\ll$	$\surd$
252	374	Ü	⊔	n
253	375	Ù	$\gg$	2
254	376	Ú	$\pm$	⊔
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.

dezimal	ASCII	Bedeutung	Tasten
0	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	bel	Bell	control g
8	bs	Backspace	control h, BS
9	ht	Horizontal tab	control i, TAB
10	lf	Line feed	control j, LF
11	vt	Vertical tab	control k
12	ff	Form feed	control l
13	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 of transmission block	control w
24	can	Cancel	control x
25	em	End of medium	control y
26	sub	Substitute	control z
27	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
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	≪	Left-pointing double angle quotation mark
172	254	ac	¬	Not sign
173	255	ad	-	Soft hyphen
174	256	ae		Registered sign
175	257	af	¯	Overline
176	260	b0	°	Degree sign

177	261	b1	±	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	º	Masculine ordinal indicator
187	273	bb	»	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		Latin capital letter l with stroke
165	245	a5		Latin capital letter l with caron
166	246	a6		Latin capital letter s with acute

169	251	a9	Latin capital letter s with caron
170	252	aa	Latin capital letter s with cedilla
171	253	ab	Latin capital letter t with caron
172	254	ac	Latin capital letter z with acute
174	256	ae	Latin capital letter z with caron
175	257	af	Latin capital letter z with dot above
177	261	b1	Latin small letter a with ogonek
178	262	b2	Ogonek
179	263	b3	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

C Die wichtigsten UNIX-Kommandos

Einzelheiten siehe Referenz-Handbücher – vor allem on-line. Das wichtigste Kommando zuerst, die übrigen nach Sachgebiet und dann alphabetisch geordnet:

`man man` Beschreibung zum Kommando `man(1)` ausgeben

Allgemeines

`alias` Alias in der Shell einrichten
```
alias r='fc -e -'
```
`at` Programm zu einem beliebigen Zeitpunkt starten
```
at 0815 Jan 24
myprogram1
EOF (meist control-d)
```
`bdf, df, du` Plattenbelegung ermitteln
```
df
```
`calendar` Terminverwaltung (Reminder Service)
(File `$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 myprocess_id
kill -s SIGHUP myprocess_id
```
`leave` an Feierabend erinnern
```
leave 2215
```
`lock` Terminal sperren
```
lock
```
`newgrp` Benutzergruppe wechseln
```
newgrp student
```
`nice` Priorität eines Programmes herabsetzen
```
nice myprogram
```
`nohup` Programm von Sitzung abkoppeln
```
nohup myprogram &
```
`passwd` Passwort ändern
```
passwd
```

`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/{}
```

Files, Verzeichnisse

`cd` Arbeitsverzeichnis wechseln
```
cd
cd /usr/local/bin
```
`chgrp` Gruppe eines Files wechseln
```
chgrp students myfile
```
`chmod` Zugriffsrechte eines Files ändern
```
chmod 755 myfile
```
`chown` Besitzer eines Files wechseln
```
chown aralex1 myfile
```
`cmp, diff` zwei Files vergleichen
```
cmp myfile1 myfile2
```
`compress` File komprimieren
```
compress myfile
uncompress myfile.Z
```
`cp` File kopieren
```
cp original kopie
```
`file` Filetyp ermitteln
```
file myfile
```
`find, whereis` Files suchen
```
find . -name myfile -print
```
`gzip` File komprimieren (GNU)

```
                    gzip myfile
                    gunzip myfile.gz
ln                  File linken
                    ln myfile hardlinkname
                    ln -s myfile softlinkname
ls                  Verzeichnisse auflisten
                    ls -al
mkdir               Verzeichnis anlegen
                    mkdir newdir
mv                  File umbenennen
                    mv oldfilename newfilename
od                  (oktalen) Dump eines Files ausgeben
                    od -c myfile
pwd                 Arbeitsverzeichnis anzeigen
                    pwd
rm, rmdir           File oder leeres Verzeichnis löschen
                    rm myfile
                    rm -r mydir
                    rmdir mydir
tar                 File-Archiv schreiben oder lesen
                    tar -cf /dev/st0 ./mydir &
touch               leeres File erzeugen, Zeitstempel ändern
                    touch myfile
```

Kommunikation, Netz

```
archie              nach File suchen
                    archie -s mysubstring > mysubstring.archie &
finger              Auskunft über Benutzer
                    finger wualex1@mvmpc100.ciw.uni-karlsruhe.de
ftp                 File Transfer
                    ftp ftp.ciw.uni-karlsruhe.de
hostname            Hostnamen anzeigen
                    hostname
irc                 Netzgeschwätz
                    irc (beenden mit /quit)
kermit              File übertragen, auch von/zu Nicht-UNIX-Anlagen
                    kermit (beenden mit exit)
mail, elm           Mail lesen und versenden
                    mail wulf.alex@ciw.uni-karlsruhe.de < myfile
                    elm
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 myarchiv.a myfile1 myfile2
```

`cb` C-Beautifier, Quelle verschönern

```
                    cb myprog.c > myprog.b
```

`cc` C-Compiler mit Linker

```
                        cc -o myprog myprog.c
```

`lint` C-Syntax-Prüfer

```
                    lint myprog.c
```

`make` Compileraufruf vereinfachen

`make` (Makefile erforderlich)

`sdb, xdb` symbolischer Debugger

`xdb` (einige Files erforderlich)

Textverarbeitung

`adjust` Text formatieren (einfachst)

```
                    adjust -j -m60 mytextfile
```

`awk` Listengenerator

`awk -f myawkscript mytextfile` (awk-Script erforderlich)

`cancel` Druckauftrag löschen

```
                    cancel lp-4711
```

`cat` von `stdin` lesen, nach `stdout` schreiben

```
                     cat mytextfile
                    cat myfile1 myfile2 > myfile.all
                    cat mytextfile
```

`cut` Spalten aus Tabellen auswählen

```
                cut -f1 mytablefile > newfile
```

`ed` Zeileneditor, mit `diff(1)` nützlich

```
ed mytextfile
```

`emacs` Editor, alternativ zum `vi(1)` (GNU)

```
emacs mytextfile
```

`expand` Tabs ins Spaces umwandeln

```
expand mytextfile > newfile
```

`grep, fgrep` Muster in Files suchen

```
grep -i Unix mytextfile
fgrep UNIX mytextfile
```

`head, tail` Anfang bzw. Ende eines Textfiles anzeigen

```
head mytextfile
```

`lp` File über Spooler ausdrucken

```
lp -dlp2 mytextfile
```

`lpstat, lpq` Spoolerstatus anzeigen

```
lpstat -t
```

`more, less, pg` Textfile schirmweise anzeigen

```
more mytextfile
ls -l | more
```

`nroff` Textformatierer

```
nroff mynrofffile | 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' mytextfile > newfile
```

`sort` Textfile zeilenweise sortieren

```
sort myliste | uniq > newfile
```

`spell` Rechtschreibung prüfen

```
spell myspelling textfile+
```

`tee` `stdout` zugleich in ein File schreiben

```
who | tee whofile
```

`tr` Zeichen in Textfile ersetzen

```
tr -d "\015" < mytextfile1 > mytextfile2
```

`uniq` sortiertes Textfile nach doppelten Zeilen durchsuchen

```
sort myliste | uniq > newfile
```

`vi` Bildschirm-Editor, alternativ zum `emacs(1)`

```
vi mytextfile
```

`view` `vi(1)` nur zum Lesen aufrufen

```
view mytextfile
```

`vis` Files mit Steuersequenzen anzeigen

```
vis mytextfile
```

`wc` Zeichen, Wörter und Zeilen zählen

```
wc mytextfile
```

Verwaltung (hier reichen Beispiele nicht, nachlesen!)

`acctsh`	Accounting System einrichten
`backup, restore`	Shellscript für Backups
`fsck`	File System Check
`lpadmin, lpc`	Drucker-Spooler einrichten
`mkfs`	Filesystem einrichten
`mknod`	Gerätefile einrichten
`mount`	weiteres File-System anschließen
`mvdir`	Verzeichnis verschieben
`stty`	Terminal-Schnittstelle konfigurieren
`tic, untic`	`terminfo`-Eintrag übersetzen
`shutdown`	System zum Abschalten vorbereiten

D Besondere UNIX-Kommandos

D.1 printf(3), scanf(3)

`printf(3)` und `scanf(3)` sind die beiden Standardfunktionen zum Ein- und Ausgeben von Daten. Wichtiger Unterschied: `printf(3)` erwartet Variable, `scanf(3)` Pointer. Die Formatbezeichner stimmen weitgehend überein:

Bezeichner	Typ	Beispiel	Bedeutung
%c	char	a	Zeichen
%s	char *	Karlsruhe	String
%d	int	-1234	dezimale Ganzzahl mit Vorzeichen
%i	int	-1234	dezimale Ganzzahl mit Vorzeichen
%u	unsigned	1234	dezimale Ganzzahl ohne Vorzeichen
%ld	long	1234	dezimal Ganzzahl doppelter Länge
%f	double	12.34	Gleitkommazahl mit Vorzeichen
%e	double	1.234 E 1	Gleitkommazahl, Exponentialdarstellung
%g	double	12.34	kürzere Darstellung von %e oder %f
%o	unsigned octal	2322	oktale Ganzzahl ohne Vorzeichen
%x	unsigned hex	4d2	hexadezimale Ganzzahl o. Vorzeichen
%p	void *	68ff32e4	Pointer
%%	-	%	Prozentzeichen

Weiteres im Referenz-Handbuch unter `printf(3)` oder `scanf(3)`. Länge, Bündigkeit, Unterdrückung führender Nullen, Vorzeichenangabe können festgelegt werden.

D.2 vi(1)

escape	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)` oder in dem Buch von Morris I. Bolsky.

D.3 emacs(1)

D.4 joe(1)

D.5 ftp(1)

Wenn man eine `ftp`-Verbindung zu einem Computer im Netz unterhält, stehen nur Kommandos für die Fileübertragung zur Verfügung, die mit den UNIX-Kommandos nichts zu tun haben. Die wichtigsten von etwa 70 sind:

Kommando	Wirkung
ftp	Beginn der FTP-Sitzung
open	Verbinden mit angegebener Adresse
user	Eingabe Benutzernamen
pwd	print working directory, wie UNIX
cd	change directory, wie UNIX
lcd	change local directory

dir	Verzeichnis auflisten (geht immer)
ls	Verzeichnis auflisten, wie UNIX (geht meist)
ascii	in ASCII-Modus schalten (für ASCII-Texte)
binary	in binären Modus schalten (für alle übrigen Files)
get	File vom Host holen
get README |more	kurzes Textfile `README` on-line lesen
put	File zum Host schicken
mget	multi-get, mehrere Files auf einmal holen
mput	multi-put, mehrere Files auf einmal schicken
passive	auf passives FTP umschalten (Firewall)
prompt	Abfragen bei `mget` und `mput` unterdrücken
rhelp	Kommandos der Übertragung anzeigen
rstatus	Status der Verbindung anzeigen
status	Status des Hosts anzeigen
close	Verbindung (nicht Sitzung) beenden
bye	Sitzung beenden
quit	Sitzung beenden
help	lokale Hilfe zu Kommandos anzeigen
? cmd	Hilfe zum Kommando `cmd` anzeigen

Eine FTP-Verbindung läßt sich teilweise automatisieren, siehe das Dot-File `.netrc`. Ferner kann man die Kommandos in ein Batchfile packen, das in Abwesenheit des Benutzers ausgeführt wird, was für regelmäßig wiederkehrende Verbindungen (`cron(1)`) zweckmäßig ist.

E UNIX-Systemaufrufe

Systemaufrufe werden vom Anwendungsprogramm wie eigene oder fremde Funktionen angesehen. Ihrem Ursprung nach sind es auch C-Funktionen. Sie sind jedoch nicht Bestandteil einer Funktionsbibliothek, sondern gehören zum Betriebssystem und sind nicht durch andere Funktionen erweiterbar.

Die Systemaufrufe – als Bestandteil des Betriebssystems – sind für alle Programmiersprachen dieselben, während die Funktionsbibliotheken zur jeweiligen Programmiersprache gehören. Folgende Systemaufrufe sind unter UNIX verfügbar:

`access`	prüft Zugriff auf File
`acct`	startet und stoppt Prozess Accounting
`alarm`	setzt Weckeruhr für Prozess
`atexit`	Funktion für Programmende
`brk`	ändert Speicherzuweisung
`chdir`	wechselt Arbeitsverzeichnis
`chmod`	ändert Zugriffsrechte eines Files
`chown`	ändert Besitzer eines Files
`chroot`	ändert Root-Verzeichnis
`close`	schließt einen File-Deskriptor
`creat`	öffnet File, ordnet Deskriptor zu
`dup`	dupliziert File-Deskriptor
`errno`	Fehlervariable der Systemaufrufe
`exec`	führt ein Programm aus
`exit`	beendet einen Prozess
`fcntl`	Filesteuerung
`fork`	erzeugt einen neuen Prozess
`fsctl`	liest Information aus File-System
`fsync`	schreibt File aus Arbeitsspeicher auf Platte
`getaccess`	ermittelt Zugriffsrechte
`getacl`	ermittelt Zugriffsrechte
`getcontext`	ermittelt Kontext eines Prozesses
`getdirentries`	ermittelt Verzeichnis-Einträge
`getgroups`	ermittelt Gruppenrechte eines Prozesses
`gethostname`	ermittelt Namen des Systems
`getitimer`	setzt oder liest Intervall-Uhr
`getpid`	liest Prozess-ID
`gettimeofday`	ermittelt Zeit
`getuid`	liest User-ID des aufrufenden Prozesses
`ioctl`	I/O-Steuerung
`kill`	schickt Signal an einen Prozess
`link`	linkt ein File

`lockf`	setzt Semaphore und Record-Sperren
`lseek`	bewegt Schreiblesezeiger in einem File
`mkdir`	erzeugt Verzeichnis
`mknod`	erzeugt File
`mount`	mountet File-System
`msgctl`	Interprozess-Kommunikation
`nice`	ändert die Priorität eines Prozesses
`open`	öffnet File zum Lesen oder Schreiben
`pause`	suspendiert Prozess bis zum Empfang eines Signals
`pipe`	erzeugt eine Pipe
`prealloc`	reserviert Arbeitsspeicher
`profil`	ermittelt Zeiten bei der Ausführung eines Programmes
`read`	liest aus einem File
`readlink`	liest symbolisches Link
`rename`	ändert Filenamen
`rmdir`	löscht Verzeichnis
`rtprio`	ändert Echtzeit-Priorität
`semctl`	Semaphore
`setgrp`	setzt Gruppen-Zugriffsrechte eines Prozesses
`setuid`	setzt User-ID eines Prozesses
`signal`	legt fest, was auf ein Signal hin zu tun ist
`stat`	liest die Inode eines Files
`statfs`	liest Werte des File-Systems
`symlink`	erzeugt symbolischen Link
`sync`	schreibt Puffer auf Platte
`szsconf`	ermittelt Systemwerte
`time`	ermittelt die Systemzeit
`times`	ermittelt Zeitverbrauch eines Prozesses
`truncate`	schneidet File ab
`umask`	setzt oder ermittelt Filezugriffsmaske
`unlink`	löscht File
`ustat`	liest Werte des File-Systems
`utime`	setzt Zeitstempel eines Files
`wait`	wartet auf Ende eines Kindprozesses
`write`	schreibt in ein File

Die Aufzählung kann durch weitere Systemaufrufe des jeweiligen Lieferanten des Betriebssystems (z. B. Hewlett-Packard) ergänzt werden. Diese erleichtern das Programmieren, verschlechtern aber die Portabilität. Zu den meisten Systemaufrufen mit `get...` gibt es ein Gegenstück `set...`, das in einigen Fällen dem Superuser vorbehalten ist.

