Einführung in Unix

Michaela K. Harlander
GeNUA Gesellschaft für Netzwerk- und Unix-Administration mbH

August 1992

Dies ist eine Einführung in Unix für Benutzer, die noch überhaupt keine Erfahrung mit Unix haben. Sie erklärt die wesentlichen Grundlagen von Unix, was man braucht, um eigene Dateien zu erzeugen und Programme zu übersetzen. Ein kurzer Überblick über Netzwerke wird ebenso gegeben wie eine Einführung in das X-Window-System. Grundkenntnisse über Computer sollten beim Leser vorhanden sein.

Eine alphabetische Liste wichtiger Unix-Kommandos ergänzt diese Einführung. Von Zeit zu Zeit sollte man darauf einen Blick werfen, während man dieses Schriftstück durcharbeitet.

Dieses Dokument enthält außerdem Zusammenstellungen der wichtigsten Kommandos in den Editoren `vi` und `emacs`.

Inhaltsverzeichnis

Dieses Dokument ist per ftp von ftp.informatik.tu-muenchen.de im Verzeichnis pub/doc/os als dintro.ps erhältlich.

Kommentare, Kritik, Spenden etc. an
michaela@genua.sub.org oder harmic@physik.tu-muenchen.de

Vorbemerkungen und Danksagung

Diese Einführung ist eine erweiterte Übersetzung meiner „Introduction to Unix“. Bei der Verfassung deutscher Computerliteratur stellt sich die Frage nach der Verwendung englischer oder eingedeutschter Fachbegriffe. Mir persönlich sagt weder die einfache Übernahme aller englischen Fachbegriffe zu noch die gewaltsame Eindeutschung derselben. Daher habe ich versucht, einen Mittelweg zu gehen. Ein „Rechenwerk“ anstelle der CPU oder gar einen „Heimatkatalog“ anstelle des Homedirectories wird man hier nicht finden, aber auch keine „Filename-Expansion“, da die Übersetzung „Dateinamen-Ergänzung“ genauso verständlich und lesbar ist wie das Original. Schließlich sind auch die englischen Begriffe nicht immer glücklich gewählt. In vielen Fällen habe ich aber die englischen Begriffe in Klammern angegeben oder alternativ zu den deutschen verwendet. Es ist eben wesentlich einfacher für Sie, das Kommando „y“ im Editor `vi` zu verstehen, wenn Sie wissen, daß es vom englischen „yank“ stammt.

Da ich die Leserinnen unter den Lesern nicht durch eine männlich betonte Sprachen verschrecken will, ich aber andererseits die oft verwendeten Formen „Leser/innen“ oder gar „LeserInnen“ für schlicht unlesbar halte, verwende ich männliche und weibliche Singularformen abwechselnd. Und ein Plural ist für mich immer noch geschlechtsneutral.

Die Maschinen– und Benutzernamen sind der mittlerweile fünfbändigen Trilogie „Per Anhalter durch die Galaxis“ von Douglas Adams entlehnt.

Dank gebührt hier natürlich allen Leuten, die mich schon bei der ersten englischen Version mit guten Ratschlägen und Änderungswünschen überhäuft haben. Besonders zu nennen ist hier Bernhard Schneck, der mit unzähligen wertvollen Hinweisen und einigen nicht ganz so wertvollen, aber dafür umso heftigeren Diskussionen über Nomenklatur und ähnliche weltanschauliche Fragen wesentlich zur Verbesserung der Einführung und zur Strapazierung meiner Nerven beitrug. Er und Hannes Faestermann haben das Korrekturlesen der deutschen Version übernommen. Leser der englischen Version regten eine Vielzahl von Verbesserungen, z.B. den Index, an. Ihnen allen gilt mein herzlicher Dank.

Natürlich ist diese Einführung alles andere als vollkommen. Wenn Sie konkrete Verbesserungsvorschläge haben, so teilen Sie mir diese bitte mit. Abhängig von Arbeitsaufwand, Termindruck, Wetter und meiner Stimmungslage werden Sie Ihre Vorschläge dann in der nächsten Version wiederfinden — oder auch nicht.

1 Grundlagen von Unix

Unix ist ein Mehrbenutzer– und Mehrprozeß–System. Es ermöglicht, eine Maschine mit anderen Benutzern gemeinsam zu nutzen und mehrere Prozesse gleichzeitig laufen zu lassen. Die Struktur ist etwa folgende:

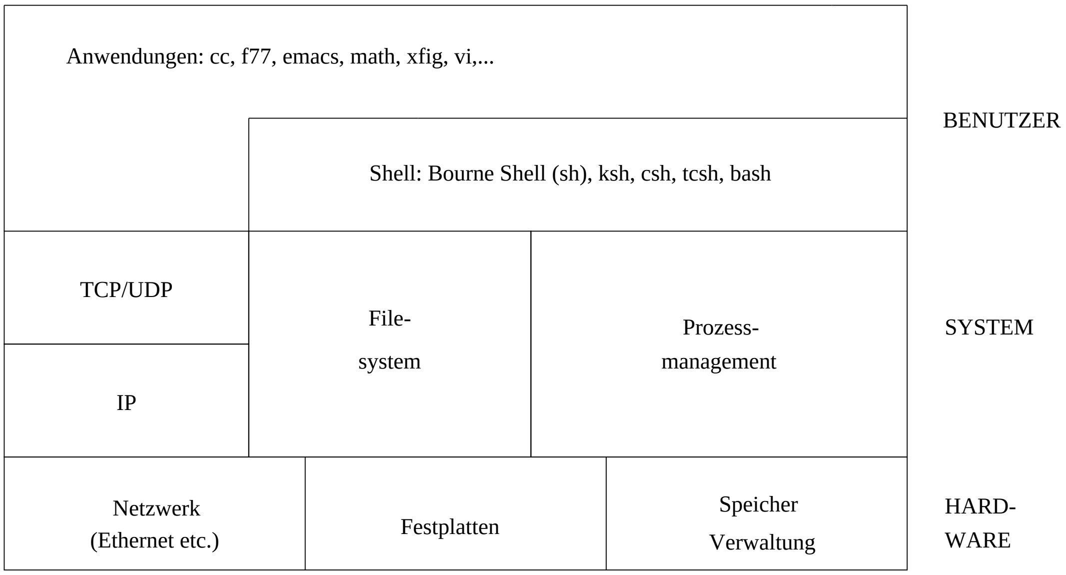

Abbildung 1: Die Struktur von Unix

Das Fundament wird von der Hardware gebildet: vom Hauptspeicher, in dem die Prozesse während der Ausführung gehalten werden (allerdings werden die Prozesse manchmal temporär auf die Festplatten ausgelagert), von der zentralen Recheneinheit (CPU) und eventuell einem Prozessor für Fließkommarechnugen, von den Festplatten, auf denen Dateien und Programme dauerhaft gespeichert werden, sowie vom Netzwerk, das die Maschine, an der Sie arbeiten, mit anderen Maschinen verbindet.

Die Systemebene wird im wesentlichen von Management-Systemen für die Hardware gebildet. Das Prozeßmanagement kümmert sich sowohl um Ihre Prozesse als auch um die anderer Benutzer, so daß sie sich nicht überschneiden und die Rechenleistung untereinander sinnvoll aufteilen. Die CPU kann immer nur ein Programm gleichzeitig bearbeiten, alle anderen Programme müssen währenddessen warten. Natürlich sehen Sie das nicht direkt, da diese Programme sehr häufig von der CPU gewechselt werden.

Wenn allerdings viele Prozesse eine CPU belagern, muß jeder Prozeß länger warten, bis er wieder bearbeitet wird, und die Laufzeit verlängert sich entsprechend. Alle Prozesse derselben Priorität müssen gleich lange warten. Die Priorität kann aber geändert werden, so daß einige Prozesse häufiger von der CPU bearbeitet werden als andere und somit eine kürzere Laufzeit haben.

Das Filesystem verwaltet den Platz auf den Festplatten und erlaubt es, mehrere Platten als ein homogenes Speichermedium zu sehen. Wenn auf Dateien einer nichtlokalen Platte zugegriffen werden soll, muß das Filesystem das Netzwerk verwenden.

Damit ein Netzwerk funktioniert, ist es notwendig, eine „Syntax“ für den Datentransfer zu definieren. Diese „Syntax“ wird über Netzwerkprotokolle geregelt.

Letztere werden in verschiedene Ebenen unterteilt: Während IP (Internet Protocol) lediglich Datenpakete über ein Kabel schickt, stellen TCP (Transmission Control Protocol) und UDP (User Datagram Protocol) weitergehende Fähigkeiten zur Verfügung. TCP stellt z.B. sicher, daß Daten auch wirklich am Bestimmungsort ankommen, selbst wenn einzelne Datenpakete unterwegs verloren gehen sollten. Um Netzwerk-Anwendungen zu benutzen, muß man aber über Netzwerkprotokolle nicht Bescheid wissen. Einige gebräuchliche Netzwerkanwendungen sind `rlogin`, das das Einloggen auf einer nicht-lokalen Maschine ermöglicht, `ftp` für Dateitransfer sowie verschiedene Programme zum Versenden und Empfangen elektronischer Post.

Zunächst jedoch sehen Sie von Ihrem System die Shell, die man am besten als Kommandointerpreter beschreibt. Für Sie als Benutzer spielt es keine besondere Rolle, ob Sie ein Programm oder ein Shellkommando aufrufen – außer in einem wesentlichen Punkt: da es mehrere verschiedene Shells gibt, können sich Shellkommandos sogar auf einem einzigen System von Benutzerin zu Benutzerin unterscheiden.

Aber nicht nur das trägt zur Verwirrung bei. Es gibt nämlich auch verschiedene Arten von Unix. Moderne Unix-Systeme kann man im wesentlichen in zwei Klassen einteilen: diejenigen, die sich an SYSTEM V anlehnen (SYSTEM V wurde von AT&T entwickelt), und diejenigen, die sich vom BSD der University of California in Berkeley ableiten.[1] Die Benutzeroberfläche der beiden unterscheidet sich in einigen Punkten. Einige Kommandos gibt es nur auf einem der beiden Systeme, andere haben unterschiedliche Syntax. Neuere Versionen von SYSTEM V wie SYSTEM V RELEASE 4 (SVR4) wurden jedoch stark von BSD beeinflußt und kennen viele Kommandos, die früher nur BSD–Benutzern vorbehalten waren. Daher treffen einige Bemerkungen in dieser Einführung über die Besonderheiten von SYSTEM V nicht mehr für diese neueren Versionen, sondern nur noch für ältere wie SYSTEM V RELEASE 3 (SVR3) zu. Wenn eine Besonderheit für SVR3 angegeben ist, so gilt das i.d.R. auch für ältere SYSTEM V - Versionen. Im Zweifelsfall wende man sich an die Systemhandbücher.

Doch zurück zum Aufbau von Unix: Die oberste Ebene wird von den Anwendungen gebildet. Das können Compiler sein, Editoren, Programme für symbolische Mathematik, Graphikprograme etc. Diese und einige Shell– bzw. Systemkommandos werden Sie vermutlich am meisten verwenden, abgesehen natürlich von eigenen Programmen.

Abb. 2 zeigt das Schema eines Mehrbenutzer– und Mehrprozeß–Systems. Jeder Benutzer hat eine Shell, die sich von Benutzer zu Benutzer unterscheiden kann. Davon ausgehend kann jede Benutzerin verschiedene Prozesse (tasks, jobs) laufen lassen. Die Prozesse werden im Schema durch ein P gefolgt von einer Ziffer symbolisiert. Da die Prozesse von der Shell aus gestartet worden sind, nennt man sie „Kinder“ dieser Shell. Das können natürlich auch weitere Shells sein.