F 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	Nummer	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 attempted from control terminal
SIGTTOU	28	background write attempted 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

G File-Kennungen

Unter UNIX ist es nicht gebräuchlich, den Typ eines Files durch ein Anhängsel (Kennung, Erweiterung, Extension) an seinen Namen zu kennzeichnen, aber es ist erlaubt. Die Länge der Kennung ist beliebig. Folgende Kennungen (a = ASCII-Text, b = binäres File, ? = wechselnd oder unbekannt) sind verbreitet:

$$$	?	Temporäres File
a	b	UNIX-Archiv, mit `ar(1)` erzeugt
adf	?	Adapter Description File (IBM)
adi	b	AutoCAD DXF
adn	?	Add In Utility (Lotus)
afm	?	Adobe Font Metrics (Adobe)
ani	b	Atari ST Graphics Format
ans	b	ANSI-Grafik
app	?	Application (RBase, NeXT)
arc	?	Archiv, mit `arc` oder `pkpak` komprimiert
arj	b	Archiv, mit ??? komprimiert
asc	a	ASCII-Textfile
asm	a	Assembler-Quelle
au	b	Sound-File (Sun, NeXT)
aux	?	TeX Hilfsfile, mit Bezügen
avi	b	Microsoft RIFF
avs	b	Intel DVI
bak	a	Backup-File
bas	b	BASIC-File
bat	a	Batchfile unter MS-DOS, entsprechend Shellscript
bbl	a	BIBTeX Literaturverzeichnis
bfx	b	Bitfax-File
bgi	b	Borland Graphic Interface
bib	?	Bibliographie
bif	b	Binary Image File
bin	b	binäres File
bit	b	TeX Ausgabe des Druckertreibers
bk	a	Backup-File (WordPerfect)
bld	b	BASIC Bload Graphics
bmp	b	Microsoft Bitmap Graphics (Windows, OS/2)
bnd	b	Microsoft RIFF
bpx	b	Lumena Paint
bsc	?	Boyan Script-File
bsv	b	BASIC Bsave Graphics
btm	a	Batchfile unter 4DOS
bw	b	SGI Image File Format
c	a	C-Quelle

C	a	C++-Quelle
cal	b	CAL Raster
cap	a	Capture-File (Telix)
cat	a	Catalogue, Verzeichnis
cbl	a	COBOL-Quelle
cc	a	C++-Quelle
cco	b	Btx-Grafik
cdf	?	Comma delimited format
cdr	b	Corel Draw Vektorgrafik
cdx	?	Compund index (FoxPro)
cfg	?	Konfigurations-File
cgm	b	Computer Graphics Metafile (Harvard Graphics, Lotus)
chk	?	von chkdsk erzeugtes File (MS-DOS)
cht	b	Harvard Graphics
clp	?	Clipboard (Windows), Clip Art, GRASP
cmd	a	Command, Skript, Batchfile unter OS/2
cmi	b	Intel DVI
cnc	a	CNC-Programme
cnf	?	Konfigurations-File
cob	a	COBOL-Quelle
cod	?	Codeliste
com	b	Kommando-File (MS-DOS), ausführbares Programm
cpi	?	Code Page Information (MS-DOS)
cpp	a	C++-Quelle
cpt	?	Compact Pro (Kompressor Macintosh)
crd	?	Cardfile
crf	?	Cross Reference File
csv	a	Comma Separated Values
ctx	b	Signaturfile (PGP)
cut	b	Dr. Halo Bitmap
cvf	b	Compressed Volume File
cxx	a	C++-Quelle
dat	a	Daten
db	b	Datenbank (Paradox)
dbf	?	Datenbank (dBase)
dcf	?	Driver Configuration file
dct	?	Dictionary
dcx	b	Fax-File
dd	?	Disk Doubler
def	?	Definitionen, Defaultwerte
dem	b	Demonstration
des	?	Description
dhp	b	Dr. Halo PIC
dib	b	Microsoft Windows Bitmap, OS/2 Bitmap
dic	?	Dictionary
dif	?	Lotus Data Interchange Format
dir	a	Directory, Verzeichnis
dll	b	Dynamically Linked Library (OS/2, Windows)
doc	a	Dokument, Textfile

dok	a	Dokument, Textfile
drs	b	Driver Resource (WordPerfect)
drv	b	Device Driver
drw	b	Drawing
dta	a	Daten
dv	a	Desqview Scriptfile
dvi	b	Metafile von TeX, geräteunabhängig
dvr	b	Device Driver
dxb	b	Drawing Interchange Binary (AutoCAD)
dxf	b	Data Exchange Format (Autocad)
ega	b	EGA-Grafik
el	a	Emacs Lisp
elc	b	Emacs Lisp compiled
eml	a	Electronic Mail
enc	b	encoded
eps	b	Encapsulated Postscript
err	a	Error, Fehlerprotokoll
etx	?	Setext File
exe	b	executable, ausführbares Programm (MS-DOS)
f	a	FORTRAN-Quelle
fax	b	Fax-File
fif	?	Fractal Image Format
fli	b	EmTeX Fontlibrary
fnt	b	Fontfile
for	a	FORTRAN-Quelle
fxs	b	Fax-File (Winfax)
gem	?	GEM Metadateien
gfb	b	Gifblast compressed GIF image
gif	b	Graphics Interchange File
glo	a	TeX Glossar
gz	b	mit GNU `gzip(1)` komprimiert
h	a	Header-File , include-File
hex	b	Hexdump
hdf	?	Hierarchical Data Format
hpp	a	Header-File C++
hqx	b	Macintosh BinHex encoded
htm	a	Hypertext-Markup-Language-File (DOS-Welt)
html	a	Hypertext-Markup-Language-File (UNIX-Welt)
i	a	C-Quelle nach Präprozessor-Durchlauf
idx	a	Index
iff	b	Interchange File Format (Amiga, Autodesk Animator)
img	b	GEM Paint, Rastergrafik/Bitmap
inf	a	Information
ini	?	Konfigurationsfile (Initialisierung)
jfi	b	JPEG File Interchange Format
jif	b	JPEG Interchange Format
jpeg	b	JPEG File
jpg	b	JPEG File
l	a	`lex(1)`-Quelle

lib	b	Library, Bibliothek
lof	a	TeX Bildverzeichnis (list of figures)
log	a	Protokollfile, TeX Meldungen während der Übersetzung
lot	a	TeX Tabellenverzeichnis (list of tables)
lst	a	Liste
lzh	b	Archiv, mit `lha` komprimiert
m4	a	`m4(1)`-Präprozessor-File
mac	b	Macintosh Paint
map	b	Map-Files (Quick C)
mak	a	Make-File
mdf	?	Menu Definition File
mid	b	Midi Sound Ffile
mif	b	Maker Interchange Format (FrameMaker)
mod	b	MODULA-Quelle, Amiga-Modul (Sound)
mpeg	b	Motion Pictures Expert Group (Filme)
msp	b	Microsoft Windows Paint
nff	?	Neutral File Format
o	b	Objekt-Code
obj	b	Objekt-Code
old	?	Kopie eines Files vor Überschreiben
ovl	b	Overlay-File
ovr	b	Overlay-File
p	a	PASCAL-Quelle
pak	b	Archiv, mit `pkpak` komprimiert
pax	b	Portable Archive Exchange
pcd	b	Kodak Photo Compact Disc
pct	b	Macintosh PICT
pcx	b	Paintbrush, Rastergrafik
pic	b	Lotus Picture File
pif	b	Program Information File
pit	b	PackIt file (Kompressor Macintosh)
pix	b	Inset PIX, Lumena Paint
pov	b	Persistance of Vision
prn	a	Druckerfile
prt	b	Parallel Ray Trace
ps	b	Postscript-File
psf	b	Permanent Swap File
qfx	b	Fax-File (Quicklink)
qtm	b	QuickTime
r	a	Rational-FORTRAN-Quelle
ras	b	SUN/CALS Raster Grafik
rdi	b	Microsoft RIFF
rff	b	Dore Raster File Format
rla	b	Wavefront Run Length Encoded Version A
rlb	b	Wavefront Run Length Encoded Version B
rpm	?	Red Hat Package Manager
rtf	?	Rich Text Format
s	a	Assembler-Quelle
sdf	?	Space Delimited File

sea	?	Self Extracting Archive (Macintosh)
sgi	b	SGI Image File Format
sh	a	Bourne-Shell-Script
shar	a	Bourne-Shell-Archiv
shr	a	Shell-Archiv
sit	b	Macintosh StuffIt Archive
src	a	Program Source Code
stf	?	Structured File
sys	b	Treiber unter MS-DOS
tar	b	`tar(1)`-Archiv
tdf	?	Trace/Typeface Definition File
tex	a	TeX- oder LaTeX-Quelltext
tfm	b	TeX-Font-Metrics
tga	b	TARGA-Bildfile
tgz	b	`.tar.gz`, tar-Archiv, GNUzip komprimiert
tif	b	Tag Image File Format (Scanner)
tmp	?	temporäres File
toc	a	Hilfsfile von TeX (table of contents)
tpu	b	Turbo Pascal Unit
ttf	b	TrueType Font
txt	a	Textfile, lesbar
tz	b	`tar.Z`, tar-Archiv, `compress(1)`-komprimiert
uu	a	uuencoded File, ASCII, aber nicht lesbar
uud	a	uuencoded File, ASCII, aber nicht lesbar
uue	a	uuencoded File, ASCII, aber nicht lesbar
vcf	b	Visiting Card File
voc	b	Creative Labs Sound File
wav	b	Microsoft Windows Sound File (RIFF)
web	a	WEB-Quelle
wfx	b	Fax-File (Winfax)
wmf	b	Windows Metafile Format
wpg	a	WordPerfect Graphics Metafile
wri	a	Windows Textfile
wps	a	MS Works Textfile
x	b	SuperDisk self-extracting archive
xwd	b	X Window Dump
y	a	`yacc(1)`-Quelle
Z	b	mit `compress(1)` (UNIX) komprimiert
z	b	mit GNU `gzip` komprimiert (veraltet, besser gz)
zip	b	mit `pkzip` komprimiert
zoo	b	mit `zoo` komprimiert

Postscript-Files bestehen zwar aus ASCII-Zeichen und lassen sich auch editieren, sofern man die Sprache kennt, dürfen aber nur im binären Modus von FTP übertragen werden, um keine Zeichen zu verändern.

H Slang im Netz

Diese Sammlung von im Netz vorkommenden Slang-Abkürzungen ist ein Auszug aus der Abklex-Liste (`http://www.ciw.uni-karlsruhe.de/abklex.html`) mit rund 6000 Abkürzungen aus Informatik und Telekommunikation.