Nun, da Sie schon etwas über Unix wissen, können Sie daran gehen, es auch zu benutzen. Bevor Sie jedoch anfangen zu tippen, sollten Sie wissen, daß Unix Groß- und Kleinbuchstaben unterscheidet.

[1] Eigentlich gilt die Bezeichnung UNIX nur für AT&T's (heute USL) Produkt. Wir folgen der allgemein üblichen Sprachregelung und bezeichnen hier mit Unix alle UNIX-artigen Systeme. Der Terminus „UNIX“ bezieht sich nur dann auf das USL-Produkt, wenn wir es nur mit Großbuchstaben schreiben.

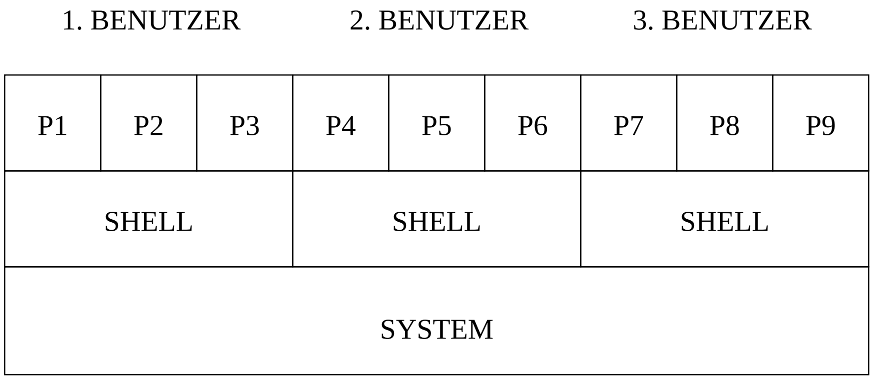

Abbildung 2: Ein Multiuser–System

2 Aller Anfang ...

2.1 Benutzerkennung und Paßwort

Um an einer Unix–Maschine arbeiten zu können, muß man eine Arbeitssitzung anmelden und sich als dazu berechtigter Benutzer ausweisen. Dieser Vorgang heißt einloggen. Bevor man sich einloggen kann, braucht man eine Benutzerkennung (Account). Üblicherweise erhält man diese Kennungen vom Systemverwalter. Dort bekommen Sie einen Loginnamen, der meistens vom richtigen Namen abgeleitet ist. Ferner werden Sie einer Gruppe zugewiesen. Die Gruppe kann z.B. alle Angehörigen eines Universitätsinstitutes umfassen, alle Angestellten einer Abteilung, alle Studenten einer Fakultät etc. Die Gruppenzugehörigkeit ist wichtig für die Zugriffsberechtigungen auf Dateien (siehe Kapitel 5.5).

Der Systemverwalter weist Ihnen eine Loginshell zu. Das ist die Shell, mit der Sie nach dem Einloggen arbeiten. Die Eigenschaften verschiedener Shells erklären wir noch in Kapitel 3. Die Loginshell kann mit Hilfe des Kommandos `chsh` auf manchen Systemen geändert werden. Außerdem muß ein Paßwort für die neue Kennung gesetzt werden. Das Paßwort dient zur Identifizierung der Benutzerin. Es darf nur dem Besitzer der Kennung bekannt sein und wird in verschlüsselter Form gespeichert, so daß niemand, nicht einmal die Systemverwalterin, es lesen kann. Das Paßwort sollte mindestens 6 Zeichen lang sein. Mehr als 8 Zeichen werden meistens nicht verwendet, auch wenn ein längeres Paßwort angegeben wird. Neuere Systeme lassen auch längere Paßworte zu. Das Paßwort sollte aus einer Mischung von Großbuchstaben, Kleinbuchstaben, Ziffern und Sonderzeichen bestehen. Die Verwendung von Steuerzeichen ist zwar möglich, wird aber nicht empfohlen, da dies zu unerwünschten Effekten führen kann. Ein Paßwort nur aus Kleinbuchstaben oder nur aus Ziffern ist nicht besonders sicher, da es viel leichter erraten werden kann. Moderne Paßwort–Suchprogramme, die übrigens weit verbreitet sind, haben mit solchen Paßwörtern relativ leichtes Spiel. Es sollte auch keine offenkundig mit der Person verbundene Information als Paßwort verwendet werden. Name, Telefonnummer etc. als Paßwort sind ziemlich fehl am Platz. Auch die Namen bekannter Romanhelden sind eher ungeeignet.

Das Paßwort sollte etwa alle drei Monate geändert werden oder sobald man vermutet, daß jemand das Paßwort wissen könnte. Zum Ändern verwendet man den Befehl `passwd` oder `yppasswd`.

2.2 Vom Umgang mit den Maschinen

Bitte achten Sie darauf, daß Sie nicht versehentlich die Maschine oder Peripheriegeräte ein- oder ausschalten. Eine Unix-Maschine wird üblicherweise nur von der Systemverwalterin unter Verwendung bestimmter Prozeduren ein- oder ausgeschaltet. Einfaches Ausschalten kann großen Schaden, besonders an den Daten auf den Festplatten, anrichten. In besonderen Notfällen, z.B. wenn der Computer zu rauchen beginnt, sollten Sie die Maschine natürlich trotzdem ausschalten, um größeren Schaden (z.B. Brand) zu vermeiden.

Will man eine Maschine benutzen, deren Bildschirm dunkel ist, so drückt man zunächst auf eine Taste[2]. Viele System verwenden nämlich einen Bildschirm-Schoner (Screen-Saver), der den Bildschirm verdunkelt, wenn längere Zeit keine Eingabe erfolgt. Ein Tastendruck stellt den Bildschirminhalt wieder her.

Falls das nicht funktioniert, muß man den Bildschirm einschalten. Vergewissern Sie sich aber, daß Sie wirklich den Schalter für den Bildschirm und nicht etwa den für die Maschine betätigen. Dabei sollte man daran denken, daß manche Computer die Recheneinheit im Bildschirmgehäuse eingebaut haben, so daß der Schalter für den Bildschirm zugleich der für die Maschine ist. Im Zweifelsfall unbedingt fragen!

2.3 Einloggen

Bevor Sie sich einloggen, werden Sie am Bildschirm etwa folgendes sehen:

```
login:
Password:
```

Das Paßwort erscheint während der Eingabe nicht am Bildschirm, d.h. Sie müssen es „blind“ eintippen. Nach der Eingabe von Benutzerkennung und Paßwort wird das Paßwort überprüft. Falls es korrekt eingegeben wurde, wird die Loginshell gestartet und eine Arbeitsumgebung eingerichtet. Was im einzelnen passiert, hängt von verschiedenen Dingen, wie etwa der gewählten Loginshell oder dem Terminaltyp ab. Falls Sie eine Workstation oder ein X-Terminal verwenden, werden meistens mehrere Fenster gestartet. Der Umgang mit Fenstern wird im Kapitel 11 über das X-Window-System näher erläutert.

Falls Sie sich über einen PC an eine Workstation einloggen, werden Sie dazu spezielle Software verwenden.

Nach dem Login werden die Kurzmitteilungen „message of the day“ (motd) ausgegeben, die aktuelle Informationen über das System enthalten. Die motd sollte immer sorgfältig gelesen werden, da dort über wichtige Neuigkeiten wie geplante Ausfallzeiten des Systems und neue Software–Installationen informiert wird.

[2] `SHIFT` und `CTRL` bzw. `STRG` sind gut geeignet, da sie keine Eingabe an eventuell laufende Anwendungen verursachen.

Nach den Kurzmitteilungen erscheint die Eingabeaufforderung (Prompt). Der Prompt ist eine Zeichenfolge, die von der Shell an den Anfang jeder Zeile gesetzt wird, in der eine Eingabe erwartet wird. In den Beispielen dieser Schrift wird ein einfaches `$`-Zeichen als Eingabeaufforderung verwendet. Das kann von System zu System und von Benutzer zu Benutzer jedoch völlig verschieden sein. Oft enthält der Prompt den Namen des Rechners oder des aktuellen Arbeitsverzeichnisses. Für die Eingabe der Kommandos ist das natürlich ohne Bedeutung.

Wenn Sie an einer Maschine eingeloggt sind, die an einem Netzwerk angeschlossen ist, so können Sie sich an allen anderen Maschinen einloggen, die über das Netz erreichbar sind und an denen Sie eine Benutzerkennung haben. Für den Login auf einer anderen als der lokalen Maschine, vor der Sie sitzen, müssen Sie das Kommando `rlogin` oder `telnet` verwenden. Im Gegensatz zum normalen `login` muß nun der Name der gewünschten Maschine angegeben werden (siehe auch Kapitel 9.1).

Beispiel:

```
$ rlogin heart_of_gold.magrathea.universe
```

Falls Sie auf der nicht-lokalen Maschine einen anderen Loginnamen haben, verwenden Sie:

```
$ rlogin heart_of_gold.magrathea.universe -l other_loginname
```

Eine `telnet`–Sitzung sieht so aus:

```
$ telnet heart_of_gold.magrathea.universe
Trying 42.42.42.42.....
Connected to heart_of_gold.magrathea.universe
Escape character is '^]'.


UNIVERSE Unix (heart_of_gold)

login: loginname
Password:
```

Zum Unterschied zwischen `telnet` und `rlogin` siehe Kapitel 9.2. Ausloggen kann man sich mit dem Kommando `logout` oder durch Tippen des EOF-Buchstabens (EOF = end of file). Dieser ist meist `C-d`, wobei die „Control“ (Ctrl) Taste (auf deutschen Tastaturen oft „Strg“ für Steuerung) und „d“ gleichzeitig gedrückt werden. Einige wenige Systeme verwenden `C-z` als EOF. Bitte dies erst dann ausprobieren, wenn die anderen beiden Möglichkeiten nicht funktionieren. Auf den meisten Systemen bewirkt `C-z` nämlich lediglich das Stoppen eines Jobs.

2.4 Eingabe von Kommandos

Die Kommandoeingabe erfolgt durch Angabe des Kommandonamens und eventuell Optionen und/oder Argumenten.

Optionen beeinflussen die Wirkungsweise eines Kommandos. Ein Beispiel für eine Option, nämlich `-l`, haben wir bereits beim `rlogin`–Kommando verwendet. Eine Option besteht typischerweise aus einem Minuszeichen gefolgt von einem Einzelbuchstaben. Mehrere Optionen können oft mit einem Minuszeichen gefolgt von den Einzelbuchstaben aller gewünschten Optionen ohne weitere Zwischenräume angegeben werden. Einige Programme, z.B. Compiler, verlangen jedoch durch Zwischenräume getrennte Angabe aller Optionen jeweils mit einem eigenen Minuszeichen. Im Zweifelsfall konsultiere man die Manuale.

Argumente spezifizieren die Objekte, auf die ein Kommando wirkt. Will man etwa eine Datei seitenweise ausgeben lassen, so gibt man `more dateiname` ein. Das Kommando `more` zeigt Dateien Seite für Seite, `dateiname` gibt an, welche Datei dargestellt werden soll.

Eine Kommandozeile wird durch Drücken der Tasten `RETURN` oder `ENTER` abgeschlossen und das Kommando vom Rechner ausgeführt.

3 Shells

3.1 Eigenschaften, Startup–Dateien

Die Shell kümmert sich um die Arbeitsumgebung. Sie setzt den Prompt, teilt der Maschine mit, wo nach neuer E-Mail (elektronischer Post) geschaut werden soll etc. Um dies zu tun, führt die Shell mindestens eine Startup–Datei, meistens sogar zwei (siehe Tabelle) aus. Dabei wird die erste Startup–Datei nur von der Login–Shell gelesen, die zweite auch von anderen Shells. Einige Shells ermöglichen die nochmalige Ausführung bereits eingegebener Kommandozeilen, ohne alles nochmals tippen zu müssen (History–Mechanismus) sowie das zeitweilige Anhalten eines laufenden Jobs (Job–Control).

Es folgt eine kurze Übersicht über die Eigenschaften verschiedener Shells.

shell	sh	csh	ksh	bash
1. Startup-Datei	`.profile`	`.login`	`.profile`	`.bash_profile` or `.profile`
2. Startup-Datei	–	`.cshrc`	`$ENV`	`$ENV`
Eigenschaften	nur sehr elementar	Job-Control einfache History	Job-Control gute History	job control gute History

sh wird auch Bourne Shell genannt. `$ENV` als zweites Startup-File für ksh und bash bedeutet, daß der Name dieser zweiten Datei in der ersten Startup-Datei gesetzt werden kann. Oft wird `.kshrc` für ksh und `.bashrc` für bash verwendet. Das `$`-Zeichen bedeutet in Kombination mit Shell-Variablen, daß der Wert dieser Variablen eingesetzt werden soll.

3.2 History und Job–Control

Die folgende Tabelle gibt einen Überblick über History–Kommandos in den verschiedenen Shells.

Wirkung	*csh*	*ksh*	*bash*
wiederhole letztes Kommando	`!!`	`r`	`!!`
wiederhole letztes Kommando, das mit `exp` *begann*	`!exp`	`r exp`	`!exp`
zeige die letzten Kommandos	`history`	`history` oder `fc -l`	`history`
wiederhole Kommando Nr. `n`	`!n`	`r n`	`!n`
wiederhole letztes Kommando, aber ersetze `a` *durch* `b`	`^a^b`	`r a=b`	`^a^b`

Es gibt in allen drei Shells noch fortschrittlichere Techniken, um vorangegangene Kommandozeilen zu modifizieren. So erlauben ksh und bash, nach einem vorangegangenen Ausdruck zu suchen und holen die entsprechende Kommandozeile zurück, so daß sie editiert werden kann. Für Einzelheiten wende man sich —wie üblich— ans Handbuch.

Auf BSD–Systemen (und SVR4) kann man mit `C-z` einen bereits laufenden Job anhalten. Das bedeutet, daß die Bearbeitung dieses Prozesses unterbrochen wird und die Eingabeaufforderung der Shell erscheint. Das Kommando `jobs` zeigt alle gestoppten Jobs mit einer Nummer versehen an. Das Kommando `fg %n` holt Job Nr. `n` zurück in den Vordergrund. `fg` ohne Argument holt den aktuellen Job zurück. Dies ist meistens der zuletzt gestoppte Job und wird in der Ausgabe des Kommandos `jobs` mit einem `+` markiert.

Tippt man nach Stoppen eines Jobs `bg %n`, so wird Job Nr. `n` in den Hintergrund geschickt und dort weiter bearbeitet, ohne daß die Shell auf die Beendigung des Jobs wartet. Daher erscheint sofort wieder die Eingabeaufforderung. Man kann einen Job auch direkt im Hintergrund starten, indem man an die entsprechende Kommandozeile ein `&`-Zeichen anhängt. Diese Methode funktioniert auch bei Systemen, die das nachträgliche Stoppen und In-den-Hintergrund-Schicken eines Jobs nicht erlauben (SVR3).

3.3 Das `nice` Kommando

Rechenintensive Jobs können eine Maschine für andere Benutzer ziemlich unbrauchbar machen, selbst wenn andere Benutzer nur wenig rechenintensive Aufgaben wie Editieren erledigen möchten. Um diese Situation zu entschärfen, sollten aufwendige Jobs mit einer niedrigeren Priorität gestartet werden. Dann nämlich wird z.B. der Editor bevorzugt behandelt, so daß vernünftiges Arbeiten damit möglich ist, obwohl auf der Maschine noch ein großer Job rechnet. Falls keine anderen Prozesse auf der Maschine laufen, wird die Laufzeit des Jobs durch die niedrigere Priorität nicht beeinflußt. Prozesse mit gleicher Priorität werden von der CPU gleichberechtigt behandelt.

Die Priorität wird mit Hilfe des `nice`-Kommandos verringert. Die Skala von `nice`-Werten reicht meistens von 0 bis 39, wobei für Benutzerprozesse der Wert

20 als Voreinstellung gewählt wird. Je höher der `nice`–Wert, desto niedrigerer die Priorität (man ist dann „nicer“ zu den anderen Benutzern). Ein Benutzer kann übrigens die Priorität seiner Prozesse nur verringern, nicht aber erhöhen.

Um einen Job mit niedrigerer Priorität zu starten, verwendet man das Kommando `nice` gefolgt von einem Wert und dem zu startenden Kommando. Um beispielsweise ein Programm mit einem um 10 niedrigeren `nice`–Wert zu starten, tippt man

```
nice -10 Programm
```

Unglücklicherweise ist diese Syntax nicht eindeutig, wenn die Benutzerin die Shell csh verwendet. Csh hat nämlich ein eigenes `nice`-Kommando und dieses verlangt anstelle des „–“ ein „+“, weil csh dieses Zeichen als Vorzeichen interpretiert und nicht als Options–Minuszeichen wie das systemeigene `nice`–Kommando, das bei allen anderen Shells verwendet wird. Um dieses Problem zu umgehen, kann man mit dem Kommando `/bin/nice` immer das systemeigene `nice`–Kommando mit der „–“ Syntax aufrufen.

4 Hilfestellungen

Unix–Systeme stellen eine Vielzahl von online–Hilfestellungen zur Verfügung. Allerdings sollte man als Anfänger nicht zuviel von diesen Hilfestellungen erwarten. Ihren wahren Nutzen offenbaren sie erst der etwas geübteren Unix–Benutzerin.

- `apropos Ausdruck` ist geeignet, wenn man sich an ein Kommando nicht mehr richtig erinnern kann oder das Kommando für eine bestimmte Operation nicht kennt. Das System sucht dann in den Überschriften der Manuale nach `Ausdruck` und gibt jeden gefundenen Kontext aus. Anstelle von `apropos` kann auch `man -k` verwendet werden.
- `whatis Kommando` liefert eine Kurzbeschreibung von `Kommando`.
- `man Kommando` liefert eine ausführliche Beschreibung von `Kommando`. Dieser Befehl ruft die online–Manualseiten auf. Diese Manualseiten enthalten einen Großteil der in den gedruckten Manualen vorhandenen Informationen, jedoch nicht alles.

Im `emacs`–Editor gibt es ebenfalls ein Informationssystem, das wir zusammen mit emacs in Anhang B.22 behandeln. Manche Rechnerinstallationen haben zusätzlich noch eigene Informationssysteme für lokal spezifische Informationen.

Im Problemfall hilft oft der Griff zu den gedruckten Handbüchern. Meistens gibt es dort einen Ordner „Global Index“ o.ä., der ein schnelles Auffinden des benötigten Manuals ermöglicht. Natürlich kann man sich auch an eventuell vorhandene Benutzerberatungen oder Programmierberatungen wenden. Der Gang zum Systemverwalter sollte allerdings nur dann erfolgen, wenn alle anderen Möglichkeiten ausgeschöpft wurden oder die Lösung nur mit Hilfe des Systemverwalters möglich ist, z.B. wenn die Benutzerkennung gesperrt wurde.

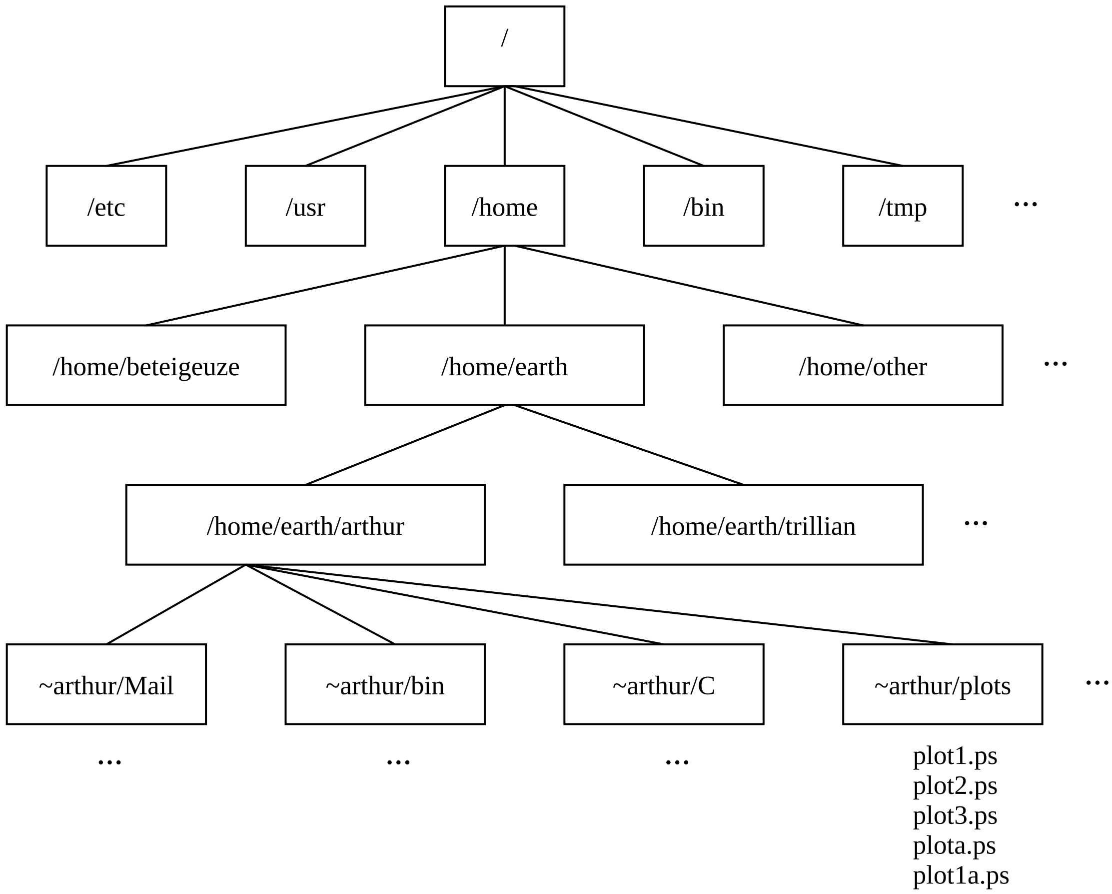

Abbildung 3: Ausschnitt aus einem typischen Unix–Datei–Baum

5 Das Filesystem

5.1 Struktur

Unix betrachtet eine Datei (File) als kontinuierlichen Informationsfluß, also im wesentlichen als eine Sequenz von Bytes. Ein Byte definiert eine kleine Informationseinheit (auf vielen Systemen ist ein Byte die Größe des Speicherplatzes, den ein einzelner Buchstabe benötigt). Jede Datei hat einen Namen. Eine spezielle Datei, die die Namen anderer Dateien enthält, wird Verzeichnis oder Directory genannt. Verzeichnisse werden zur Strukturierung der Dateien auf einem System verwendet. Auch die voreingestellten Eingabe– und Ausgabe–Kanäle sind spezielle Dateien.