AAMOF	As A Matter Of Fact (Slang)
AEG	Auspacken, Einschalten, Geht (nicht) (Slang)
AFAIAA	As Far As I Am Aware (Slang)
AFAIC	As Far As I am Concerned (Slang)
AFAIK	As Far As I Know (Slang)
AFJ	April Fool's Joke (Slang)
AFK	Away From Keyboard (Slang)
AIMB	As I Mentioned Before (Slang)
AISI	As I See It (Slang)
AIUI	As I Understand It (Slang)
AKA	Also Known As (Slang)
ANFSCD	And Now For Something Completely Different (Slang)
APOL	Alternate Person On Line (Slang)
ASAP	As Soon As Possible (Slang),
ATFSM	Ask The Friendly System Manager (Slang)
ATM	At The Moment (Slang)
ATST	At The Same Time (Slang)
ATT	At This Time (Slang)
AWA	A While Ago (Slang)
AWB	A While Back (Slang)
AYOR	At Your Own Risc (Slang)
B4N	Bye For Now (Slang)
BBL	Be Back Later (Slang)
BBR	Backbone Ring, Burnt Beyond Recognition (Slang)
BCNU	Be Seeing You (Slang)
BFBI	Brute Force and Bloody Ignorance (Slang)
BFI	Brute Force and Ignorance (Slang)
BFMI	Brute Force and Massive Ignorance (Slang)
BFN	British Forces Network, Bye For Now (Slang)
BNF	Backus-Naur Form, Big Name Fan (Slang)
BNFSCD	But Now For Something Completely Different (Slang)
BOF	Board of Fellows, Birds Of a Feather (Slang)
BOT	Begin Of Tape/Table/Transaction, Back On Topic (Slang)
BRB	Be Right Back (Slang)
BTAIM	Be That As It May (Slang)
BTC	Bit Test and Complement, Biting The Carpet (Slang)
BTHOM	Beats The Hell Outta Me (Slang)
BTIC	But Then, I'am Crazy (Slang)
BTK	Back To Keyboard (Slang)

BTSOOM	Beats The Shit Out Of Me (Slang)
BTW	By The Way (Slang)
CU	See You (Slang)
CUL	See You Later (Slang)
DAU	Dümmster Anzunehmender User (Slang), Digital Announcement Unit
DHRVVF	Ducking, Hiding and Running Very Very Fast (Slang)
DIY	Do It Yourself (Slang)
DSH	Desparately Seeking Help (Slang)
DWIM	Do What I Mean (Slang)
EMFBI	Excuse Me For Butting In (Slang)
ETOL	Evil Twin On Line (Slang)
F2F	Face to Face (Slang)
FAFWOA	For A Friend Without Access (Slang)
FB	Fine Business (Slang)
FHS	For Heaven's Sake (Slang)
FIAWOL	Fandom Is A Way Of Life (Slang)
FISH	First In, Still Here (Slang)
FITB	Fill In The Blank (Slang)
FITNR	Fixed In The Next Release (Slang)
FOAF	Friend Of A Friend (Slang)
FTASB	Faster Than A Speeding Bullet (Slang)
FTL	Faster Than Light (Slang)
FUBAR	Fouled Up Beyond All Repair/Recognition (Slang)
FUBB	Fouled Up Beyond Belief (Slang)
FUBS	Fido Used Book Squad (Slang)
FUD	(spreading) Fear, Uncertainty, and Disinformation (Slang),
FWIW	For What It's Worth (Slang)
FYA	For Your Amusement (Slang)
GAFIA	Get Away From It All (Slang)
GAL	Gate/Generic Array Logic, Get A Life (Slang)
GIGO	Garbage In, Garbage/Gospel Out (Slang)
GIWIST	Gee I Wish I'd Said That (Slang)
GR+D	Grinning, Running + Ducking (Slang)
HAK	Hugs and Kisses (Slang)
HHOK	Ha Ha Only Kidding (Slang)
HHOS	Ha Ha Only Serious (Slang)
HTH	Hope This/That Helps (Slang)
IAC	In Any Case (Slang), Interapplication Communication (Apple)
IAE	ISDN Anschalteinheit/Anschlusseinheit, In Any Event (Slang)
IANAL	I Am Not A Lawyer (Slang)
IBTD	I Beg To Differ (Slang)
IC	I see (Slang), Incoming Call, Input/Integrated Circuit,
ICOCBW	I Could, Of Course, Be Wrong (Slang)
IIRC	If I Remember Correctly (Slang)
IIWM	If It Were Me/Mine (Slang)
ILLAB	Ich liege lachend am Boden (Slang, vgl. ROTFL)
IMAO	In My Arrogant Opinion (Slang)
IMCO	In My Considered Opinion (Slang)
IME	In My Experience (Slang)

IMHO	In My Humble Opinion (Slang)
IMNSCO	In My Not So Considered Opinion (Slang)
IMNSHO	In My Not So Humble Opinion (Slang)
IMO	In My Opinion (Slang)
IMOBO	In My Own Biased Opinion (Slang)
INPO	In No Particular Order (Slang)
IOW	In Other Words (Slang)
IRL	Inter Repeater Link, In Real Life (Slang)
ISTM	It Seems To Me (Slang)
ISTR	I Seem To Remember (Slang)
IWBNI	It Would Be Nice If (Slang)
IYSWIM	If You See What I Mean (Slang)
ISWYM	I See What You Mean (Slang)
JSNM	Just Stark Naked Magic (Slang)
KIBO	Knowledge In, Bullshit Out (Slang)
LLTA	Lots and Lots of Thundering Applause (Slang)
LOL	Laughing Out Loud (Slang)
LOL	Lots Of Luck (Slang)
MHOTY	My Hat's Off To You (Slang)
MNRE	Manual Not Read Error (Slang)
MOTAS	Member Of The Appropriate Sex (Slang)
MTFBWY	May The Force Be With You (Slang)
MYOB	Mind Your Own Business (Slang)
NCNCNC	No Coffee, No Chocolate, No Computer (Slang)
NFI	No Frigging Idea (Slang)
NIMBY	Not In My Backyard (Slang)
NLA	Not Long Ago (Slang)
NRN	Netware Remote Node, No Reply Necessary (Slang)
NTTAWWT	Not That There's Anything Wrong With That (Slang)
OATUS	On A Totally Unrelated Subject (Slang)
OAUS	On An Unrelated Subject (Slang)
OBTW	Oh, By The Way (Slang)
OBO	Or Best Offer (Slang)
OIC	Oh, I See (Slang)
ONNA	Oh No, Not Again (Slang)
ONNTA	Oh No, Not This Again (Slang)
OOP	Object Oriented Pleasure/Programming, Out Of Print (Slang)
OOTB	Out Of The Box (Slang)
OOTC	Obligatory On-topic Comment (Slang)
OT	Object Technology, Open Transport, Off Topic (Slang)
OTOH	On The Other Hand (Slang)
OTTH	On The Third Hand (Slang)
PDQ	Pretty Darned Quick (Slang)
PFM	Postscript Font Metric, Pure Fantastic Magic (Slang)
PITA	Pine In The Ass (Slang)
PMFJIB	Pardon Me For Jumping In But (Slang)
POV	Point Of View (Slang), Persistence Of Vision
PTO	Public Telecommunication Network Operator, Please Turn Over (Slang)
RAEBNC	Read And Enjoyed, But No Comment (Slang)

RL	Remote Loopback, Real Life (Slang)
ROFL	Rolling On the Floor Laughing (Slang)
ROFLBTC	Rolling On the Floot Biting The Carpet (Slang)
ROFLMAO	Rolling On The Floor Laughing My Ass Off (Slang)
ROTFL	Roll/Rolling On The Floor Laughing (Slang)
RSN	Real Soon Now (Slang)
RTFAQ	Read The FAQ list (Slang)
RTFB	Read The Funny Binary (Slang)
RTFF	Read The Fantastic FAQ (Slang)
RTFM	Read The Fine/Fantastic/Funny ... Manual (Slang)
RTFS	Read The Funny Source (Slang)
RTM	Read The Manual (Slang)
SCNR	Sorry, Could Not Resist (Slang)
SEP	Separation, Somebody else's problem (Slang)
SFMJI	Sorry For My Jumping In (Slang)
SIASL	Stranger In A Strange Land (Slang)
SIMCA	Sitting In My Chair Amused (Slang)
SITD	Still In The Dark (Slang)
SNAFU	Situation Normal All Fed/Fucked Up (Slang)
TAFN	That's All For Now (Slang)
TANJ	There Ain't No Justice (Slang)
TANSTAAFL	There Ain't No Such Things As A Free Lunch (Slang)
TBH	To Be Honest (Slang)
TGAL	Think Globally, Act Locally (Slang)
THWLAIAS	The hour was late, and I am senile (Slang)
TIA	Thanks In Advance (Slang)
TIC	Tongue In Cheek (Slang)
TINAR	This Is Not A Review (Slang)
TINWIS	That Is Not What I Said (Slang)
TNX	Thanks (Slang)
TPTB	The Powers That Be (Slang)
TRDMC	Tears Running Down My Cheers (Slang)
TTBOMK	To The Best Of My Knowledge (Slang)
TTFN	Ta Ta For Now (Slang)
TTYL	Type/Talk To You Later (Slang)
TYCLO	Turn Your Caps Lock Off (Slang)
TYVM	Thank You Very Much (Slang)
UL	Urban Legend (Slang)
UTSL	Use The Source, Luke (Slang)
VL	Virtual Life (Slang)
WAMKSAM	Why Are My Kids Staring At Me? (Slang)
WDYMBT	What Do You Mean By That (Slang)
WOMBAT	Waste Of Money, Brains, And Time (Slang)
WRT	With Respect To (Slang)
WT	Write Through, Without Thinking (Slang)
WTTM	Without Thinking Too Much (Slang)
YMMV	Your Mileage May Vary (Slang)

I Formelbeispiele LaTeX

I.1 Gelatexte Formeln

$$c = \sqrt{a^2 + b^2} \tag{I.1}$$

$$\sqrt[3]{1+x} \approx 1 + \frac{x}{3} \qquad \text{für} \quad x \ll 1 \tag{I.2}$$

$$r = \sqrt[3]{\frac{3}{4\pi} V} \tag{I.3}$$

$$\lim_{x \to 0} \frac{\sin x}{x} = 1 \tag{I.4}$$

$$a = \frac{F_0}{k} \frac{1}{\sqrt{(1 - \frac{\Omega^2}{\omega_0^2})^2 + (\frac{\Omega_c}{k})^2}} \tag{I.5}$$

$$m\ddot{x} + c\dot{x} + kx = \sum_k F_k \cos \Omega_k t \tag{I.6}$$

$$\Theta \frac{d^2\varphi}{dt^2} + k^* \frac{d\varphi}{dt} + D^* \varphi = |\vec{D}| \tag{I.7}$$

$$\bar{Y} \approx f(\bar{x}) + \frac{1}{2} \frac{N-1}{N} f''(\bar{x})\, s_x^2 \tag{I.8}$$

$$\vec{F_\Gamma} = -\frac{, mM}{r^2} \vec{e_r} = -\frac{, mM}{r^3} \vec{r} \tag{I.9}$$

$$\begin{aligned} \text{Arbeit} &= \lim_{\Delta r_i \to 0} \sum \vec{F}_i \Delta \vec{r}_i \\ &= \int_{\vec{r}_0}^{\vec{r}(t)} \vec{F}(\vec{r})\, d\vec{r} \end{aligned} \tag{I.10}$$

$$\prod_{j \geq 0} \left(\sum_{k \geq 0} a_{jk} z^k \right) = \sum_{n \geq 0} z^n \left(\sum_{\substack{h_0, k_1 \ldots \geq 0 \\ k_0 + k_1 + \cdots = 0}} a_{0k_o} a_{1k_1} \ldots \right) \tag{I.11}$$

$$\vec{x} = \begin{pmatrix} x - x_s \\ y - y_s \\ z - z_s \end{pmatrix} \tag{I.12}$$

$$\mathbf{\Psi} = \begin{pmatrix} \binom{ab}{cd} & \frac{e+f}{g-h} \\ \Re z & \left|\begin{matrix} ij \\ kl \end{matrix}\right| \end{pmatrix} \tag{I.13}$$[1]

$$dE_\omega = V\frac{\hbar}{\pi^2 c^3}\omega^3 \cdot e^{-\frac{\hbar\omega}{T}} \cdot d\omega \tag{I.14}$$

$$\oint \vec{E}\, d\vec{s} = -\frac{\partial}{\partial t}\int \vec{B}\, d\vec{A} \tag{I.15}$$

$$\frac{1}{2\pi j}\int_{x-j\infty}^{x+j\infty} e^{ts}\, f(s)\, ds = \begin{cases} 0 & \text{für} \quad t<0 \\ F(t) & \text{für} \quad t>0 \end{cases} \tag{I.16}$$

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1} \tag{I.17}$$

$$\boxed{\forall x \in \mathrm{R}: \qquad x^2 \geq 0} \tag{I.18}$$

$$\forall x, y, z \in \mathrm{M}: \qquad (xRy \wedge xRz) \Rightarrow y = z \tag{I.19}$$

$$A \cdot B = \overline{\bar{A} + \bar{B}} \tag{I.20}$$

$$\rho \cdot \bar{v}_k \cdot \frac{\partial \bar{v}_j}{\partial x_k} = -\frac{\partial \bar{p}}{\partial x_j} + \frac{\partial}{\partial x_k}\left(\mu\frac{\partial \bar{v}_j}{\partial x_k} - \rho\,\overline{v_k' v_j'}\right) \tag{I.21}$$

$$r' = \frac{\overline{v_1' v_2'}}{\sqrt{\bar{v}_1'^2}\sqrt{\bar{v}_2'^2}} \tag{I.22}$$

$$\tau_{tur} = \rho l^2 \left|\frac{\partial \bar{v}_1}{\partial x_2}\right| \frac{\partial \bar{v}_1}{\partial x_2} \tag{I.23}$$

$$\begin{aligned} \ddot{R} &= \frac{1}{\Psi}(V_r - \dot{R}) + R\dot{\Phi}^2 \\ \ddot{\Phi} &= [\frac{1}{\Psi}(V_\varphi - R\dot{\Phi}) - 2\dot{R}\dot{\Phi}]\frac{1}{R} \\ \ddot{Z} &= \frac{1}{\Psi}(V_Z - \dot{Z}) \end{aligned} \tag{I.24}$$

[1] Fette Griechen gibt es nur als Großbuchstaben. Die Erzeugung dieser Fußnote war übrigens nicht einfach.

$$\hat{\chi}^2 = \frac{n(n-1)}{B(n-B)} \sum_{i=1}^{k} \frac{(B_i - E_i)^2}{n_i} > \chi^2_{k-1;\alpha} \tag{I.25}$$

$$V(r,\vartheta,\varphi) = \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} q_{l,m} \frac{Y_{l,m}\vartheta,\varphi)}{r^{l+1}} \tag{I.26}$$

$$\vec{q} = \begin{pmatrix} q_{0,0} \\ q_{1,1} \\ q_{1,0} \\ q_{1,-1} \\ q_{2,2} \\ q_{2,1} \\ q_{2,0} \\ q_{2,-1} \\ q_{2,-2} \end{pmatrix} \tag{I.27}$$

$$q'_{l',m'} = \sum_{o=0}^{\infty} \sum_{p=-o}^{o} n_{o,p}\; q_{o,p}\; (\nabla)_{l'+o,m'-p;o,p}\; \frac{Y_{l'+o,m'-p}(\vartheta_a,\varphi_a)}{a^{l'+o+1}} \tag{I.28}$$

$$q_{l,m} = -q'_{l,m} R^{2l+1} \frac{l(\epsilon_i - \epsilon_a)}{l(\epsilon_a + \epsilon_i) + \epsilon_a} \tag{I.29}$$

$$\vec{q'}_{1,1} = \vec{q'}_{1,0} + Ind_{2,1}\; \vec{q'}_{2,0} \tag{I.30}$$

$$\vec{q'}_{2,1} = \vec{q'}_{2,0} + Ind_{1,2}\; \vec{q'}_{1,0} \tag{I.31}$$

$$\begin{aligned} &\nabla_{\pm 1} \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} q_{l,m}\; \frac{Y_{l,m}(\vartheta,\varphi)}{r^{l+1}} \\ &= \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} (\tilde{\nabla_{\pm 1}})_{l,m}\, q_{l,m}\, \frac{Y_{l+1,m\pm 1}(\vartheta,\varphi)}{r^{l+2}} \end{aligned} \tag{I.32}$$

$$\underbrace{a + \overbrace{b + \cdots + y}^{123} + z}_{\alpha\beta\gamma}$$

Lange Formeln muß man selbst in Zeilen auflösen:

$$\begin{aligned} w + x + y + z = \\ & a + b + c + d + e + f + \\ & g + h + i + j + k + l \end{aligned} \tag{I.33}$$

$$N_{+1} = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
(\tilde{\nabla}_+)_{0,0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & (\tilde{\nabla}_+)_{1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & (\tilde{\nabla}_+)_{1,0} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & (\tilde{\nabla}_+)_{1,-1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & (\tilde{\nabla}_+)_{2,2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_+)_{2,1} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_+)_{2,0} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_+)_{2,-1} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (\tilde{\nabla}_+)_{2,-2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