Die Filesystemstruktur unter Unix ist hierarchisch. Ausgangspunkt ist das root–Directory. Von dort aus entwickelt sich das Filesystem baumartig. Pfadkomponenten werden durch den Schrägstrich `/` getrennt. Abb. 3 zeigt einen kleinen Ausschnitt eines solchen Datei–Baums, in dem als Beispiel auch das Homedirectory des fiktiven Benutzers „arthur“ gezeigt wird.

`/home/earth/arthur` ist das Homedirectory des Benutzers `arthur`, in dem er sich nach dem Einloggen wiederfindet. `arthur` gehört zur Gruppe `earth`, und in diesem Beispiel gliedert sich der Datei–Baum in die Gruppen auf, bevor er sich in die Homedirectories aufspaltet. Auf den meisten Systemen können in csh, bash und ksh die Homedirectories mit der Abkürzung `~loginname` angesprochen werden. Das eigene Homedirectory wird dann einfach mit `~` bezeichnet.

Es gibt keine Beschränkung für die Zahl der Unterverzeichnisse. Deswegen sollte man Unterverzeichnisse intensiv als Mittel zur Strukturierung des Homedirectories verwenden. So ist es praktisch, ein Subverzeichnis für C–Programme anzulegen und dieses Verzeichnis wieder durch Unterverzeichnisse für jedes Projekt zu untergliedern. Ein Subdirectory namens `bin` wird oft verwendet, um Binärdateien (ausführbare Programme) abzulegen. Folgende Kommandos braucht man für den Umgang mit Directories:

`mkdir dirname` lege das neue Subdirectory `dirname` an

`rmdir dirname` entferne das (leere!) Verzeichnis `dirname`

`cd dirname` gehe in das Verzeichnis `dirname`

`cd ..` gehe im Dateibaum ein Verzeichnis nach oben (Elternverzeichnis)

`cd` gehe ins Homedirectory

`ls` liste die Dateien im Arbeitsverzeichnis

`ls -l` ausführliche Liste des aktuellen Verzeichnisses

`pwd` zeige den Namen des aktuellen Verzeichnisses

Das aktuelle Verzeichnis, auch Arbeitsverzeichnis genannt, wird durch einen einzelnen Punkt repräsentiert, das Elternverzeichnis durch zwei Punkte. Beispiel:

```
$ pwd
/home/earth/arthur
$ mkdir test
$ cd test
$ pwd
/home/earth/arthur/test
$ cd ../C
$ pwd
/home/earth/arthur/C
$ cd
$ pwd
/home/earth/arthur
$
```

5.2 Dateinamen

Unix stellt keine besonderen Ansprüche an Dateinamen und erlaubt im Prinzip alle Zeichen außer dem Schrägstrich in Dateinamen. Will man allerdings auf Nummer Sicher gehen, so sollte man sich auf folgende Zeichen in Dateinamen beschränken:

- Klein– und Großbuchstaben
- Ziffern

- Unterstreichzeichen (_)
- Punkt (.)
- Minuszeichen (-)

Andere Zeichen können von der Shell mißinterpretiert werden. Bei der Verwendung von Groß– und Kleinbuchstaben sollte man daran denken, daß Unix diese unterscheidet. Dateinamen dürfen bis zu 253 Zeichen lang sein. Einige ältere SYSTEM-V-Maschinen lassen allerdings höchstens 14 Zeichen lange Dateinamen zu. Ebenso können manche Anwendungen nur Dateinamen verarbeiten, die deutlich kürzer als 253 Zeichen sind.

Im Gegensatz zu MS-DOS verwendet Unix keine Extensions (das sind in DOS Teile des Dateinamens, die durch einen Punkt abgetrennt sind und eine spezielle Bedeutung für das Betriebssystem haben). Daher kann man in Unix beliebig viele Punkte in einem Dateinamen verwenden, denn der Punkt hat keine spezielle Bedeutung. Die Feststellung, ob eine Datei ausführbar ist etc. erfolgt in Unix über andere Mechanismen, während dies in DOS über Extensions gesteuert wird. Allerdings verwenden unter Unix einige Anwendungen, z.B. Compiler, die letzten Buchstaben in einem Dateinamen, die durch einen Punkt abgetrennt wurden, zur Einordnung der Dateien. Einige übliche Endungen sind:

.tex TeX–Quellcode

.dvi übersetztes TeX

.ps Postscript–Dateien (Postscript ist eine Graphiksprache)

.tar Archiv–Datei

.Z mittels `compress` komprimierte Datei

.F mittels `freeze` komprimierte Datei

.c C–Quellcode

.f FORTRAN–Quellcode

Es gibt noch eine Vielzahl solcher Endungen. Die wichtigsten Endungen, die Compiler verwenden, werden im Kapitel über Compiler erklärt.

Dateinamen, die mit einem Punkt beginnen, sind „unsichtbar“, d.h. `ls` zeigt diese Dateien nicht an. So beginnen alle Startup–Dateien mit einem Punkt, da man diese nicht bei jedem `ls` sehen will. `ls -a` zeigt auch unsichtbare Dateien an.

Um eine Datei anzusprechen, kann man entweder den absoluten oder den relativen Pfadnamen verwenden. Der absolute Pfadname besteht aus dem kompletten Pfad vom root–Verzeichnis `/` aus bis zu dem Verzeichnis, in dem sich die Datei befindet. Daher beginnen absolute Pfadnamen immer mit einem `/`.
`/home/earth/arthur/plots/plot1.ps` ist z.B. so ein absoluter Pfadname.

Ein relativer Pfadname bezeichnet den Pfad bezüglich des aktuellen Arbeitsverzeichnisses. Man kann daher das Arbeitsverzeichnis auch dadurch definieren, daß der relative Pfadname der darin enthaltenen Dateien der einfache Dateiname ist. Folgende relative Pfadnamen beziehen sich auf dieselbe Datei (s. Abb. 3).

relativer Pfadname	ausgehend vom Arbeitsverzeichnis
`plot1.ps`	`~arthur/plots`
`plots/plot1.ps`	`~arthur`
`../plots/plot1.ps`	`~arthur/C`

Wenn wir im folgenden nur den einfachen Dateinamen ohne jeden Pfad bezeichnen wollen, werden wir „Basis–Dateiname“ verwenden. Ein Basis–Dateiname enthält keine Schrägstriche.

5.3 Automatische Ergänzung von Dateinamen

Manchmal muß man mehrere Dateien ansprechen, will aber nicht jeden Namen einzeln eingeben, oder man will einen langen Dateinamen abkürzen. Die Shell kann Dateinamen automatisch vervollständigen (Filename Expansion oder File Globbing genannt). Die wesentlichen Mechanismen sind:

- Ein * steht für eine beliebige Anzahl von Zeichen, auch für null Zeichen. In `~arthur/plots` listet `ls *.ps` alle Dateien, die auf `.ps` enden.
- Ein ? steht für genau ein Zeichen. `ls plot?.ps` listet daher `plot1.ps`, `plot2.ps`, `plot3.ps` und `plota.ps`, aber nicht `plot1a.ps`.
- Wenn nur eine Auswahl von Zeichen repräsentiert werden soll, gibt man eine entsprechende Liste oder einen Bereich in eckigen Klammern an. `[ae]` kann für ein a oder ein e stehen, `[a-z]` repräsentiert einen beliebigen Kleinbuchstaben. `ls plot[1-3].ps` listet `plot1.ps`, `plot2.ps` und `plot3.ps`, aber nicht `plota.ps` oder `plot1a.ps`.

Sonderzeichen, die für eine Gruppe von Zeichen stehen, werden oft Wildcards genannt. Speziell ksh und bash bieten umfangreichere Werkzeuge zur Dateinamen-Ergänzung an. Man wende sich mit viel Geduld gewappnet an die entsprechenden Handbücher.

5.4 Löschen und Umbenennen von Dateien

`mv fname1 fname2` nenne Datei `fname1` jetzt `fname2`; eine bereits existierende Datei `fname2` wird dabei gelöscht. Der Name des Kommandos leitet sich von „move“ ab.

`mv fname dir` verschiebe die Datei `fname` in das Verzeichnis `dir`; die Datei hat nachher denselben Basis-Dateinamen, aber einen anderen Pfadnamen. Wenn `dir` bereits eine Datei desselben Basis-Dateinamens enthält, wird diese überschrieben.

`mv dir1 dir2` schiebe das gesamte Verzeichnis `dir1` mit allen Unterverzeichnissen in das Verzeichnis `dir2`. Falls `dir2` schon existiert, wird `dir1` Unterverzeichnis von `dir2`. Falls es noch nicht existiert, wird `dir2` neu erzeugt und ist mit dem alten `dir1` identisch. `dir2` darf kein Unterverzeichnis von `dir1` sein. Auf älteren SYSTEM V-Maschinen kann diese Spielart von `mv` nicht verwendet werden.

`cp fname1 fname2` kopiere die Datei `fname1` nach `fname2`; eine bereits existierende Datei `fname2` wird dabei überschrieben.

`cp fname1 dir` kopiere die Datei `fmane1` in das Verzeichnis `dir` und behalte dabei denselben Basis–Dateinamen.

`cp -r dir1 dir2` kopiere alle Dateien in `dir1` sowie die Unterverzeichnisse rekursiv nach `dir2`, so daß die Datei/Verzeichnis–Struktur erhalten bleibt. Auf älteren SYSTEM V–Maschinen ist dies nicht möglich.

`rm fname` Lösche die Datei `fname`. Im Gegensatz zu MS-DOS ist dieses Kommando unter Unix unwiderruflich!

5.5 Zugriffsrechte und Dateiattribute

Das Kommando `ls -lg plot1.ps` in `~arthur/plots` könnte folgende Ausgabe liefern (auf einem BSD-System):

```
-rwxr-xr--  1 arthur  earth  3451 Jul 25 13.15 plot1.ps
```

Die Option `-g` bewirkt in BSD–Systemen, daß die Gruppenzugehörigkeit der Datei angezeigt wird. In System-V–Systemen geschieht dies bereits bei Verwendung der Option `-l`. Dort wird die Ausgabe der Gruppe durch `-g` abgeschaltet. Das Minuszeichen am Anfang bedeutet, daß es sich bei `plot1.ps` um eine einfache Datei (also kein Verzeichnis etc.) handelt. Bei einem Verzeichnis wäre ein „d" zu sehen. Die nächsten 9 Zeichen erklären die Zugriffsrechte für diese Datei. Nach der Ziffer 1 für die Zahl der Verweise (Links)[3] steht der Besitzer der Datei, dann die Größe in Bytes und das Datum der letzten Änderung. Die letzte Spalte ist der Dateiname.

Drei Dinge kann man mit einer Datei tun: lesen, darauf schreiben und ausführen. Die Zugriffsrechte legen nun fest, wer was mit dieser Datei tun darf. Dabei werden drei Arten von Benutzern unterschieden: der Besitzer `u` (user), die Gruppe `g` (group) und der Rest der Welt `o` (others). Die drei Rechte werden für Lesen mit `r` (read), für Schreiben mit `w` (write) und für Ausführen mit `x` (execute) angegeben. Ein Minuszeichen anstelle eines `r`, `w` oder `x` bedeutet, daß das entsprechende Recht nicht gewährt wird. Die ersten drei Zeichen der Zugriffsrechte zeigen, daß der Besitzer der Datei diese lesen, beschreiben und ausführen darf. Die nächsten drei Zeichen geben die Rechte für die Gruppe (hier `earth`) an. Die Mitglieder der Gruppe dürfen das File lesen und ausführen, aber nicht darauf schreiben. Die letzten drei Zeichen bedeuten, daß die Datei von allen Benutzern, die weder Besitzer noch Gruppenangehörige sind, zwar gelesen, aber weder beschrieben noch ausgeführt werden darf.

Zugriffsrechte können mittels des Kommandos `chmod` geändert werden. Dabei gibt man an, für wen die Rechte geändert werden sollen, nämlich für den Besitzer (u), die Gruppe (g), alle anderen (o) oder alle drei zusammen (a für all). Ein Plus– bzw. Minuszeichen dient der Angabe, ob ein bestimmtes Recht für diese Benutzer hinzugefügt oder entfernt werden soll. Die Rechte selbst werden durch r, w und x bezeichnet. Mit einem `=` wird ein Recht komplett neu gesetzt.

[3] Verweise sind zusätzliche Namen, unter denen eine Datei angesprochen werden kann. Die Zahl 1 bedeutet, daß diese Datei nur einen Namen hat. Siehe auch S. 45.

```
chmod u-x plot1.ps
```

macht die Datei für den Besitzer `arthur` nicht–ausführbar:

```
$ ls -l plot1.ps
-rw-r-xr--  1 arthur     3451 Jul 25 13.15 plot1.ps
```

Das folgende Kommando sorgt dafür, daß die Gruppe die Datei nur noch lesen kann:

```
chmod g=r plot1.ps

$ ls -l plot1.ps
-rw-r--r--  1 arthur     3451 Jul 25 13.15 plot1.ps
```

Man kann die Zugriffsrechte auch über einen Zahlencode angeben. Dabei werden den Buchstaben r, w, x folgende Werte zugeordnet:

```
r 4, w 2, x 1
```

Den korrekten Zahlencode erhält man durch Addieren der Werte der Rechte jeder Benutzergruppe, d.h. der Code ist eine dreistellige Zahl, bei der jede Ziffer zwischen 0 und 7 liegt. `-rw-r--r--` kann somit durch `644` beschrieben werden. Das zuletzt angeführte `chmod` Kommando kann daher durch

```
chmod 644 plot1.ps
```

ersetzt werden.

5.6 Datensicherung

Im Falle der Beschädigung einer Festplatte können die Dateien auf dieser Platte verloren gehen. Deswegen ist es unbedingt notwendig, von Dateien von Zeit zu Zeit eine Sicherungskopie anzulegen, so daß das Versagen einer Festplatte nicht zum Datenverlust führt. Auch versehentliches Löschen führt gelegentlich zu Datenverlust.

An manchen Orten führen die Systemverwalter Sicherungskopien aller Dateien durch. Um aber ganz sicher zu sein (auch Systemverwalter sind Menschen und können die Datensicherung vergessen), sollte man zumindest von wichtigen Dateien eigene Sicherungskopien (Backups) anlegen. Dies kann etwa auf eine Diskette oder eine Bandkassette erfolgen.

Um den Backup bequem erledigen zu können, ist es günstig, sich des `tar` Kommandos zu bedienen, das im Anhang A beschrieben ist. Dieses Kommando kann komplette Verzeichnisse, einschließlich aller Unterverzeichnisse, zu einer Archiv–Datei zusammenfassen.

Um Speicherplatz zu sparen, können die resultierenden `tar` Dateien komprimiert werden, bevor sie auf das externe Speichermedium transportiert werden. Gebräuchlich ist `compress`, aber auch das effizientere `freeze`. Siehe Seite 42 für eine Beschreibung von `compress` und einen Beispiel–Backup; das Kommando `freeze` ist in Kapitel A.14 erläutert. Allerdings besteht bei komprimierten Dateien die Gefahr, daß durch geringfügige Materialfehler auf den Aufzeichnungsmedien die Dateien nicht mehr ent-komprimiert werden können und der Backup somit wertlos wird.

6 Arbeiten mit der Shell

6.1 Standard–Eingabe und Standard–Ausgabe

Viele Programme lesen Eingabeparameter, verarbeiten diese und liefern eine Ausgabe. Daher gibt es in Unix je einen vordefinierten Standard–Eingabe– und Ausgabekanal, die stdin und stdout genannt werden. Beide sind lediglich vordefinierte Dateien. Der Eingabekanal wird i.d.R. mit der Eingabe über die Tastatur identifiziert, der Ausgabekanal mit dem Bildschirm. In einer Umgebung mit verschiedenen logischen Terminals , wie etwa in einer Fensterumgebung, wo jedes Fenster ein eigenes logisches Terminal darstellt, sind stdin und stdout für jedes Fenster extra definiert.

Es gibt noch ein weiteres solches File, das stderr genannt wird und an das Fehlermeldungen weitergegeben werden. Es wird ebenfalls mit dem Terminal identifiziert, ist aber dennoch vom stdout zu unterscheiden. Deswegen können stdout und stderr in verschiedene Dateien umgeleitet werden, wenn man die Ausgabe nicht auf dem Bildschirm wünscht. Verschiedene Dateien sind oft praktisch, wenn die eigentlichen Ergebnisse nicht mit den Fehlermeldungen vermischt werden sollen.

6.2 Umleitung der Standard–Dateien

Umleitung bedeutet, daß den Standard–Dateien eine andere Datei zugeordnet wird, z.B. daß die Eingabeparameter aus einer normalen Datei gelesen werden oder die Ausgabe in eine normale Datei erfolgt. Standard–Eingabe wird mittels des Zeichens `<` umgeleitet.

```
$ Kommando < Eingabedatei
```

liest die Eingabe für `Kommando` aus der Datei `Eingabedatei`. Standard–Ausgabe wird durch ein `>`-Zeichen umgeleitet:

```
$ Kommando > Ausgabedatei
```

schreibt die Ausgabe von `Kommando` in die `Ausgabedatei`. Eine eventuell bereits existierende `Ausgabedatei` wird überschrieben.

Um die Ausgabe an eine Datei anzuhängen statt diese Datei vorher zu löschen, ersetzt man das `>`-Zeichen durch `>>`.

In sh, ksh und bash kann stderr mit `2>` umgeleitet werden. Falls stdout bereits umgeleitet wurde und die Fehlermeldungen in der gleichen Datei gewünscht werden, verwendet man `2>&1`. Beispiele:

```
$ Kommando < infile > outfile 2> errfile
```

verwendet die Eingabedaten aus `infile`, schreibt die Ergebnisse nach `outfile` und die Fehlermeldungen nach `errfile`. Die Reihenfolge der Umleitungen der Standard–Dateien kann ausgetauscht werden.

```
$ Kommando > outfile 2>&1
```

schreibt die Ergebnisse und Fehlermeldungen nach `outfile`. In diesem Fall ist die Reihenfolge der Umleitungen wichtig. Stdout muß umgeleitet werden, bevor stderr in dasselbe File umgeleitet werden kann.

In csh wird `>&` verwendet, um stdout und stderr in eine Datei umzuleiten:

```
$ Kommando >& outfile
```

Getrennte Umleitung von stderr ist in csh nicht auf einfache Weise möglich.

6.3 Pipes

Pipes ermöglichen, daß die Standard–Ausgabe eines Kommandos als Standard–Eingabe eines anderen Kommandos fungiert. Das Symbol für eine Pipe ist `|`. Im folgenden Beispiel will man wissen, ob der Benutzer „arthur" im lokalen Netz eingeloggt ist. Man kann dazu die Ausgabe des `rwho`-Kommandos verwenden, das alle am lokalen Netz eingeloggten Benutzer auflistet. Diese Ausgabe verwendet man dann als Eingabe für das `grep`-Kommando (`grep` sucht nach einem Muster und zeigt alle Zeilen an, die dieses Muster enthalten):

```
rwho | grep arthur
```

zeigt alle Zeilen an, die „arthur" enthalten.

Ein anderes Beispiel ist ein ausführliches Listing (`ls -l`) eines umfangreichen Verzeichnisses. Um sich solch eine Liste seitenweise ansehen zu können, wird die Ausgabe des `ls -l` in das `more` Kommando umgeleitet:

```
ls -l | more
```

6.4 Umgebungsvariablen

Während der Login–Prozedur setzt die Shell verschiedene Umgebungsvariablen. Einige wichtige davon haben wir unten aufgelistet. Um den Wert von `VARIABLE` angezeigt zu bekommen, verwendet man `echo $VARIABLE`. `echo` kopiert seine Argumente auf stdout und das `$`-Zeichen referenziert `VARIABLE`, d.h. es setzt deren Wert ein. Umgebungsvariablen unterscheiden sich von normalen Shellvariablen insbesondere dadurch, daß sie an Kinder der Shell vererbt werden. Daher müssen Umgebungsvariablen entweder mit einem speziellen Befehl gesetzt werden (in csh mit `setenv`) oder explizit für Kindprozesse zugänglich gemacht werden (in sh, ksh und bash durch den Befehl `export`).

`TERM` Der verwendete Terminaltyp, z.B. xterm oder vt100. Diese Variable wird von vielen Programmen verwendet, um die richtigen Kontrollzeichen etc. benutzen zu können.

`DISPLAY` Das Display; diese Variable ist wichtig, wenn X verwendet wird (siehe Kapitel 11) .

`PRINTER` Der voreingestellte Drucker; wird z.B. von `lpr` verwendet (siehe Anhang A).

`PATH` Suchpfad, d.i. eine Liste von Verzeichnissen, in denen die Shell nach Kommandos sucht.

`PS1` Der Ausdruck, der in sh, ksh und bash als (primärer) Prompt verwendet wird.

`prompt` Der Ausdruck, der in csh als (primärer) Prompt verwendet wird.

`HOME` Das Homedirectory, in dem man nach dem Login landet.

Um Umgebungsvariablen zu ändern, verwende man

- in csh und tcsh: `setenv VARIABLE value`
- in ksh und bash: `VARIABLE=value; export VARIABLE`

Der Suchpfad `PATH` ist von den auf Seite 11 beschriebenen Pfadnamen zu unterscheiden. Ein absoluter oder relativer Pfadname kennzeichnet eine Datei oder ein Verzeichnis. Der Suchpfad ist eine Liste von Pfadnamen von Verzeichnissen, die durchsucht werden sollen, wenn ein Kommando mittels eines Basis–Dateinamens aufgerufen wird. Um ein weiteres Verzeichnis dem Suchpfad anzugliedern, z.B. weil der Benutzer arthur sein `bin`–Unterverzeichnis im Suchpfad haben möchte, verwendet man:

- in csh: `setenv PATH ${PATH}:$HOME/bin`
- in sh, ksh und bash: `PATH=${PATH}:$HOME/bin; export PATH`

Im Suchpfad wird das aktuelle Verzeichnis durch einen Punkt angegeben.