I.2 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}
\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}+Ind_{2,1} \; \vec{q'}_{2,0} \\[0.3cm]
\vec{q'}_{2,1}=\vec{q'}_{2,0}+Ind_{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*}
```

J ISO 3166 Ländercodes

Bei den Namen von Computern im Internet ist es außerhalb der USA üblich, als letzten Teil den Ländercode nach ISO 3166 anzugeben. Dies ist jedoch nicht zwingend, es gibt auch Bezeichnungen, die kein Land sondern eine Organisation oder dergleichen angeben und daher nicht von ISO 3166 festgelegt werden. Die vollständige Tabelle findet sich unter `ftp://ftp.ripe.net/iso3166-countrycodes`.

Land	A 2	A 3	Nummer
ALBANIA	AL	ALB	008
ALGERIA	DZ	DZA	012
ANDORRA	AD	AND	020
ANTARCTICA	AQ	ATA	010
ARGENTINA	AR	ARG	032
AUSTRALIA	AU	AUS	036
AUSTRIA	AT	AUT	040
BELARUS	BY	BLR	112
BELGIUM	BE	BEL	056
BOSNIA AND HERZEGOWINA	BA	BIH	070
BRAZIL	BR	BRA	076
BULGARIA	BG	BGR	100
CANADA	CA	CAN	124
CHILE	CL	CHL	152
CHINA	CN	CHN	156
CROATIA	HR	HRV	191
CYPRUS	CY	CYP	196
CZECH REPUBLIC	CZ	CZE	203
DENMARK	DK	DNK	208
EGYPT	EG	EGY	818
ESTONIA	EE	EST	233
FAROE ISLANDS	FO	FRO	234
FINLAND	FI	FIN	246
FRANCE	FR	FRA	250
GEORGIA	GE	GEO	268
GERMANY	DE	DEU	276
GIBRALTAR	GI	GIB	292
GREECE	GR	GRC	300
GREENLAND	GL	GRL	304
HUNGARY	HU	HUN	348
ICELAND	IS	ISL	352
INDIA	IN	IND	356
INDONESIA	ID	IDN	360
IRELAND	IE	IRL	372

ISRAEL	IL	ISR	376
ITALY	IT	ITA	380
JAPAN	JP	JPN	392
KOREA, REPUBLIC OF	KR	KOR	410
LATVIA	LV	LVA	428
LEBANON	LB	LBN	422
LIECHTENSTEIN	LI	LIE	438
LITHUANIA	LT	LTU	440
LUXEMBOURG	LU	LUX	442
MACEDONIA, REPUBLIC OF	MK	MKD	807
MALAYSIA	MY	MYS	458
MALTA	MT	MLT	470
MEXICO	MX	MEX	484
MONACO	MC	MCO	492
MOROCCO	MA	MAR	504
NETHERLANDS	NL	NLD	528
NEW ZEALAND	NZ	NZL	554
NORWAY	NO	NOR	578
PAKISTAN	PK	PAK	586
POLAND	PL	POL	616
PORTUGAL	PT	PRT	620
ROMANIA	RO	ROM	642
RUSSIAN FEDERATION	RU	RUS	643
SAN MARINO	SM	SMR	674
SAUDI ARABIA	SA	SAU	682
SINGAPORE	SG	SGP	702
SLOVAKIA	SK	SVK	703
SLOVENIA	SI	SVN	705
SOUTH AFRICA	ZA	ZAF	710
SPAIN	ES	ESP	724
SVALBARD AND JAN MAYEN	SJ	SJM	744
SWEDEN	SE	SWE	752
SWITZERLAND	CH	CHE	756
TAIWAN	TW	TWN	158
TUNISIA	TN	TUN	788
TURKEY	TR	TUR	792
UKRAINE	UA	UKR	804
UNITED KINGDOM	GB	GBR	826
UNITED STATES	US	USA	840
VATICAN	VA	VAT	336
YUGOSLAVIA	YU	YUG	891

K Requests For Comment (RFCs)

Das Internet wird nicht durch Normen, sondern durch RFCs (Request For Comment) beschrieben, gegenwärtig etwa 2500 an der Zahl. Wird ein RFC durch einen neueren abgelöst, bekommt dieser auch eine neue, höhere Nummer. Es gibt also keine Versionen oder Ausgaben wie bei den DIN-Normen. Sucht man eine Information, besorgt man sich einen aktuellen Index der RFCs und startet in den Titeln, bei der höchsten Nummer beginnend, eine Stichwortsuche. Einige RFCs sind zugleich FYIs (For Your Information) mit eigener Zählung. Diese enthalten einführende Informationen. Andere RFCs haben den Rang von offiziellen Internet-Protokoll-Standards mit zusätzlicher, eigener Numerierung, siehe RFC 2200 *Internet Official Protocol Standards* vom Juni 1997, wobei ein Standard mehrere RFCs umfassen kann. Schließlich sind einige RFCs zugleich BCPs (Best Current Practice), siehe RFC 1818, oder RTRs (RARE Technical Report). Eine vollständige Sammlung findet sich auf `ftp.nic.de/pub/doc/rfc/`. Die Files mit den Übersichten sind:

- Request For Comment: `rfc-index.txt`, 300 kbyte,
- For Your Information: `fyi-index.txt`, 10 kbyte,
- Internet Standard: `std-index.txt`, 10 kbyte.

Hier folgt eine Auswahl, nach der Nummer sortiert.

K.1 Ausgewählte RFCs, ohne FYIs

0001	Host Software (1969)
0681	Network Unix (1975)
0814	Name, Addresses, Ports, and Routes (1982)
0821	Simple Mail Transfer Protocol (1982)
0822	Standard for the Format of ARPA Internet Text Messages (1982)
0902	ARPA-Internet Protocol policy (1984)
0959	File Transfer Protocol (1985)
1000	The Request For Comments Reference Guide (1987)
1034	Domain names – concepts and facilities (1987)
1087	Ethics and the Internet (1989)
1094	NFS: Network File System Protocol specification (1989)
1118	Hitchhiker's Guide to the Internet (1989)
1173	Responsibilities of Host and Network Managers (1991)
1180	TCP/IP Tutorial (1991)
1208	Glossary of Networking Terms (1991)
1281	Guidelines for the secure operations of the Internet (1991)
1295	User bill of rights for entries and listing in the public directory (1992)
1296	Internet Growth (1981 – 1991) (1992)

1310 Internet standards process (1992)
1327 Mapping between X.400(1988)/ISO 10021 and RFC 822 (1992)
1331 Point-to-Point Protocol (PPP) (1992)
1336 Who's who in the Internet (1992)
1345 Character Mnemonics and Character Sets (1992)
1361 Simple Network Time Protocol (1992)
1378 PPP AppleTalk Control Protocol (1992)
1432 Recent Internet books (1993)
1436 Internet Gopher Protocol (1993)
1441 SMTP Introduction to version 2 of the Internet-standard Network Management Framework (1993)
1459 Internet Relay Chat Protocol 91993)
1460 Post Office Protocol - Version 3 (1993)
1466 Guidelines for Management of IP Address Space (1993)
1475 TP/IX: The Next Internet (1993)
1501 OS/2 User Group (1993)
1506 A Tutorial on Gatewaying between X.400 and Internet Mail (1993)
1510 The Kerberos Network Authentication Service (1993)
1511 Common Authentication Technology Overview (1993)
1591 Domain Name System Structure and Delegation (1994)
1601 Charter of the Internet Architecture Board (IAB) (1994)
1603 IETF Working Group Guidelines and Procedures (1994)
1607 A VIEW FROM THE 21ST CENTURY (1994)
1618 PPP over ISDN (1994)
1661 The Point-to-Point Protocol (PPP) (1994)
1684 Introduction to White Pages Services based on X.500 (1994)
1690 Introducing the Internet Engineering and Planning Group (IEPG) (1994)
1700 ASSIGNED NUMBERS (1994)
1704 On Internet Authentication (1994)
1738 Uniform Resource Locators (URL) (1994)
1750 Randomness Recommendations for Security (1994)
1752 The Recommendation for the IP Next Generation Protocol (1995)
1775 To Be On the Internet (1995)
1789 INETPhone: Telephone Services and Servers on Internet (1995)
1808 Relative Uniform Resource Locators (1995)
1825 Security Architecture for the Internet Protocol (1995)
1835 Architecture of the WHOIS++ service (1995)
1871 Addendum to RFC 1602 – Variance Procedure (1995)
1881 IPv6 Address Allocation Management (1995)
1882 The 12-Days of Technology Before Christmas (1995)
1898 CyberCash Credit Card Protocol Version 0.8. (1996)
1912 Common DNS Operational and Configuration Errors (1996)
1913 Architecture of the Whois++ Index Service (1996)
1918 Address Allocation for Private Internets (1996)
1924 A Compact Representation of IPv6 Addresses (1996)
1925 The Twelve Networking Truths (1996)
1928 SOCKS Protocol Version 5. (1996)
1935 What is the Internet, Anyway? (1996)

1938 A One-Time Password System (1996)
1939 Post Office Protocol - Version 3. (1996)
1945 Hypertext Transfer Protocol – HTTP/1.0. (1996)
1952 GZIP file format specification version 4.3. (1996)
1955 New Scheme for Internet Routing and Addressing (ENCAPS) for IPNG (1996)
1957 Some Observations on Implementations of the Post Office Protocol (POP3) (1996)
1958 Architectural Principles of the Internet (1996)
1963 PPP Serial Data Transport Protocol (SDTP) (1996)
1968 The PPP Encryption Control Protocol (ECP) (1996)
1972 A Method for the Transmission of IPv6 Packets over Ethernet Networks (1996)
1984 IAB and IESG Statement on Cryptographic Technology and the Internet (1996)
2014 IRTF Research Group Guidelines and Procedures (1996)
2015 MIME Security with Pretty Good Privacy (PGP) (1996)
2026 The Internet Standards Process – Revision 3. (1996)
2030 Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI (1996)
2045 Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies (1996)
2046 Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types (1996)
2047 MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text (1996)
2048 Multipurpose Internet Mail Extension (MIME) Part Four: Registration Procedures (1996)
2049 Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples (1996)
2068 Hypertext Transfer Protocol – HTTP/1.1. (1997)
2070 Internationalization of the Hypertext Markup Language (1997)
2083 PNG (Portable Network Graphics) Specification (1997)
2084 Considerations for Web Transaction Security (1997)
2100 The Naming of Hosts (1997)
2110 MIME E-mail Encapsulation of Aggregate Documents, such as HTML (MHTML) (1997)
2111 Content-ID and Message-ID Uniform Resource Locators (1997)
2112 The MIME Multipart/Related Content-type (1997)
2133 Basic Socket Interface Extensions for IPv6 (1997)
2134 Articles of Incorporation of Internet Society (1997)
2135 Internet Society By-Laws (1997)
2145 Use and Interpretation of HTTP Version Numbers (1997)
2146 U.S. Government Internet Domain Names (1997)
2147 TCP and UDP over IPv6 Jumbograms (1997)
2153 PPP Vendor Extensions (1997)
2167 Referral Whois (RWhois) Protocol V1.5. (1997)
2168 Resolution of Uniform Resource Identifiers using the Domain Name System (1997)

2180 IMAP4 Multi-Accessed Mailbox Practice (1997)
2182 Selection and Operation of Secondary DNS Servers (1997)
2185 Routing Aspects of IPv6 Transition (1997)
2186 Internet Cache Protocol (ICP), version 2 (1997)
2187 Application of Internet Cache Protocol (ICP) (1997)
2192 IMAP URL Scheme (1997)
2200 INTERNET OFFICIAL PROTOCOL STANDARDS (1997)
2202 Test Cases for HMAC-MD5 and HMAC-SHA-1 (1997)
2212 Specification of Guaranteed Quality of Service (1997)
2222 Simple Authentication and Security Layer (SASL) (1997)
2223 Instructions to RFC Authors (1997)
2228 FTP Security Extensions (1997)
2231 MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations (1997)
2237 Japanese Character Encoding for Internet Messages (1997)
2245 Anonymous SASL Mechanism (1997)

K.2 Alle FYIs

1150 FYI on FYI: Introduction to the FYI notes (FYI 1) (1990)
1470 FYI on a Network Management Tool Catalog: Tools for monitoring and debugging TCP/IP internets and interconnected devices (FYI 2) (1993)
1175 FYI on Where to Start: A bibliography of internetworking information (FYI 3) (1991)
1594 FYI on Questions and Answers - Answers to Commonly asked New Internet User Questions (1994) (FYI 4)
1178 Choosing a name for your computer (FYI 5) (1991)
1198 FYI on the X window system (FYI 6) (1991)
1207 FYI on Questions and Answers: Answers to commonly asked experienced Internet user questions (FYI 7) (1991)
2196 Site Security Handbook (1997) (FYI 8)
1336 Who's Who in the Internet: Biographies of IAB, IESG and IRSG Members (1992) (FYI 9)
1402 There is Gold in them thar Networks! or Searching for Treasure in all the Wrong Places (FYI 10) (1993)
2116 X.500 Implementations Catalog-96 (1997) (FYI 11)
1302 Building a Network Information Services Infrastructure (1992) (FYI 12)
1308 Executive Introduction to Directory Services Using the X.500 Protocol (1992) (FYI 13)
1309 Technical Overview of Directory Services Using the X.500 Protocol (1992) (FYI 14)
1355 Privacy and Accuracy Issues in Network Information Center Databases (1992) (FYI 15)
1359 Connecting to the Internet – What Connecting Institutions Should Anticipate (1992) (FYI 16)

1718 The Tao of IETF - A Guide for New Attendees of the Internet Engineering Task Force (1994) (FYI 17)
1983 Internet Users' Glossary (1996) (FYI 18)
1463 FYI on Introducing the Internet – A Short Bibliography of Introductory Internetworking Readings (FYI 19) (1993)
1462 FYI on What is the Internet (FYI 20) (1993)
1491 A Survey of Advanced Usages of X.500 (FYI 21) (1993)
1941 Frequently Asked Questions for Schools (1996) (FYI 22)
1580 Guide to Network Resource Tool (1994) (FYI 23)
1635 How to Use Anonymous FTP (1994) (FYI 24)
1689 A Status Report on Networked Information Retrieval: Tools and Groups (1994) (FYI 25)
1709 K-12 Internetworking Guidelines (1994) (FYI 26)
1713 Tools for DNS debugging (1994) (FYI 27)
1855 Netiquette Guidelines (1995) (FYI 28)
2007 Catalogue of Network Training Materials (1996) (FYI 29)
2151 A Primer On Internet and TCP/IP Tools and Utilities (1997) (FYI 30)
2150 Humanities and Arts: Sharing Center Stage on the Internet (1997) (FYI 31)
2235 Hobbes' Internet Timeline (1997) (FYI 32)

L Frequently Asked Questions (FAQs)

In vielen Newsgruppen tauchen immer wieder dieselben Fragen auf. Irgendwann erbarmt sich ein Leser und sammelt sie samt den zugehörigen Antworten unter der Überschrift *Frequently Asked Questions*, abgekürzt FAQ. Diese FAQs (man beachte: der Plural eines Plurals) sind eine wertvolle Informationsquelle. Die Spektren der Themen und der Qualität sind so breit wie das Netz. Innerhalb der Netnews enthalten die FAQs naturgemäß nur Text, manche werden jedoch auch parallel dazu im WWW angeboten und können dort Grafik verwenden. Sie sind zu finden:

- in der jeweiligen Newsgruppe,
- in der Newsgruppe `news.answers` bzw. `de.answers`,
- auf `rtfm.mit.edu` in den Verzeichnissen `/pub/usenet-by-group/` bzw. `/pub/usenet-by-hierarchie/`

Um einen Überblick zu gewinnen, hole man sich per FTP von dort das File `Index-byname.gz`. Nachfolgend sind einige FAQs aufgeführt:

- Unix - Frequently Asked Questions. Siebenteilig, von Ted Timar. In `news.answers` und `comp.unix.questions`, seit 1989, daher ausgereift.

M 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) Hundefutter (0 Punkte) kleinste Dateneinheit (2 Punkte)
Festplatte	Was zum Essen, vom Partyservice (1) Schallplatte (0) Massenspeicher (2)
Menu	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) amerikanisches Fort (0) Programmiersprache (2)
Mainframe	Frachtkahn auf dem Main (0) Schiff, mit dem FRIDTJOF NANSEN zum Nordpol wollte (0) großer Computer (2)
PC	Plumpsklo (Gravitationstoilette) (1) Personal Computer (2) Power Computing Language (0)
Puffer	Was zum Essen (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 (2) Untiefe im Wattenmeer (0)
Alias	altgriechisches Epos (0) alttestamentarischer Prophet (0) Zweitname eines Kommandos (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)
C	Vitamin (1) Programmiersprache (2) Körperteil (0)
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)

IEC-Bus	Schnittstelle (2) Intercity-Bus (0) Internationale Bus-Gesellschaft (0)
Wysiwig	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)
Datex	Klebstoff (0) Datendienst der Post (2) Kommando zum Löschen von Daten (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 (2) Rohkost-Liebhaber (0)
Schleife	Kleidungsstück (1) Schlitterbahn (1) Kontrollstruktur 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)
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 169 Punkte: Überlassen Sie das Rechnen künftig dem Computer.
- 84 bis 169 Punkte: Mit etwas Fleiß wird aus Ihnen ein EDV-Experte.
- 17 bis 83 Punkte: Machen Sie eine möglichst steile Karriere außerhalb der EDV und suchen Sie sich fähige Mitarbeiter.
- unter 17 Punkten: Vielleicht hatten Sie schlechte Lehrer?

N Zeittafel

Diese Übersicht ist auf Karlsruher Verhältnisse zugeschnitten. Ausführlichere Angaben sind den im Anhang O *Literatur* in Abschnitt *Geschichte* aufgeführten Werken zu entnehmen. Die meisten Errungenschaften entstanden nicht zu einem Zeitpunkt, sondern entwickelten sich über manchmal lange Zeitspannen, so daß vor viele Jahreszahlen *um etwa* zu setzen ist. Das Deutsche Museum in München zeigt in den Abteilungen *Informatik* und *Telekommunikation* einige der hier genannten Maschinen.

ca. 10^8 v. Chr.	Der beliebte Tyrannosaurus hatte zwei Finger an jeder Hand und rechnete vermutlich im Dualsystem, wenn überhaupt.
ca. 2000 v. Chr.	Die Babylonier verwenden für besondere Aufgaben ein gemischtes Stellenwertsystem zur Basis 60.
ca. 400 v. Chr.	In China werden Zählstäbchen zum Rechnen verwendet.
ca. 20	In der Bergpredigt wird das Binärsystem erwähnt (Matth. 5, 37). Die Römer schieben Rechensteinchen (calculi).
600	Die Inder entwickeln das heute übliche reine Stellenwertsystem, die Null ist jedoch älter. Etwa gleichzeitig entwicklen die Mayas in Mittelamerika ein Stellenwertsystem zur Basis 20.
1200	Leonardo von Pisa, genannt Fibonacci, setzt sich für die Einführung des indisch-arabischen Systems im Abendland ein.
1550	Die europäischen Rechenmeister verwenden sowohl die römische wie die indisch-arabische Schreibweise.
1617	John Napier erfindet die Rechenknochen (Napier's Bones).
1623	Erste mechanische Rechenmaschine mit Zehnerübertragung und Multiplikation, von Wilhelm Schickard, Tübingen.
1642	Rechenmaschine von Blaise Pascal, Paris für kaufmännische Rechnungen seines Vaters.
1674	Gottfried Wilhelm Leibniz baut eine mechanische Rechenmaschine für die vier Grundrechenarten und befaßt sich mit der dualen Darstellung von Zahlen. In der Folgezeit technische Verbesserungen an vielen Stellen in Europa.
1801	Joseph Marie Jacquard erfindet die Lochkarte und steuert Webstühle damit.
1821	Charles Babbage stellt der Royal Astronomical Society eine programmierbare mechanische Rechenmaschine vor, die jedoch keinen wirtschaftlichen Erfolg hat. Er denkt auch an das Spielen von Schach oder Tic-tac-toe auf Maschinen.
1840	Samuel Morse entwickelt einen aus zwei Zeichen plus Pausen bestehenden Telegrafencode, der die Buchstaben entsprechend ihrer Häufigkeit codiert.
1847	George Boole entwickelt die symbolische Logik.

1890 Herman Hollerith erfindet die Lochkartenmaschine und setzt sie bei einer Volkszählung in den USA ein. Das ist der Anfang von IBM.

1894 Otto Luegers *Lexikon der gesamten Technik* führt unter dem Stichwort *Elektrizität* als Halbleiter Aether, Alkohol, Holz und Papier auf.

1896 Gründung der Tabulating Machine Company, der späteren IBM.

1910 Gründung der Deutschen Hollerith Maschinen GmbH, Berlin, der Vorläuferin der IBM Deutschland.

1924 Aus der Tabulating Machine Company von Herman Hollerith, später in Computing-Tabulating-Recording Company umbenannt, wird die International Business Machines (IBM).
Eugen Nesper schreibt in seinem Buch *Der Radio-Amateur*, jeder schlechte Kontakt habe gleichrichtende Eigenschaften, ein Golddraht auf einem Siliziumkristall sei aber besonders gut als Kristalldetektor geeignet; eine heiße Spur.

1937 Alan Turing veröffentlicht sein Gedankenmodell eines Computers.

1938 Konrad Zuse baut den programmgesteuerten Rechner Z 1. Sein wichtigstes Werkzeug dabei ist die Laubsäge.

1939 Gründung der Firma Hewlett-Packard, Palo Alto, Kalifornien durch William Hewlett und David Packard. Ihr erstes Produkt ist ein Oszillator für Tonfrequenzen.

1941 Konrad Zuse baut die Z3.

1942 Die Purdue University beginnt mit der Halbleiterforschung und untersucht Germaniumkristalle.

1944 Die Zuse Z4 wird fertig (2200 Relais, mechanischer Speicher). Sie arbeitet von 1950 bis 1960 in der Schweiz.

1945 Konrad Zuse entwickelt den Plankalkül, die erste höhere Programmiersprache. William Bradford Shockley startet ein Forschungsprojekt zur Halbleiterphysik in den Bell-Labs.

1946 John von Neumann veröffentlicht sein Computerkonzept.
J. Presper Eckert und John W. Mauchly bauen in den USA die ENIAC (Electronic Numerical Integrator and Calculator), die erste elektronische Rechenmaschine. Die ENIAC arbeitet dezimal, enthält 18000 Vakuumröhren, wiegt 30 t, ist 5,5 m hoch und 24 m lang, braucht für eine Addition 0,2 ms, ist an der Entwicklung der Wasserstoffbombe beteiligt und arbeitet bis 1955. Sie ist der Urahne der UNIVAC.

1948 Claude E. Shannon begründet die Informationstheorie.
John Bardeen, Walter Houser Brattain und William Bradford Shockley entwickeln in den Bell-Labs den Transistor, der 10 Jahre später die Vakuumröhre ablöst.

1949 Erster Schachcomputer: Manchester MADM.

1952 IBM bringt ihre erste elektronische Datenverarbeitungsanlage, die IBM 701, heraus. Vorschläge für integrierte Schalt-

kreise, nicht verwirklicht.

1954 Remington-Rand bringt die erste UNIVAC heraus, IBM die 650. Silizium beginnt, das Germanium zu verdrängen.

1955 IBM entwickelt die erste höhere Programmiersprache: FORTRAN (Formula Translator) und verwendet Transistoren.

1956 Konrad Zuse baut die Z22. Sie kommt 1958 auf den Markt. Bis 1961 werden 50 Stück verkauft.
Bardeen, Brattain und Shockley erhalten den Nobelpreis für Physik.
IBM stellt die erste Festplatte vor, 5 MByte, groß wie ein Schrank.

1957 Die IBM 709 braucht für eine Multiplikation 0,12 ms.
Weltweit arbeiten rund 1300 Computer.
Seminar von Prof. Johannes Weissinger über *Programmgesteuerte Rechenmaschinen* im SS 1957 der TH Karlsruhe.
Karl Steinbuch prägt den Begriff *Informatik*.
Erster Satellit (Sputnik, Sowjetunion) kreist um die Erde.

1958 Die TH Karlsruhe erhält eine Zuse Z22. Die Maschine verwendet 400 Vakuumröhren und wiegt 1 t. Der Arbeitsspeicher faßt 16 Wörter zu 38 Bits, d. h. 76 Byte. Der Massenspeicher, eine Magnettrommel, faßt rund 40 KByte. Eine Gleitkommaoperation dauert 70 ms. Das System versteht nur Maschinensprache (Freiburger Code). Es läuft bis 1972.
Im SS 1958 hält Priv.-Doz. Karl Nickel (Institut für Angew. Mathematik) eine Vorlesung *Programmieren mathematischer und technischer Probleme für die elektronische Rechenmaschine Z22*.
Die Programmiersprache ALGOL 58 kommt heraus.
Bei Texas Instuments baut Jack St. Clair Kilby den ersten IC.

1959 Im SS 1959 hält Priv.-Doz. Karl Nickel erstmals die Vorlesung *Programmieren I*, im WS 1959/60 die Vorlesung *Programmieren II*. Erstes Werk von Hewlett-Packard in Deutschland. Siemens baut die Siemens 2002.

1960 Programmieren steht noch in keinem Studienplan, sondern ist freiwillig. Die Karlsruher Z22 läuft Tag und Nacht. Die Programmiersprache COBOL wird veröffentlicht. Ein Computerspiel namens *Spacewar* läuft auf einer PDP-1 im MIT.
Digital Equipment (DEC) bring die PDP 1 heraus.
Al Shugart etnwickelt ein Verfahren zur Aufzeichnung von Daten auf einer magnetisch beschichteten Scheibe.

1961 Die TH Karlsruhe erhält eine Zuse Z23, die mit 2400 Transistoren arbeitet. Ihr Hauptspeicher faßt 240 Wörter zu 40 Bits. Eine Gleitkommaoperation dauert 15 ms. Außer Maschinensprache versteht sie ALGOL. Weltweit arbeiten etwa 7300 Computer.

1962 Die TH Karlsruhe erhält eine SEL ER 56, die bis 1968 läuft.
An der Purdue University wird die erste Fakultät für Informatik (Department of Computer Science) gegründet. Texas Instruments

und Fairchild nehmen die Serienproduktion von ICs (Chips) auf.

1963 Weltweit arbeiten etwa 16.500 Computer.
Erster geostationärer Satellit (Syncom).

1964 Die Programmiersprache BASIC erscheint. In den USA wird der Begriff *Computer Science* geprägt.
IBM legt das Byte zu 8 Bits fest (IBM 360).
Ein Chip enthält auf 0,5 cm^2 10 Transistoren.

1966 Die TH Karlsruhe erhält eine Electrologica X 8, die bis 1973 betrieben wird. Gründung des Karlsruher Rechenzentrums.
Hewlett-Packard steigt in die Computerei ein (HP 2116 A).