Derartige Erweiterungen zum Suchpfad können in eine der Startup–Dateien geschrieben werden, so daß sie bei jedem Einloggen ausgeführt werden. Damit spart man sich das Tippen ganzer Pfadnamen, sofern das Kommando, das man ausführen will, in einem der in `PATH` gelisteten Verzeichnisse auftaucht. Gibt man ein Kommando mit einem Basis–Dateinamen so sucht die Shell in allen Verzeichnissen des Suchpfades nach diesem Kommando. Dies geschieht genau in der in `PATH` spezifizierten Reihenfolge. Falls die Shell in keinem dieser Verzeichnisse das gesuchte Kommando findet, so wird die Fehlermeldung `command not found` ausgegeben. Um ein Kommando aufzurufen, das sich in keinem der Suchpfadverzeichnisse befindet, muß der volle oder der relative Pfadname angegeben werden. Um alle Verzeichnisse zu sehen, die sich momentan im Suchpfad befinden, tippt man `echo $PATH`.

6.5 Aliase

Häufig gebrauchte Shell–Kommandos will man gerne abkürzen. Zu diesem Zweck können Aliase definiert werden. Besonders bequem ist es, diese Aliase in die zweite der Startup–Dateien zu schreiben. Im folgenden Beispiel definieren wir das Alias `ll` für das Kommando `ls -l`.

- `alias ll 'ls -l'` in csh
- `alias ll='ls -l'` in ksh und bash

Bourne Shell (sh) erlaubt keine Aliase. Neuere Versionen von sh ermöglichen jedoch die Definition von Funktionen, die Aliase ersetzen können. Die richtige Anwendung von Anführungszeichen in der Shell ist nicht einfach. Bei Problemen kann man alle notwendige Information in den Manualen finden (wenn auch oft nicht besonders übersichtlich). Die oben angegebenen Formen sollten eigentlich in den meisten Fällen gut funktionieren.

6.6 Verwendung von Shell–Sonderzeichen als normale Zeichen

Manchmal werden in den Argumenten einer Kommandozeile Zeichen benötigt, die für die Shell eine besondere Bedeutung haben. Da man aber eigentlich nicht die besondere Interpretation durch die Shell wünscht, müssen solche Zeichen zitiert werden. Zum Beispiel taucht dieses Problem auf, wenn man in einer Datei namens „Telefon“ nach „Huber&Sohn“ suchen will.

```
grep Huber&Sohn Telefon
```

funktioniert aber nicht, da das `&` Zeichen der Shell mitteilt, daß das Kommando hier zu Ende ist und im Hintergrund bearbeitet werden soll. Der Rest der Zeile, nämlich `Sohn Telefon` wird als ein weiteres Kommando interpretiert. Um der Interpretation des `&`-Zeichens durch die Shell zu entkommen, kann man zwischen den folgenden Möglichkeiten wählen:

- Stecke den gesuchten Ausdruck in doppelte Anführungszeichen. Alles in doppelten Anführungszeichen wird wörtlich interpretiert. Referenzen zu Shellvariablen werden aber dennoch erkannt. Daher funktioniert diese Methode nicht mit $-Zeichen im Such-Ausdruck. Dateinamen–Ergänzung (File–Globbing) wird nicht durchgeführt.

- Stecke den gesuchten Ausdruck in einfache Anführungszeichen. Im Gegensatz zu den doppelten Anführungszeichen werden hier Referenzen zu Shellvariablen nicht erkannt. Dateinamen-Ergänzung wird ebenfalls nicht durchgeführt.

- Man kann das `&`-Zeichen direkt zitieren, d.h. man schreibt das Escape–Zeichen der Shell, den Backslash `\` davor. Das Escape–Zeichen bewahrt das direkt folgende Zeichen vor der Interpretation durch die Shell. Dieses Zeichen hat aber nichts mit der `ESC`–Taste zu tun!

Damit das Beispiel von oben funktioniert, kann man demnach verwenden:

```
grep "Huber&Sohn" Telefon
grep 'Huber&Sohn' Telefon
grep Huber\&Sohn Telefon
```

6.7 Abhilfe in Notfällen

Wenn ein Prozeß definitiv etwas ganz anderes tut, als eigentlich beim Start bezweckt, gibt es mehrere Möglichkeiten, darauf zu reagieren:

- Abwarten und Tee trinken, sofern der Prozeß nicht destruktiv ist oder viel Rechenzeit verbraucht.

- Unterbrechen des Prozesses durch `C-c`. Dies unterbricht einfache Kommandos, andere können etwa zu ihrer Eingabe–Ebene o.ä. zurückkehren.

- Falls das Programm `C-c` ignoriert, kann man es mit `C-\` versuchen. Dies ist eine stärkere Anfrage an die Maschine als `C-c`, den Prozeß zu unterbrechen und produziert normalerweise einen core dump (deswegen findet man nachher eine

Datei namens `core` im aktuellen Verzeichnis des unterbrochenen Programms. Diese Datei beschreibt den Zustand des Programms beim Abbruch und kann zur Fehlersuche mit einem Debugger benutzt werden).

- Kille den Prozeß von einer anderen Shell aus. Zunächst muß man mit `ps ux` (`ps ef` in SYSTEM V) eine Liste aller aktiven Prozesse des Benutzers erzeugen und dort die Prozeß–Identifikationsnummer (`PID`) des ungehörigen Prozesses suchen. Mit `kill PID` kann der Prozeß dann terminiert werden.
- Falls Sie ohne Fenster–System arbeiten, sollten Sie den Prozeß erst mit `C-z` stoppen (wenn es ein BSD–System ist). Unmittelbar danach kann mit `kill %` der soeben gestoppte Prozeß terminiert werden. Wurden mehrere Prozesse gestoppt, so muß die entsprechende Jobnummer nach dem `%`-Zeichen eingegeben werden. Die Jobnummer wird von `jobs` angezeigt.
- Kille den Prozeß *wirklich*. Falls das einfache kill–Kommando wie soeben beschrieben nicht funktioniert hat, wiederholt man das Ganze mit `kill -9 PID` oder `kill -9 %`.

Falls sich das Terminal sehr eigenartig verhält, etwa Eingaben nicht anzeigt oder komische Zeichen ausgibt, muß man die Terminalparameter wieder richtig setzen. Ein derartiges Verhalten kann z.B. das Resultat eines vi–Absturzes sein. Man gebe dann `C-jstty saneC-j` ein. Es kann sein, daß diese Eingabe nicht am Bildschirm erscheint. Die beiden `C-j` ersetzen dabei `RETURN`. Einige Systeme wollen anstelle des `stty sane` ein `reset` sehen.

6.8 Shell–Skripts

Die Shell kann auch als Programmiersprache verwendet werden. Konstruktionen wie Schleifen, Bedingungen etc. sind dazu vorhanden. Die Kommandos sind für die einzelnen Shells verschieden, aber sh, ksh und bash haben eine ziemlich große gemeinsame Schnittmenge. Die exakte Syntax findet man in den Manualseiten zu den Shells. Falls man Shell–Skripts schreiben will, ist die Verwendung von sh unbedingt zu empfehlen, da sh garantiert in der Minimalversion auf jeder Unix–Maschine vorhanden ist, so daß sh–Shell–Skripts meistens problemlos auf andere Maschinen portiert werden können.

Shell–Skripts können auch einfach dazu dienen, häufig verwendete Kommandofolgen in eine Datei zu schreiben, denn alles, was in einer Kommandozeile stehen kann, kann auch in einem Shell–Skript stehen. Angenommen, man hat eine solche Datei, so gibt es zwei Möglichkeiten, diese als Kommando aufzurufen:

- Man ruft das Skript explizit mit der zugehörigen Shell in der Kommandozeile auf:

  ```
  sh Skript
  ```

  Dazu muß die Datei `Skript` lesbar sein.
- Schreibe in `Skript` folgende erste Zeile:

  ```
  #!/bin/sh
  ```

> `Skript` kann nun durch einfaches Angeben des Namens in der Kommandozeile aufgerufen werden. Dazu muß `Skript` allerdings ausführbar sein. Sind die ersten Zeichen einer Datei `#!`, so erwartet das System als nächstes den Pfadnamen eines Interpreters, z.B. einer Shell, und verwendet diesen Interpreter, um die Datei abzuarbeiten. Auf einigen sehr alten SYSTEM–V–Maschinen funktioniert diese Methode nicht.

Es folgen einige Beispiele. Das erste verwendet das `find`–Programm. Um dieses Skript zu verstehen, ist ein Blick auf die Manualseite für `find` sicher von Nutzen.

Das folgende Skript durchsucht das Homedirectory und dessen Unterverzeichnisse nach Dateien, die wahrscheinlich nicht mehr gebraucht werden, wie etwa Sicherungsdateien des emacs–Editors (diese enden auf eine Tilde ~), `.dvi`, `.aux` oder `.log` Dateien aus LaTeX–Läufen sowie core–Dateien. Core–Dateien werden erzeugt, wenn ein Programm wegen eines schwerwiegenden Laufzeitfehlers abgebrochen werden mußte. Diese Dateien sind oft sehr groß und können daher viel Plattenplatz verbrauchen. Das gelegentliche Durchführen einer solchen Aufräumaktion hilft, weniger Plattenplatz zu beanspruchen.

```
#!/bin/sh
echo Beginne mit Aufr"aumen...
cd
find . \( -name '*~' -o -name '*.log' -o -name '*.dvi' -o \
       -name '*.aux' -o -name core \) -ok rm {} \;
echo Aufger"aumt!
```

Das erste `cd` stellt sicher, daß man sich zu Beginn des Kommandos im Homedirectory befindet. Das `find`–Kommando startet im aktuellen Arbeitsverzeichnis (also dem Homedirectory), das durch den Punkt dargestellt wird. Es sucht dort und in allen Unterverzeichnissen nach Dateinamen mit den Eigenschaften, die in den zitierten Klammern in Anführungszeichen angegeben werden. Das `-name` bedeutet, daß nach dem entsprechenden Namen gesucht werden soll, das `-o` ist ein logisches Oder. Die Ausdrücke mit den Wildcards müssen in Anführungszeichen stehen, um einer Interpretation durch die Shell vorzubeugen. Das `-ok rm {} \;` bedeutet, daß `find` jede gefundene Datei löschen soll, aber vorher jeweils eine Bestätigung verlangen muß. Die beiden `echo`–Kommandos geben informative Meldungen aus.

Es folgt ein Skript, das eine Archivdatei der `tex`- und `C`- Unterverzeichnisse eines Homedirectories anlegen und komprimieren soll. Eine derartige Datei kann dann als Sicherungsdatei auf einer Diskette oder einer Bandkassette gespeichert werden.

```
#!/bin/sh

cd
tar cvf - tex C | compress > backup.tar.Z
echo backup file: backup.tar.Z
```

Derartige einfache Shell–Skripts sind besonders bei komplizierten Kommandos nützlich. Das letzte Beispiel faßt einige der typischen Morgenaktivitäten einer typischen Benutzerin zusammen und kommentiert diese mit aufmunternden Bemerkungen :-)[4]. Die # Zeichen leiten Kommentare ein, d.h. der Rest der Zeile wird von der Shell ignoriert. Das Beispiel enthält auch eine `for`–Schleife.

```
#!/bin/sh

for i in 1 2 3      # eine for-Schleife in sh von 1 bis 3
do                  # gehoert zur for-Schleife
echo Guten Morgen!!!# gibt "Guten Morgen!" am Bildschirm aus
sleep 2             # wartet 2 Sekunden (zum Aufwachen)
done                # beendet die for-Schleife
echo                # gibt eine Leerzeile aus
echo -n Heute ist   # das -n verhindert einen Zeilenumbruch
                    # (nur BSD)
date                # gibt Datum und Zeit aus
echo
echo Wer ist hier denn sonst noch um diese fruehe Stunde???
rwho | more          # sieht nach anderen Benutzern im Netz
echo
echo Und wie geht es dieser Maschine??
uptime               # zeigt den Status der Maschine
echo
echo Was haben wir heute zu tun?
cat ~/TO_DO          # viele Leute schreiben ihre Aufgaben
                     # in eine Datei wie TO_DO
echo Ey, jetzt schau nicht so verschlafen aus der Waesche!!
```

[4] Falls die Bedeutung der letzten drei Zeichen unklar sein sollte: das ist ein Smiley! Einfach den Kopf nach links neigen —na? Diese Smileys werden in elektronischer Post und in Netnews viel verwendet, um Ironie etc. für den Fall zu verdeutlichen, daß es der Leser anders nicht verstanden hätte.