1967 Erster elektronischer Taschenrechner (Texas Instruments).
Beim Bundesministerium für wissenschaftliche Forschung wird ein Fachbeirat für Datenverarbeitung gebildet.

1968 Die Programmiersprache PASCAL kommt heraus.
Die Firma Intel wird gegründet.
Hewlett-Packard baut den ersten wissenschaftlichen programmierbaren Tischrechner (HP 9100 A).

1969 In Karlsruhe wird das Institut für Informatik gegründet, erster Direktor Karl Nickel. Im WS 1969/70 beginnt in Karlsruhe die Informatik als Vollstudium mit 91 Erstsemestern.
Gründung der Gesellschaft für Informatik (GI) in Bonn.
In den Bell Labs UNIX in Assembler auf einer DEC PDP 7.
Beginn des ARPANET-Projektes und der TCP/IP-Protokolle, erste Teilnehmer U. of California at Los Angeles, Stanford Research Institute, U. of California at Santa Barbara und U. of Utah.
RFC 0001: Host Software, von Steve Crocker.

1970 Die Universität Karlsruhe erhält eine UNIVAC 1108, die bis 1987 läuft und damit den Rekord an Betriebsjahren hält. Die Karlsruher Fakultät für Informatik wird gegründet.

1971 UNIX auf C umgeschrieben, erster Mikroprozessor (Intel 4004).

1972 IBM entwickelt das Konzept des virtuellen Speichers und stellt die 8-Zoll-Floppy-Disk vor. Xerox (Bob Metcalfe), DEC und Intel entwickeln den Ethernet-Standard.
Das ARPANET wird der Öffentlichkeit vorgestellt.
Ein Student namens Stephan G. Wozniak lötet sich einen Computer zusammen, der den Smoke-Test nicht übersteht.
In der Bundesrepublik arbeiten rund 8.200 Computer.
Hewlett-Packard baut den ersten wissenschaftlichen Taschenrechner (HP 35).

1973 Erste internationale Teilnehmer am ARPANET: U. College of London und Royal Radar Establishment, Norwegen.

1974 Der erste programmierbare Taschenrechner kommt auf den Markt (Hewlett-Packard 65), Preis 2500 DM.

1975 UNIX wird veröffentlicht, Beginn der BSD-Entwicklung.
Die Zeitschrift *Byte* wird gegründet.
Erste, mäßig erfolgreiche Personal Computer (Altair).

1976 Steven P. Jobs und Stephan G. Wozniak gründen

die Firma Apple und bauen den Apple I. Er kostet 666,66 Dollar.
Al Shugart stellt die 5,25-Zoll-Diskette vor.
Königin Elizabeth II. von England verschickt eine E-Mail.

1978 In der Bundesrepublik arbeiten rund 170.000 Computer.
Der Commodore PET – ein Vorläufer des C64 – kommt heraus.

1979 Faxdienst in Deutschland eingeführt.
Die Zusammenarbeit von Apple mit Rank Xerox führt zur Apple Lisa, ein Mißerfolg, aber der Wegbereiter für den Macintosh.

1980 Erster Jugendprogrammier-Wettbewerb der GI.
Sony führt die 3,5-Zoll-Diskette ein. In den Folgejahren entwickeln andere Firmen auch Disketten mit Durchmessern von 3 bis 4 Zoll.

1981 Die Universität Karlsruhe erhält eine Siemens 7881 als zentralen Rechner. IBM bringt in den USA den IBM-PC heraus mit MS-DOS (PC-DOS 1.0) als wichtigstem Betriebssystem.

1982 Die Firma SUN wird gegründet, entscheidet sich für UNIX und baut die ersten Workstations.

1983 Die Universität Karlsruhe erhält einen Vektorrechner Cyber 205 und eine Siemens 7865. Die Cyber leistet 400 Mio. Gleitkommaoperationen pro Sekunde.
IBM bringt den PC auf den deutschen Markt.
UNIX kommt als System V von AT&T in den Handel, Gründung der X/Open-Gruppe.
MS-DOS 2.0 (PC-DOS 2.0) und Novell Netware kommen heraus.

1984 Der erste Macintosh kommt auf den Markt.
Der IBM PC/AT mit Prozessor Intel 80 286 und MS-DOS 3.0 kommen heraus. Siemens steigt in UNIX ein.
Entwicklung des X Window Systems am MIT.

1985 MS-Windows 1.0, IBM 3090 und IBM Token Ring Netz.
X-Link an der Uni Karlsruhe stellt als erstes deutsches Netz eine Verbindung zum nordamerikanischen ARPA-Net her.
Hewlett-Packard bringt den ersten Laserjet-Drucker heraus.

1986 Weltweit etwa eine halbe Million UNIX-Systeme und 3000 öffentliche Datenbanken.
Mit dem Computer-Investitionsprogramm des Bundes und der Länder (CIP) kommen mehrere HP 9000/550 unter UNIX an die Universität Karlsruhe.

1987 Microsoft XENIX für den IBM PC/AT
IBM bringt die PS/2-Reihe unter MS-OS/2 heraus.
Weltweit mehr als 5 Millionen Apple Computer und etwa 100 Millionen PCs nach Vorbild von IBM.
Das MIT veröffentlicht das X Window System Version 11 (X11).
In Berkeley wird die RAID-Technologie entwickelt.

1988 Eine Siemens (Fujitsu) VP 400 ersetzt die Cyber 205.
Das Campusnetz KARLA wird durch das Glasfasernetz KLICK ausgetauscht. Das BELWUE-Netz nimmt den Betrieb auf.
Gründung der Open Software Foundation und der UNIX

International Inc. MS-DOS 4.0 für PCs.
Der Internet-Wurm namens Morris geht auf die Reise, darauf hin Gründung des Computer Emergency Response Teams (CERT).
Erstes landmobiles Satellitensystem für Datenfunk (Inmarsat-C).

1989 Im Rechenzentrum Karlsruhe löst die IBM 3090 die Siemens 7881 ab. ISDN in Deutschland eingeführt.

1990 Zunehmende Vernetzung, Anschluß an weltweite Netze. Computer-Kommunikation mittels E-Mail, Btx und Fax vom Arbeitsplatz aus. Optische Speichermedien (CD-ROM, WORM).
UNIX System V Version 4. Die mittlere Computerdichte in technisch orientierten Instituten und Familien erreicht 1 pro Mitglied.

1991 Das UNIX-System OSF/1 mit dem Mach-Kernel der Carnegie-Mellon-Universität kommt heraus.
Anfänge von LINUX, einem freien UNIX aus Finnland.
Der Vektorrechner im RZ Karlsruhe wird erweitert auf den Typ S600/20.
MS-DOS 5.0 für PCs. Anfänge von Microsoft Windows NT.
IBM, Apple und Motorola kooperieren mit dem Ziel, einen Power PC zu entwickeln.
Tim Berners-Lee entwickelt am CERN das World Wide Web.

1992 Die Universität Karlsruhe nimmt den massiv parallelen Computer MasPar 1216A mit 16000 Prozessoren in Betrieb.
Novell übernimmt von AT&T die UNIX-Aktivitäten (USL).
Eine Million Knoten im Internet.

1993 MS-DOS Version 6.0. Microsoft kündigt Windows-NT an.
DEC stellt PC mit Alpha-Prozessor vor, 150 MHz, 14.000 DM.
UNIX-Workstations konkurrieren preislich mit hochwertigen PCs.
Novell tritt das Warenzeichen UNIX an die X/Open-Gruppe ab.
Das DE-NIC kommt ans RZ der Universität Karlsruhe.

1994 Weltweit 10 Mio. installierte UNIX-Systeme prognostiziert.
Das Internet umfaßt etwa 4 Mio. Knoten und 20 Mio. Benutzer.
Erste Spam-Mail (Canter + Siegel).

1995 Die Universität Karlsruhe ermöglicht in Zusammenarbeit mit dem Oberschulamt nordbadischen Schulen den Zugang zum Internet. Ähnliche Projekte werden auch an einigen anderen Hoch- und Fachhochschulen durchgeführt.
Die mittlere Computerdichte in technisch orientierten Instituten und Familien erreicht 2 pro Mitglied.

1996 Die Massen erobern das Internet.

1997 100-Ethernet ist erschwinglich geworden, über das Gigabit-Ethernet wird geredet. In Deutschland gibt es rund 20 Mio. PCs und 1 Mio. Internetanschlüsse (Quelle: Fachverband Informationstechnik).

1998 Compaq übernimmt die Digital Equipment Corporation (DEC). Einer der Gründe für den Niedergang von DEC ist die zu späte und halbherzige Unterstützung von UNIX. Der Trend zum Drittcomputer hält an.

O Literatur

Die Auswahl ist subjektiv und enthält Texte, die wir noch lesen wollen, schon haben oder sogar schon gelesen haben. Die hier angeführte Electronic Information ist auf `ftp.ciw.uni-karlsruhe.de`, `www.ciw.uni-karlsruhe.de` und anderen verfügbar.

1. Sammlung von URLs (Bookmarks)

   W. Alex, B. Alex, A. Alex UNIX, C/C++, Internet usw.
   `http://www.ciw.uni-karlsruhe.de/technik.html`
   Zu jedem Thema des Buches finden sich dort weiterführende WWW-Seiten. Mit dieser Sammlung arbeiten wir selbst.

2. Literaturlisten

   – Newsgruppen:
   de.etc.lists (wechselnde Listen aus dem deutschsprachigen Raum)
   news.lists (internationale Listen)
   alt.books.technical
   misc.books.technical

   – RFC 1175 (FYI 3): FYI on Where to Start –
   A Bibliography of Internetworking Information
   ftp://ftp.nic.de/pub/doc/rfc/rfc-1100-1199/rfc1175.txt
   1990, 45 S., ASCII
   Empfehlungen und kurze Kommentare, Erklärungen

   – X Technical Bibliography, presented by The X Journal
   ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/xws/xbiblio.ps.gz
   1994, 22 S., Postscript
   Kurze Inhaltsangaben, teilweise kommentiert

   J. December Information Sources: The Internet and Computer-Mediated Communication
   ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/cmc.gz
   1994, 33 S., ASCII
   Hinweise, wo welche Informationen im Netz zu finden sind.

   S. Ko A Concise Guide to UNIX Books
   Netnews: misc.books.technical oder comp.unix.questions
   ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/unix/unix-books
   1993, 22 S., ASCII
   Empfehlungen und kurze Kommentare

   D. A. Lamb Software Engineering Readings
   Netnews: comp.software-eng

ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/misc/sw-engng-reading
1994, 10 S., ASCII Teilweise kommentiert

R. E. Maas MaasInfo.DocIndex
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/maasinfo-idx
1994, 20 S., ASCII
Bibliografie von rund 100 On-line-Texten zum Internet

J. Quarterman RFC 1432: Recent Internet Books
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/rfc/rfc1432.txt
1993, 15 S., ASCII
Empfehlungen und kurze Kommentare

C. Spurgeon Network Reading List: TCP/IP, Unix and Ethernet
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/reading-list.ps.gz
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/reading-list.txt.gz
1993, ca. 50 S., Postscript und ASCII
Ausführliche Kommentare und Hinweise

M. Wright Yet Another book List (YABL)
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/misc/yabl.gz
1993, ca. 100 S., ASCII
Tabellarisch, kurze Kommentare

3. Lexika, Glossare, Wörterbücher

– Newsgruppen:
news.answers
de.etc.lists
news.lists

– RFC 1392 (FYI 18): Internet Users' Glossary
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/rfc/rfc1392.txt
1993, 53 S.

– Duden Informatik
Dudenverlag, Mannheim, 1993, 800 S., 42 DM
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., 113 DM
Wörterbuch und Glossar

W. Alex Abkürzungs-Liste ABKLEX (Informatik, Telekommunikation)
ftp://ftp.ciw-karlsruhe.de/pub/misc/abklex.txt
http://www.ciw-karlsruhe.de/abklex.html

V. Anastasio Wörterbuch der Informatik Deutsch – Englisch – Französisch – Italienisch – Spanisch
VDI-Verlag, Düsseldorf, 1990, 400 S., 128 DM

A. E. Cawkell Encyclopedic Dictionary of Information Technology and Systems
Saur, München, 1993, 350 S., 190 DM

F. Krückeberg, O. Spaniol Lexikon Informatik und Kommunikationstechnik
VDI-Verlag, Düsseldorf, 1990, 693 S., 168 DM

A. Ralston, E. D. Reilly Encyclopedia of Computer Science
Chapman + Hall, London, 1993, 1558 S., 60 £
Ausführliche Erläuterungen

E. S. Raymond The New Hacker's Dictionary
The MIT Press, Cambridge, 1996, 547 S., 41 DM
Siehe auch `http://www.ciw.uni-karlsruhe.de/kopien/jargon/`
Begriffe aus dem Netz, die nicht im Duden stehen

H.-J. Schneider Lexikon der Informatik und Datenverarbeitung
Oldenbourg, München, 1991, 989 S., 128 DM
Ethernet, SQL stehen darin, Internet nicht.

4. Informatik

– Newsgruppen:
comp.* (alles, was mit Computer Science zu tun hat, mehrere hundert Untergruppen)
de.comp.* (dito, deutschsprachig)
alt.comp.*

F. L. Bauer, G. Goos Informatik 1. + 2. Teil
Springer, Berlin, 1991/92, 1. Teil 393 S., 42 DM
2. Teil 345 S., 42 DM
Umfassende Einführung, auch für Nicht-Informatiker

W. Coy Aufbau und Arbeitsweise von Rechenanlagen
Vieweg, Braunschweig, 1992, 367 S., 50 DM
Digitale Schaltungen, Rechnerarchitektur, Betriebssysteme am Beispiel von UNIX

L. Goldschlager, A. Lister Informatik
Hanser und Prentice-Hall, München, 1990, 366 S., 40 DM
Einführung, ähnlich wie Bauer + Goos

G. Goos Vorlesungen über Informatik
Springer, Berlin, 1995, Band 1 393 S., ?? DM

D. E. Knuth The Art of Computer Programming, 3 Bände
Addison-Wesley, zusammen 330 DM
Klassiker, stellenweise mathematisch, 7 Bände geplant

W. Schiffmann, R. Schmitz Technische Informatik
Springer, Berlin, 1993/94, 1. Teil Grundlagen der digitalen Elektronik, 282 S., 38 DM; 2. Teil Grundlagen der Computertechnik, 283 S., 42 DM

U. Schöning Theoretische Informatik kurz gefaßt
BI-Wissenschaftsverlag, Mannheim, 1992, 188 S., 20 DM
Automaten, Formale Sprachen, Berechenbarkeit, Komplexität

K. W. Wagner Einführung in die Theoretische Informatik
Springer, Berlin, 1994, 238 S.,
Grundlagen, Berechenbarkeit, Komplexität, BOOLEsche Funktionen,
Autoamten, Grammatiken, Formale Sprachen

H. Waldschmidt Informatik für Ingenieure
Oldenbourg, München, 1987, 258 S., 40 DM
Algorithmen und Programme, Programmierfehler, Ergänzung
zu einem Programmierkurs

5. Algorithmen, Numerische Mathematik

– Newsgruppen:
sci.math.*
zer.z-netz.wissenschaft.mathematik

G. Engeln-Müllges, F. Reutter Formelsammlung zur
Numerischen Mathematik mit C-Programmen
BI-Wissenschaftsverlag, Mannheim, 1990, 744 S., 88 DM
Algorithmen und Formeln der Numerischen Mathematik samt
C-Programmen. Auch für FORTRAN, PASCAL, BASIC und
MODULA erhältlich

E. Horowitz, S. Sahni Algorithmen
Springer, Berlin, 1981, 770 S., 116 DM

D. E. Knuth (siehe unter Informatik)

T. Ottmann, P. Widmayer Algorithmen und Datenstrukturen
BI-Wissenschafts-Verlag, Mannheim, 1993, 755 S., 74 DM

W. H. Press u. a. Numerical Recipes in C
Cambridge University Press, 1993, 994 S., 98 DM
mit Diskette, auch für FORTRAN und PASCAL erhältlich

H. R. Schwarz Numerische Mathematik
Teubner, Stuttgart, 1993, 575 S., 48 DM

R. Sedgewick Algorithmen in C
Addison-Wesley, Bonn, 1992, 742 S., 90 DM
Erklärung gebräuchlicher Algorithmen und Umsetzung in C.
Auch in Englisch und für PASCAL

R. Sedgewick Algorithmen in C++
Addison-Wesley, Bonn, 1992, 742 S., 90 DM
Wie vorstehend.

J. Stoer, R. Bulirsch Numerische Mathematik
Springer, Berlin, 1. Teil 1993, 314 S., 32 DM,
2. Teil 1990, 341 S., 32 DM

F. Stummel, K. Hainer Praktische Mathematik
Teubner, Stuttgart, 1982, 367 S., 40 DM

N. Wirth Algorithmen und Datenstrukturen
Teubner, Stuttgart, 1983, 320 S., 42 DM
Viel zu Datenstrukturen, weniger zu Algorithmen

6. Betriebssysteme

– Newsgruppen:
comp.os.*
de.comp.os.*

– Microsoft MS-DOS-Handbücher

– Microsoft MS-Windows-NT-Handbücher

– OS/2 Version 2.0 Technical Compendium (Red Books)
IBM, Boca Raton, 1992, 5 Bände, 1158 S., 100 DM
OPD software.watson.ibm.com im Verzeichnis /pub/os2/misc
auch auf ftp://ftp.uni-stuttgart.de/pub/soft/os2/info/redbooks

L. Bic, A. C. Shaw Betriebssysteme
Hanser, München, 1990, 420 S., 58 DM
Allgemeiner als Tanenbaum 1

H. M. Deitel, M. S. Kogan The Design of OS/2
Addison-Wesley, Reading, 1992, 389 S., 95 DM

A. S. Tanenbaum Operating Systems, Design and Implementation
Prentice-Hall, London, 1987, 719 S., 79 DM
Einführung in Betriebssysteme am Beispiel von UNIX

A. S. Tanenbaum Modern Operating Systems
Prentice-Hall, London, 1992, 728 S., 100 DM
Allgemeiner und moderner als vorstehendes Buch; MS-DOS, UNIX, MACH und Amoeba

H. Wettstein Systemarchitektur
Hanser, München, 1993, 514 S., 68 DM
Grundlagen, kein bestimmtes Betriebssystem

7. UNIX allgemein

– Newsgruppen:
comp.unix.*
comp.sources.unix
comp.std.unix
de.comp.os.unix
fr.comp.os.unix
alt.unix.wizards
cern.unix

M. J. Bach Design of the UNIX Operating System
Prentice-Hall, London, 1987, 512 S., 52 US-$
Filesystem und Prozesse, wenig zur Shell

S. R. Bourne Das UNIX System V (The UNIX V Environment)
Addison-Wesley, Bonn, 1988, 464 S., 62 DM
Einführung in UNIX und die Bourne-Shell

D. Gilly u. a. UNIX in a Nutshell
O'Reilly, Sebastopol, 1992, ca. 250 S., 22 DM
Nachschlagewerk zu den meisten UNIX-Kommandos

J. Gulbins, K. Obermayr UNIX
Springer, Berlin, 4. Aufl. 1995, 838 S., ?? DM
Benutzung von UNIX, ausführlich, geht in die Einzelheiten

H. Hahn A Student's Guide to UNIX
McGraw-Hill, New York, 1993, 633 S., 66 DM
Einführendes Lehrbuch, ohne C, mit Internet-Diensten

J. A. Illik (siehe unter Programmieren in C)

B. W. Kernighan, P. J. Plauger Software Tools
Addison-Wesley, Reading, 1976, 338 S., 38 US-$
Grundgedanken einiger UNIX-Werkzeuge, Programmierstil

B. W. Kernighan, R. Pike Der UNIX-Werkzeugkasten
Hanser, München, 1986, 402 S., 76 DM
Gebrauch der UNIX-Kommandos, fast nichts zum `vi(1)`

D. G. Korn, M. I. Bolsky The Kornshell, Command and Programming Language
auf deutsch: Die KornShell, Hanser, München, 1991, 98 DM
Einführung in UNIX und die Korn-Shell

M. Loukides UNIX for FORTRAN Programmers
O'Reilly, Sebastopol, 1990, 244 S., 55 DM
Kurze, allgemeine Einführung in UNIX, ausführliche Behandlung der Programmer's Workbench im Hinblick auf FORTRAN

J. Peek u. a. UNIX Power Tools
O'Reilly, Sebastopol, 1993, 1119 S., 119 DM
Viele nützliche Hinweise für den Anwender, mit CD

M. J. Rochkind Advanced UNIX Programming
Prentice-Hall, London, 1986, 224 S., 47 US-$
Beschreibung aller UNIX System Calls

A. T. Schreiner Professor Schreiners UNIX-Sprechstunde
Hanser, München, 1987, 316 S., 64 DM
Shellscripts und kurze C-Programme für verschiedene Zwecke

R. M. Stallman The GNU Manifesto
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/misc/manifest-gnu
1985, 8 S., ASCII
Ziele des GNU-Projekts

W. R. Stevens Advanced Programming in the UNIX Environment
Addison-Wesley, Reading, 1992, 744 S., 110 DM
Ähnlich wie Rochkind

S. Strobel, T. Uhl LINUX - vom PC zur Workstation
Springer, Berlin, 1994, 238 S., 38 DM

L. Wirzenius, M. Welsh LINUX Information Sheet
Netnews: comp.os.linux
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/unix/linux-info
1993, 6. S., ASCII
Anfangsinformation zu LINUX, was und woher.

8. UNIX Einzelthemen

– Newsgruppen:
comp.unix.*

A. V. Aho, B. W. Kernighan, P. J. Weinberger The AWK Programming Language
Addison-Wesley, Reading, 1988, 210 S., 58 DM
Standardwerk zum AWK

B. Anderson u. a. UNIX Communications
Sams, North College, 1991, 736 S., 73 DM
Unix-Mail, Usenet, uucp und weiteres

M. I. Bolsky The vi User's Handbook
Prentice-Hall, Englewood Cliffs, 1985, 66 S., 59 DM (!)
Alle vi-Kommandos übersichtlich, aber keine Interna

D. Cameron, B. Rosenblatt Learning GNU Emacs
O'Reilly, Sebastopol, 1991, 442 S., 21 £

F. da Cruz, C. Gianone C-Kermit
Heise, Hannover, 1994, 650 S., 90 DM
Kermit-Terminalemulation und -Fileübertragung

I. F. Darwin Checking C Programs with lint
O'Reilly, Sebastopol, 1988, 82 S., 10 £

B. Goodheart UNIX Curses Explained
Prentice-Hall, Englewood-Cliffs, 1991, 287 S., ca. 80 DM
Einzelheiten zu `curses(3)` und `terminfo(4)`

L. Lamb Learning the vi Editor
O'Reilly, Sebastopol, 1990, 192 S., 17 £

E. Nemeth, G. Snyder, S. Seebass UNIX System Administration Handbook
Prentice-Hall, Englewood-Cliffs, 1990, 624 S., 47 US-$
Empfehlung eines Stuttgarter Kollegen

A. Oram, S. Talbott Managing Projects with make
O'Reilly, Sebastopol, 1993, 149 S., 35 DM

W. R. Stevens UNIX Network Programming
Prentice Hall, Englewood Cliffs, 1990, 772 S., 60 US-$
C-Programme für Clients und Server der Netzdienste

J. Strang u. a. termcap & terminfo
O'Reilly, Sebastopol, 1988, 270 S., 17 £

I. A. Taylor Taylor UUCP
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/unix/uucp.ps.gz
1993, 93 S., Postscript

L. Wall, R. Schwartz Programming Perl
O'Reilly, Sebastopol, 1991, 482 S., 22 £

9. Grafik

– Newsgruppen:
comp.graphics.*
alt.graphics.*

– American National Standard for Information Systems
Computer Graphics – Graphical Kernel System (GKS)
Functional Description. ANSI X3.124-1985
GKS-Referenz

J. Bechlars, R. Buhtz GKS in der Praxis
Springer, Berlin, 1994, 500 S., 98 DM
GKS für FORTRAN-Programmierer

J. D. Foley Computer Graphics – Principles and Practice
Addison-Wesley, Reading, 1992, 1200 S., 83 US-$
Standardwerk zur Computer-Grafik

T. Gaskins The PHIGS Programming Manual
O'Reilly, Sebastopol, 1992, 908 S., 102 DM
Lehrbuch und Nachschlagewerk, auch unter X11

I. Grieger Graphische Datenverarbeitung
mit einer Einführung in PHIGS und PHIGS-PLUS
Springer, Berlin, 1992, 389 S., 48,– DM

H. Kopp Graphische Datenverarbeitung
Hanser, München, 1989, 211 S., 40 DM
mathematische Methoden, Algorithmen, GKS

10. Netze (TCP/IP, OSI, Internet)

– Newsgruppen:
comp.infosystems.*
comp.internet.*
comp.protocols.*
alt.best.of.internet
alt.bbs.internet
alt.internet.*
de.comm.internet
de.comp.infosystems
fr.comp.infosystemes

– Internet Resources Guide
NSF Network Service Center, Cambridge, 1993
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/resource-guide-help
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/resource-guide.ps.tar.gz
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/resource-guide.txt.tar.gz
Beschreibung der Informationsquellen im Internet

S. Carl-Mitchell, J. S. Quarterman Practical Internetworking with TCP/IP and UNIX
Addison-Wesley, Reading, 1993, 432 S., 52 US-$

D. E. Comer Internetworking with TCP/IP (4 Bände)
Prentice-Hall, Englewood Cliffs, I. Band 1991, 550 S., 90 DM;
II. Band 1991, 530 S., 88 DM; IIIa. Band (BSD) 1993, 500 S., 86 DM;
IIIb. Band (AT&T) 1994, 510 S., 90 DM
Prinzipien, Protokolle und Architektur des Internet

EARN Guide to Network Resource Tools
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/nettools.ps.gz
1993, 70 S., Postscript
Übersicht über Netzdienste wie Gopher, WWW, WAIS, ARCHIE, NETSERV, NetNews und Listserv

A. Gaffin, J. Heitkötter Big Dummy's Guide to the Internet
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/bdgtti2.ps.gz
1993, 220 S., Postscript, andere Formate im Netz
Einführung in die Dienste des Internet

H. Hahn, R. Stout The Internet Complete Reference
Osborne MacGraw-Hill, Berkeley, 1994, 818 S., 60 DM
Das Netz und seine Dienste von Mail bis WWW; Lehrbuch und Nachschlagewerk für Benutzer des Internet, Standardwerk

Ch. Hedrick Introduction to the Internet Protocols
ftp://ftp.ciw.uni-karlsruhe.e/pub/docs/net/general/tcp-ip-intro.ps.gz
ftp://ftp.ciw.uni-karlsruhe.e/pub/docs/net/general/tcp-ip-intro.doc.gz
1988, 20 S., ASCII und Postscript

Ch. Hedrick Introduction to Administration of an Internet-based Local Network
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/tcp-ip-admin.ps.gz
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/general/tcp-ip-admin.doc.gz
1988, 39 S., ASCII und Postscript
Adressen, Routing, Netztopologie im Internet

K. Hughes Entering the World-Wide Web: A Guide to Cyberspace
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/www/hughes-guide.ps.gz
1993, 20 S., Postscript
Erklärungen, Entstehung, Glossar

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

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., 25 US-$

T. L. LaQuey User's Directory of Computer Networks
Digital Press, Bedford, 1990, 653 S.,
Ins einzelne gehende Informationen über zahlreiche Netze

J. S. Quarterman The Matrix: Computer Networks and
Conferencing Systems Worldwide
Digital Press, Bedford, 1990, 746 S., 80 DM
Praxisnahe Einführung, Netzliste nicht mehr aktuell

M. T. Rose The Open Book
Prentice-Hall, Englewood Cliffs, 1990, 682 S., 64 US-$
OSI-Protokolle, Vergleich mit TCP/IP

M. Scheller u. a. Internet: Werkzeuge und Dienste