7 Editieren von Dateien

Auf den meisten Unix–Systemen stehen zwei komfortable Editoren zur Verfügung: `vi` und `emacs`. Neben diesen beiden gibt es auch noch den zeilenorientierten Editor `ed`. Um eine Datei nicht–interaktiv zu bearbeiten, können `sed`, `awk` und `perl` verwendet werden.

Um eine Datei nur anzusehen (ohne sie ändern zu können), genügt das `cat` Kommando.

```
cat Dateiname
```

stellt die Datei `Dateiname` auf einmal am Bildschirm dar. Um diese Datei Seite für Seite zu zeigen, verwendet man

```
more Dateiname
```

7.1 vi

`vi` ist immer noch der am meisten verwendete Editor unter Unix, obwohl er nicht so vielseitig ist wie `emacs`. Leider gibt es keine gute online–Manualseite, die gedruckten Systemhandbücher enthalten aber meist eine brauchbare Beschreibung von `vi`. Zum schnellen Nachschlagen ist die Referenzkarte im Anhang dieser Einführung geeignet.

`vi` wird durch den Aufruf `vi Dateiname` gestartet. Beschwert sich `vi` darüber, daß der Terminaltyp nicht bekannt ist, so verläßt man `vi` erst einmal durch Eingabe von `:q!` und setzt die Umgebungsvariable `TERM` (siehe vorangegangenes Kapitel). Falls Ihnen Ihr Terminaltyp nicht bekannt ist, verwenden Sie einfach den „Standardtyp" `vt100`.

`vi` kennt drei verschiedene Modi: Eingabemodus (Insert Mode), Kommandomodus (Command Mode) und den Letzte–Zeile–Modus (auch `ex`–Modus oder :-Modus genannt).

Nach dem Aufruf von `vi` befindet man sich im Kommandomodus, d.h. man kann Kommandos, aber keinen Text eingeben. Um in den Eingabemodus zu wechseln, können folgende Kommandos verwendet werden:

- `i` Eingabe vor der aktuellen Cursorposition
- `a` Eingabe direkt hinter der aktuellen Cursorposition
- `o` öffne eine neue Zeile unterhalb der derzeitigen Cursorposition für die Eingabe

Weitere Kommandos für die Eingabe von Text stehen im Anhang C.

Um wieder zurück in den Kommandomodus zu kommen, was z.B. zum Bewegen des Cursors notwendig ist, drückt man die Taste `ESC`. Falls man vergessen hat, in welchem Modus man sich gerade befindet, hilft auch ein Druck auf die `ESC` Taste, denn anschließend befindet man sich sicher im Kommandomodus: war vorher Eingabemodus eingestellt, so wird in den Kommandomodus gewechselt, war bereits der Kommandomodus eingestellt, so erklingt ein akustisches Signal und `vi` bleibt im Kommandomodus.

Der Kommandomodus dient der Eingabe einfacher Kommandos, z.B. dem Löschen einer Zeile oder Cursorbewegungen. Diese Kommandos bestehen immer nur aus wenigen Buchstaben, die nirgendwo angezeigt werden. Kompliziertere Kommandos, wie z.B. das Ersetzen von Mustern im Text oder das Einlesen eines weiteren Files können mit diesen einfachen Kommandos nicht bewältigt werden. Dafür gibt es den Letzte–Zeile–Modus. Um in diesen Modus zu wechseln, tippt man einen Doppelpunkt (daher der Name :-Modus). Damit wird in der untersten Zeile eine Kommandozeile zur Eingabe geöffnet. Alles bis zum nächsten `RETURN` wird dann als Kommando interpretiert. Diese Kommandos unterscheiden sich aber von denen im einfachen Kommandomodus. Um die Unterscheidung zu erleichtern, leiten wir in dieser Einführung alle Kommandos für den Letzte–Zeile–Modus mit dem Doppelpunkt ein, der diesen Kommandos ja stets vorausgeht.

`vi` unterscheidet Groß– und Kleinschreibung. Da die Kommandos im Kommandomodus bei der Eingabe nicht gesehen werden, hilft es oft, die Position der `CAPS LOCK`–Taste zu kontrollieren, falls sich `vi` beharrlich weigert, Ihre Befehle zu verstehen.

Eine Tilde am Zeilenanfang bedeutet, daß diese Zeile nicht mehr zur Datei gehört. Dies erleichtert das Erkennen des Dateiendes.

`vi` wird durch `ZZ` oder `:wq` verlassen, wobei alle an der Datei vorgenommenen Änderungen abgespeichert werden. Falls `vi` verlassen werden soll, ohne die Änderungen abzuspeichern, so verwendet man `:q!`.

Die wichtigsten Editierkommandos in `vi` sind im Anhang C zusammengestellt.

7.2 Emacs

`Emacs` ist wesentlich mehr als nur ein Editor. Er ist eine Programmierumgebung, kann zum Lesen von E-Mail verwendet werden und hat ein komfortables Informationssystem. Bei regelmäßiger Benutzung wird man viele weitere Möglichkeiten zur Verwendung von `emacs` entdecken. Es soll Leute geben, die ihren `emacs` über Wochen hinweg nicht verlassen (außer zum Kaffeekochen, das kann `emacs` leider immer noch nicht).

Die Beschreibung von Emacs in dieser Einführung bezieht sich auf den Emacs der Free Software Foundation.[5] Für andere Varianten von Emacs konsultiere man die entsprechenden Handbücher.

Um `emacs` zu starten, tippt man einfach `emacs`. Geschieht das zum ersten Mal, so sollte unbedingt mit `C-h t` das Tutorial aufgerufen werden. In einem ca. halbstündigen Kurs lernt man damit die Grundzüge von `emacs` kennen. Einen Überblick über die wichtigsten Kommandos in `emacs` bietet die Referenzkarte im Anhang B.

Mit dem Kommando `C-x C-f`, gefolgt vom gewünschten Dateinamen, werden Dateien eingelesen. `Emacs` achtet dabei auf besondere Endungen der Dateinamen, wie `.c` oder `.tex`, und startet für diese Dateien einen passenden Modus. Diese Modi erleichtern das Editieren sehr. Der C–Modus rückt z.B. den Code automatisch passend ein, während man tippt, der TEX–Modus erlaubt es, zueinander gehörende `$`–Zeichen und Klammern zu finden (es gibt spezielle Modi für LATEX, die bequeme

[5] Für die Free Software Foundation FSF (manchmal GNU-Projekt genannt) arbeiten Mitarbeiter wissenschaftlicher und kommerzieller Organisationen. GNU-Software ist kostenlos erhältlich und wird im Quellcode verteilt. Die Software unterliegt speziellen Copyrightbestimmungen. Die FSF arbeitet langfristig an einem freien Unix.

Abkürzungen für viele LaTeX–Kommandos bereitstellen). Information über den aktuellen Modus erhält man mit `C-h m`. In `emacs` können mehrere Dateien gleichzeitig editiert werden. Jeder Editiervorgang erhält einen eigenen Pufferbereich. Zwischen den einzelnen Puffern kann beliebig gewechselt und es können Teile hin– und herkopiert werden.

Das Informationssystem von `emacs` wird durch `C-h i` gestartet. Die wichtigsten Kommandos werden angezeigt. Es gibt dort ein Tutorial über das Informationssystem selbst. Des weiteren enthält das Info–System meist ausführliche Dokumentation zu `emacs` und anderer Software der FSF.

Um ein editiertes File abzuspeichern, verwendet man `C-x C-s`. Verlassen wird `emacs` mit `C-x C-c`. Dabei wird für jedes editierte File gefragt, ob die Änderungen abgespeichert werden sollen. Für den Fall eines System–Crashes legt `emacs` während des Editierens Sicherheitskopien an. Diese heißen `#filename#`, wobei `filename` der Name der editierten Datei ist. Nach explizitem Abspeichern der betreffenden Datei wird die Sicherungskopie gelöscht. Falls es einmal nötig sein sollte, eine derartige Sicherungskopie über die Shell anzusprechen, müssen die `#`-Zeichen in Anführungszeichen gesetzt werden, da sie für die Shell Kommentarsymbole darstellen. Zusätzlich zu diesen Sicherheitskopien erzeugt `emacs` eine Kopie der vorletzten explizit abgespeicherten Version des editierten Files. Diese Backup–Dateien heißen `filename~`. Sie sind nützlich, wenn im Übereifer in einer Datei zuviel gelöscht und diese Version auch noch abgespeichert wurde. Kleinere, noch nicht explizit abgespeicherte Malheurs sind einfach durch wiederholte Anwendung der Undo-Funktion `C-x u` oder `C-_` zu beheben.

8 Compiler

Die gebräuchlichsten Compiler unter Unix sind:

- `cc` Der C–Compiler; häufig in einer ANSI- und einer nicht–ANSI–Version vorhanden (ANSI-C ist standardisiertes C).
- `gcc` Der GNU C–Compiler; `gcc` kennt ANSI- und nicht–ANSI–C. Siehe auch den Info–Eintrag im `emacs`.
- `CC` C++ Compiler (kann auch andere Namen haben)
- `g++` Der GNU C++ Compiler; er ist mit dem `gcc` kombiniert.
- `f77` Der FORTRAN–Compiler, meistens deutlich erweitertes Fortran 77. Fortran 90 Compiler sind noch sehr selten; sie werden oft durch `f90` aufgerufen.
- `pc` Der Pascal–Compiler (kann anders heißen); welchen der vielen Pascal–Dialekte Ihr Compiler versteht, ist (hoffentlich) den Manualen zu entnehmen.

Der Compilier–Prozeß durchläuft folgende Stufen:

1. Präprozessorlauf. Alle Kommentare werden entfernt. Präprozessoranweisungen werden eingesetzt, sofern der Compiler diese unterstützt (z.B. C–Compiler).

2. Übersetzen.
3. Optimieren, sofern dies beim Compileraufruf gewünscht wurde. Nach Compilation und Optimierung liegt ein Assembler–Programm vor.
4. Assemblieren. Dies übersetzt das Assembler–Programm in eine maschinenlesbare Objektdatei.
5. Binden. Alle erzeugten Objektdateien und notwendigen bzw. angeforderten Bibliotheken werden zusammen zu einer ausführbaren Datei gebunden.