Springer, Berlin, 1994, 280 S., 49 DM
http://www.ask.uni-karlsruhe.de/books/inetwd.html

A. S. Tanenbaum Computer Networks
Prentice-Hall, London, 1988, 658 S., 88 DM
Einführung in Netze mit Schwerpunkt auf dem OSI-Modell

11. X-Window-System, Motif

– Newsgruppen:
comp.windows.x.*
fr.comp.windows.x11

– OSF/Motif Users's Guide
OSF/Motif Programmer's Guide
OSF/Motif Programmer's Reference
Prentice-Hall, Englewood Cliffs, 1990
Beschreibung der OSF/Motif Benutzeroberfläche

F. Culwin An X/Motif Programmer's Primer
Prentice-Hall, New York, 1994, 344 S., 80 DM

K. Gottheil u. a. X und Motif
Springer, Berlin, 1992, 694 S., 98 DM

A. Nye XLib Programming Manual
O'Reilly, Sebastopol, 1990, 635 S., 90 DM
Einführung in das XWS und den Gebrauch der XLib

V. Quercia, T. O'Reilly X Window System Users Guide
O'Reilly, Sebastopol, 1990, 749 S., 90 DM
Einführung in X11 für Benutzer

R. J. Rost X and Motif Quick Reference Guide
Digital Press, Bedford, 1993, 400 S., 22 £
Zusammenfassung aus den Referenz-Handbüchern

12. Programmieren allgemein

– Newsgruppen:
comp.programming
comp.unix.programmer
comp.lang.*
comp.software.*
comp.software-eng
comp.compilers
de.comp.lang.*

A. V. Aho u. a. Compilers, Principles, Techniques and Tools
Addison-Wesley, Reading, 1986, 796 S., 78 DM
Zum tieferen Verständnis von Programmiersprachen

B. Beizer Software Testing Techniques
Van Nostrand-Reinhold, 1990, 503 S., 43 US-$

F. P. Brooks jr. The Mythical Man-Month
Addison-Wesley, Reading, 1995, 322 S., 44 DM
Organisation großer Software-Projekte

N. Ford Programmer's Guide
`ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/misc/pguide.txt`
1989, 31 S., ASCII
allgemeine Programmierhinweise, Shareware-Konzept

T. Grams Denkfallen und Programmierfehler
Springer, Berlin, 1990, 159 S., 58 DM
PASCAL-Beispiele, gelten aber auch für C-Programme

D. Gries The Science of Programming
Springer, Berlin, 1981, 366 S., 48 DM
Grundsätzliches zu Programmen und ihrer Prüfung,
mit praktischer Bedeutung.

E. Horowitz Fundamentals of Programming Languages
Springer, Berlin, 1984, 446 S., ??? DM

Überblick über Gemeinsamkeiten und Konzepte von
Programmiersprachen von FORTRAN bis Smalltalk,
kein Programmierkurs, sondern eine Ergänzung dazu

M. Marcotty, H. Ledgard The World of Programming Languages
Springer, Berlin, 1987, 360 S., 90 DM

S. Pfleeger Software Engineering: The Production of Quality Software
Macmillan, 1991, 480 S., 22 £(Studentenausgabe)
Empfehlung aus dem Netz

R. W. Sebesta Concepts of Programming Languages
Benjamin/Cummings, Redwood City, 1993, 560 S., 65 US-$
ähnlich wie Horowitz

I. Sommerville Software Engineering
Addison-Wesley, Reading, 1992, 688 S., 52 US-$
Wie man ein Programmierprojekt organisiert;
Werkzeuge, Methoden; sprachenunabhängig

N. Wirth Systematisches Programmieren
Teubner, Stuttgart, 1993, 160 S., 27 DM
Allgemeine Einführung ins Programmieren, PASCAL-nahe

13. Programmieren in C/C++/Objective C

– Newsgruppen:
comp.lang.c
comp.std.c
comp.lang.object
comp.lang.c++
comp.lang.objective-c
comp.std.c++
de.comp.lang.c
de.comp.lang.c++

– Microsoft Quick-C-, C-6.0- und Visual-C-Handbücher
mehrere Bände bzw. Ordner

G. Booch Object-Oriented Analysis and Design with Applications
Benjamin + Cummings, Redwood City, 1994, 590 S., 112 DM

U. Claussen Objektorientiertes Programmieren
Springer, Berlin, 1993, 246 S., 48 DM
Konzept und Methodik von OOP, Beispiele und Übungen in C++,
aber kein Lehrbuch für C++

B. J. Cox, A. J. Novobilski Object-Oriented Programming
Addison-Wesley, Reading, 1991, 270 S., 76 DM
Objective C

H. M. Deitel, P. J. Deitel C How to Program
Prentice Hall, Englewood Cliffs, 1994, 926 S., 74 DM
Enthält auch C++. Ausgeprägtes Lehrbuch.

A. R. Feuer Das C-Puzzle-Buch
Hanser Verlag, München, 1991, 196 S., 38 DM
Kleine, feine Aufgaben zu C-Themen

O. Hartwig C Referenz-Handbuch
Sybex, Düsseldorf, 1987, 432 S., 54 DM (vergriffen?)
Nachschlagewerk für K&R-C und ANSI-C

R. House Beginning with C
An Introduction to Professional Programming
International Thomson Publishing, Australien, 1994, 568 S., 64 DM
Ausgeprägter Lehrbuch-Charakter, ANSI-C, vorbereitend auf C++

J. A. Illik Programmieren in C unter UNIX
Sybex, Düsseldorf, 1992, 750 S., 89 DM
Lehrbuch, C und UNIX mit Schwerpunkt Programmieren

R. Jones, I. Steart The Art of C Programming
Springer, Berlin, 1987, 186 S., 52 DM

B. W. Kernighan, D. M. Ritchie The C Programming Language
Deutsche Übersetzung: Programmieren in C
Zweite Ausgabe, ANSI C
Hanser Verlag, München, 1990, 283 S., 56 DM
Standardwerk zur Programmiersprache C, Lehrbuch

R. Klatte u. a. C-XSC
Springer, Berlin, 1993, 269 S., 74 DM
auch auf englisch erhältlich
C++-Klassenbibliothek für wissenschaftliches Rechnen

S. Lippman, J. Lajoie C++ Primer
Addison-Wesley, Reading, 3. Aufl. 1998, 1072 S., ?? DM
Verbreitetes Lehrbuch für Anfänger

P. J. Plauger, J. Brodie Referenzhandbuch Standard C
Vieweg, Braunschweig, 1990, 236 S., 64 DM

P. J. Plauger The Standard C Library
Prentice-Hall, Englewood Cliffs, 1991, 498 S., 73 DM
Die Funktionen der Standardbibliothek nach ANSI

H. Schildt ANSI C made easy
Osborne McGraw-Hill, Berkeley, 1989, 452 S., 50 DM
Leichtverständliche Einführung in ANSI-C

B. Stroustrup The C++ Programming Language
bzw. Die C++ Programmiersprache
Addison-Wesley, Reading/Bonn, 3. Aufl. 1997, 976 S., 100 DM
Lehrbuch für Fortgeschrittene, der Klassiker für C++

R. Ward Debugging C
Addison-Wesley, Bonn, 1988, 322 S., 68 DM
Systematische Fehlersuche, hauptsächlich in C-Programmen

14. Anwendungen

- Newsgruppen:
  comp.theory.info-retrieval
  comp.databases.*

- Guide to Commands
  STN International c/o FIZ Karlsruhe, 1991, 314 S.
  Beschreibung der Retrieval-Sprache Messenger

M. Gossens u. a. The LaTeX-Companion
Addison-Wesley, Reading, 1994, 530 S., 40 US-$

H. Kopka LaTeX - eine Einführung
Addison-Wesley, Bonn, 1990, 340 S., 68 DM
Ausführliche Anleitung zu LaTeX, viele Beispiele

H. Kopka LaTeX - Erweiterungsmöglichkeiten
Addison-Wesley, Bonn, 1990, 479 S., 80 DM
Erweiterungen, AMS-TeX, Grafik, Metafont, WEB

L. Lamport LaTeX User's Guide and Reference Manual
Addison-Wesley, Reading, 1986, 242 S., 78 DM
Standardwerk zu LaTeX

H. Partl u. a. LaTeX-Kurzbeschreibung
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/latex/lkurz.ps.gz
ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/latex/lkurz.tar.gz
1990, 46 S., Postscript und LaTeX-Quellen
Einführung, mit deutschsprachigen Besonderheiten (Umlaute)

E. D. Stiebner Handbuch der Drucktechnik
Bruckmann, München, 1992, 362 S., 98 DM

F. W. Weitershaus Duden Satz- und Korrekturanweisungen
Dudenverlag, Mannheim, 1980, 268 S., 17 DM (vergriffen?)
Hilfe beim Herstellen von Druckvorlagen

15. Sicherheit

- Newsgruppen:
  comp.security.*
  comp.virus
  sci.crypt
  alt.security.*
  alt.comp.virus
  de.comp.security

- RFC 1244 (FYI 8): Site Security Handbook
  ftp://ftp.ciw.uni-karlsruhe.de/pub/docs/net/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

F. L. Bauer Kryptologie
  Springer, Berlin, 1994, 369 S., 48 DM

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-Manager, mit Checkliste

D. Ferbrache A Pathology of Computer Viruses
  Springer, Berlin, 1992, 299 S., 74 DM
  Geschichte, Wirkungsweise, Gegenmaßnahmen, Reaktionen
  der Öffentlichkeit; auch UNIX- und Internet-Viren

B. Schneier Angewandte Kryptographie
  Addison-Wesley, Bonn, 1996, 844 S., 120 DM

16. Geschichte der Informatik

– Newsgruppen:
  comp.society.folklore
  alt.folklore.computers
  de.alt.folklore.computer

– 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

F. L. Bauer, G. Goos Informatik 2. Teil
  (siehe unter Informatik)

O. A. W. Dilke Mathematik, Maße und Gewichte in der Antike (Universalbibliothek Nr. 8687 [2])
Reclam, Stuttgart, 1991, 135 S., 6 DM

A. Hodges Alan Turing, Enigma
Kammerer & Unverzagt, Berlin, 1989, 680 S., 58 DM

S. Levy Hackers – Heroes of the Computer Revolution
Penguin Books, London, 1994, 455 S., 33 DM

R. Oberliesen Information, Daten und Signale
Deutsches Museum, rororo Sachbuch Nr. 7709 (vergriffen)

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, 3. Aufl. 1993, 220 S., 58 DM
Autobiografie Konrad Zuses

17. Computerrecht

- Newsgruppen:
  comp.society.privacy
  comp.privacy
  comp.patents
  alt.privacy
  de.soc.recht

- Computerrecht (Beck-Texte)
  Beck, München, 1994, 13 DM

U. Dammann, S. Simitis Bundesdatenschutzgesetz
Nomos Verlag, Baden-Baden, 1993, 606 S., 38 DM
BDSG mit Landesdatenschutzgesetzen und Internationalen Vorschriften; Texte, kein Kommentar

G. v. Gravenreuth Computerrecht von A – Z (Beck Rechtsberater)
Beck, München, 1992, 17 DM

H. Hubmann, M. Rehbinder Urheber- und Verlagsrecht
Beck, München, 1991, 319 S., 40 DM

A. Junker Computerrecht. Gewerblicher Rechtsschutz, Mängelhaftung, Arbeitsrecht. Reihe Recht und Praxis
Nomos Verlag, Baden-Baden, 1988, 267 S., 45 DM

18. Philosophische Feigenblätter

- Newsgruppen:
  comp.ai.philosophy
  sci.philosophy.tech
  alt.fan.hofstadter

D. R. Hofstadter Gödel, Escher, Bach - ein Endloses Geflochtenes Band
dtv/Klett-Cotta, München, 1992, 844 S., 30 DM

J. Ladd Computer, Informationen und Verantwortung
in: Wissenschaft und Ethik, herausgegeben von H. Lenk
Reclam-Band 8698, Ph. Reclam, Stuttgart, 15 DM

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., 6 DM

P. Schefe u. a. Informatik und Philosophie
BI Wissenschaftsverlag, Mannheim, 1993, 326 S., 38 DM
Sammlung von 18 Aufsätzen 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., 20 DM

H. Zemanek Das geistige Umfeld der Informationstechnik
Springer, Berlin, 1992, 303 S., 39 DM
Zehn Vorlesungen über Technik, Geschichte und Philosophie des Computers, von einem der Pioniere

19. Zeitschriften

- IX
  Verlag Heinz Heise, Hannover, monatlich, ca. 130 S.
  für Anwender von Multi-User-Systemen, technisch
  `http://www.ix.de/`
- Offene Systeme
  GUUG/Springer, Berlin, viermal im Jahr,
  offizielle Zeitschrift der German UNIX User Group
- The C/C++ Users Journal
  Miller Freeman Inc., USA, monatlich, ca. 150 S.
  `http://www.cuj.com/`
- Dr. Dobb's Journal
  Miller Freeman Inc., USA, monatlich, ca. 180 S.
  Software Tools for the Professional Programmer; viel C und C++
- unix/mail
  Hanser Verlag, München, sechsmal im Jahr, ca. 70 S.
  für Entwickler und Benutzer

– UNIX Open
  Aktuelles Wissen Verlagsgesellchaft mbH, Trostberg
  monatlich, ca. 100 S.
– Unix Welt
  IDG Communications Verlag, München, monatlich, ca. 110 S.
– Unix World
  MacGraw-Hill, USA, monatlich, ca. 200 S.
  das Neueste aus dem Ursprungsland von UNIX

Hier noch einige Verlage:

- Addison-Wesley, Bonn, `http://www.addison-wesley.de/`
- Carl Hanser Verlag, München, `http://www.hanser.de/`
- Verlag Heinz Heise, Hannover, `http://www.ix.de\`
- R. Oldenbourg Verlag, München, `http://www.oldenbourg.de/`
- O'Reilly, Deutschland, `http://www.ora.de/`
- 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 usw., `http://www.springer.de/`

> 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

Sach- und Namensverzeichnis

Einige Begriffe sind unter ihren Oberbegriffen zu finden, beispielsweise Gerätefile unter File oder Bourne-Shell unter Shell. Verweise (*s. ...*) zeigen entweder auf ein bevorzugtes Synonym, auf einen Oberbegriff oder auf die deutsche Übersetzung eines englischen oder französischen Fachwortes.