Folgende Dateiendungen werden üblicherweise vom jeweiligen Compiler erkannt:

.c C–Quellcode–Datei

.cc C++ Quellcode–Datei (manchmal auch auf .C endend)

.f FORTRAN–Quellcode–Datei (manchmal auch .f77 oder .for)

.p Pascal–Quellcode–Datei (manchmal auch .pas)

.s Assembler–Datei

.o compilierte (aber noch nicht gebundene) Objektdatei

.i C–Quellcode–Datei nach dem Präprozessorlauf (wird durch Aufruf des Compilers mit -P erzeugt)

Folgende Optionen werden von den meisten Compilern erkannt. Compileroptionen müssen getrennt angegeben werden, jede einzelne mit einem Minuszeichen davor. Die Optionen werden durch ein Leerzeichen getrennt. Wegen der genauen Anordnung der Optionen und der erwarteten Argumente ist ein Blick auf die Manualseiten erforderlich. Nach der Auflistung der Optionen geben wir einige typische Beispiele an.

`-O` Erzeugt optimierten Code; das so generierte ausführbare File ist wesentlich schneller. Oft kann zwischen verschiedenen Optimierungsstufen gewählt werden. Allerdings können die Optimierungsalgorithmen eines Compilers fehlerhaft sein. Deswegen sollte die Optimierung während der Entwicklungsphase eines Programmes eher vorsichtig oder gar nicht eingesetzt werden.

`-g` Erzeugt zusätzliche Informationen, die benötigt werden, um ein Programm mit einem Quellcode–Debugger zu untersuchen (siehe weiter unten).

`-c` Die Datei wird compiliert, aber nicht gebunden. Die resultierende Objektdatei hat die Endung `.o`.

`-o Name` Nennt das ausführbare File `Name` und nicht `a.out`, wie der voreingestellte Wert lautet.

`-l Bibliothek` Bindet die Objektdatei mit der angegebenen Bibliothek zusammen. Will man z.B. mathematische Funktionen verwenden, so müssen die Objektdateien mit der Mathematik–Bibliothek `libm.a` gebunden werden. Da alle Bibliotheken Namen der Form `libx.a`

> haben, muß nur der Teil *x* beim Aufruf unmittelbar nach dem `-l` angegeben werden. Deswegen bindet man die Mathematik–Bibliothek mit der Option `-lm` (siehe auch das Beispiel weiter unten). Bei der Angabe mehrerer Bibliotheken, die voneinander abhängen, ist darauf zu achten, daß eine Bibliothek, die Information einer anderen Bibliothek verwendet, vor dieser dazugebunden wird.

Um die Datei `erster_Versuch.c` mit `cc` zu übersetzen, wobei optimiert werden soll, aber nicht gebunden, gibt man an:

```
cc -O -c erster_Versuch.c
```

Um die Datei `expo.c` mit `gcc` zu übersetzen, wobei neben der Anlage von Symboltabellen die Mathematik–Bibliothek dazugebunden werden und die ausführbare Datei `expo` heißen soll, verwendet man:

```
gcc -g -o expo expo.c -lm
```

In der Regel tut ein Programm beim ersten Testlauf nicht genau das, was von ihm erwartet wird[6]. Um rein syntaktische Fehler in einem C–Programm zu finden, verwendet man den Syntaxprüfer `lint`. Ähnliche Prüfprogramme gibt es auch für andere Programmiersprachen, allerdings sind sie z.B. für FORTRAN leider nicht so verbreitet.

Ein Quellcode–Debugger erlaubt es, Schritt für Schritt durch ein Programm zu laufen, die Werte von Parametern zu überprüfen usw., während gleichzeitig der entsprechende Quellcode angezeigt wird. Leider unterstützen viele Debugger auf Unix–Systemen nur C, allerdings scheint sich das langsam zu verbessern. Auf den meisten Unix–Systemen ist der Debugger `dbx` verfügbar,[7] auf manchen auch nur der weniger mächtige `sdb`. Sehr gute Debugger sind die frei erhältlichen Debugger `gdb` und `ups` sowie einige kommerzielle Produkte.

Um diese Quellcode–Debugger verwenden zu können, muß das zu untersuchende Programm mit der `-g` Option übersetzt worden sein. Viele Compiler lassen die gleichzeitige Verwendung der Optionen `-g` und `-O` nicht zu. Da aber ein Optimierer ohnehin den Fluß eines Programmes sehr verändern kann, ist dies auch nur in wenigen Fällen wirklich wünschenswert.

Um die Compilierung einfacher und angenehmer zu machen, gibt es das Programm `make` (siehe Anhang A). Mit `make` kann das Übersetzen weitgehend automatisiert werden, was insbesondere bei größeren Programmierprojekten essentiell ist.

---

[6] Manche Leute meinen, das wäre eine pessimistische Haltung; ich jedenfalls traue einem Programm überhaupt nicht, das beim ersten Mal wirklich zu tun scheint, wofür es geschrieben wurde. Es liegt ganz sicher nur ein besonders gut versteckter und hinterhältiger Fehler vor (Eine Abwandlung von Murphy's Gesetzen...)!

[7] Die Hersteller von Betriebssystemen gehen leider teilweise dazu über, die Unix–typische Entwicklungsumgebung nicht mehr mit dem Betriebssystem auszuliefern, sondern gesondert zu verkaufen. Falls Ihr System dazu gehört, müssen Sie entweder die entsprechenden Entwicklungspakete kaufen oder die frei erhältlichen Pakete (z.B. der FSF) auf Ihren Anlagen installieren oder installieren lassen.

Um die Entwicklung mehrerer Programmversionen unter Kontrolle zu halten, existieren Programme wie `SCCS` oder `RCS`. Damit können ältere Versionen von Programmen platzsparend gespeichert und leicht wiederhergestellt werden. Diese Programme sind nicht auf allen Systemen verfügbar.

9 Arbeiten mit dem Netzwerk

Das Netz wurde in dieser Einführung ja schon einige Male erwähnt: man kann sich darüber an einer entfernten Maschine einloggen etc.

Es gibt verschiedene Arten von Netzwerken. Die Unterschiede zwischen den einzelnen Netzen basieren dabei weniger auf unterschiedlichen physikalischen Kabeln, sondern auf den verschiedenen Protokollen, die in diesen Netzen gesprochen werden. Einige bekannte Netze sind BITNET, HEPNET, SPAN (die beiden letzten sind DECnet–Netze) und natürlich das Internet mit der TCP/IP–Protokollsuite. Es gibt eine Standard–Protokollfamilie, nämlich OSI; sie ist jedoch weltweit noch nicht besonders verbreitet. Derzeit ist TCP/IP am meisten verbreitet, was nicht zuletzt daran liegt, daß jedes moderne Unix eine TCP/IP–Implementation enthält. In den folgenden Kapiteln beschränken wir uns daher auf Anwendungen, die auf der TCP/IP–Protokollsuite aufbauen.

9.1 Domain–Namen

Netzwerk–Anwendungen verlangen, daß eine Maschine beim Namen genannt wird. Jede Maschine hat einen eindeutigen Namen, den sog. Domain–Namen. Diese Namen haben folgende Form:

```
Machine.Subdomain1....Subdomainn.Top-Level-Domain
```

Er besteht aus dem eigentlichen Maschinennamen und diversen Subdomains, alles voneinander durch Punkte getrennt. Die letzte Domain in diesem Namen ist die Top–Level–Domain. Jedem Land, das am Domainnamen–System teilnimmt, wird eine solche Top–Level–Domain zugewiesen, üblicherweise der ISO-2–Buchstaben–Ländercode. Zusätzlich gibt es sieben Top–Level–Domains, die eher organisatorische Gesichtspunkte berücksichtigen:

EDU Institutionen aus Forschung und Lehre

COM kommerzielle Institutionen

ORG nicht–kommerzielle Organisationen

GOV Regierungseinrichtungen; das können auch nationale Forschungslaboratorien sein

MIL militärische Einrichtungen

NET größere Einrichtungen von Netzwerkbetreibern

ARPA historisch; wird kaum verwendet

Einige der Ländercodes sind

AU	Australien	AT	Österreich
CH	Schweiz	DE	Deutschland
DK	Dänemark	FI	Finnland
FR	Frankreich	IL	Israel
IT	Italien	JP	Japan
KR	Südkorea	MX	Mexiko
NL	Niederlande	NO	Norwegen
SE	Schweden	UK	Großbritannien

Die USA haben zwar den Ländercode `US`, er wird aber fast nicht verwendet, da sich die USA hauptsächlich der sieben organisatorischen Domains bedienen. Diese Domains werden aber auch von anderen Ländern genutzt.

Für lokale Netzwerke werden oft Aliasnamen definiert, damit beim Adressieren einer lokalen Maschine nicht immer der lange Domainname getippt werden muß. So könnte die Maschine

```
Herz_aus_Gold.Magrathea.Universum
```

im lokalen Netz als `Herz_aus_Gold` angesprochen werden.

9.2 Login auf entfernten Maschinen

Die beiden Programme für entfernten Login, `telnet` und `rlogin`, wurden bereits in Kapitel 2.3 besprochen. Hier wollen wir daher nur auf die Hauptunterschiede zwischen den beiden eingehen.

- `rlogin` funktioniert nur zwischen Unix–Maschinen. Um sich an einer nicht–Unix–Maschine einzuloggen, braucht man `telnet`.
- Um zu sehr weit entfernten Maschinen eine Verbindung aufzubauen, sollte `telnet` verwendet werden. In der Regel wird bei solch einer Verbindung nämlich jeder einzelne Buchstabe in ein Datenpaket verpackt (das wesentlich mehr Platz braucht als der einzelne Buchstabe, da dort zusätzlich Informationen über Absender, Adressat etc. enthalten sind) und übers Netz geschickt. Es wäre natürlich viel ökonomischer, jeweils eine ganze Zeile zu verschicken. Wenn die Leitung schlecht ist (was in Deutschland außerhalb lokaler Netze fast überall der Fall ist, und damit natürlich auch bei Verbindungen in andere Länder), wird dies eine essentielle Frage. Das Verschicken der Einzelbuchstaben belastet dann die ohnehin schon überlasteten Strecken unnötig. `telnet` ermöglicht das Verschicken ganzer Zeilen im sog. Zeilenmodus, d.h. die ganze Zeile wird auf der lokalen Maschine bearbeitet (also auch eventuelle Korrekturen) und nach der Eingabe von `RETURN` verschickt. Um in den Zeilenmodus zu gelangen, startet man `telnet` und tippt das Escape–Zeichen von `telnet`, meist `C-]`, gefolgt vom Kommando `mode line`. Dieser Modus kann aber nicht für Editoren verwendet werden, da diese auf buchstabenweise Verarbeitung der Eingabe angewiesen sind. Um zurück in den buchstabenorientierten Modus zu schalten, wird das Escape–Zeichen gefolgt von `mode character` angegeben.

- Mit `rlogin` kann man sich an anderen Maschinen einloggen, ohne ein Paßwort anzugeben, indem man in eine Datei namens `.rhosts`, die im Homedirectory anzulegen ist, die Maschinen einträgt, von denen aus ein `rlogin` ohne Paßwort gestattet wird. Aus Sicherheitsgründen wird dringend empfohlen, diesen Mechanismus so wenig wie möglich zu verwenden: wird nämlich eine der „vertrauenswürdigen" Kennungen von einem Eindringling geknackt, so sind damit auch alle Kennungen, die `rlogin` ohne Paßwort von dieser Maschine aus erlauben, für den Eindringling offen. Eine `.rhosts` Datei ist auch dann nicht akzeptabel, wenn sie sich auf Maschinen eines lokalen Netzes beschränkt, auf denen man ohnehin automatisch dasselbe Paßwort hat. Auf jeden Fall sollte die Datei `.rhosts` die Zugriffsberechtigung `400` haben, d.h. nur für den Benutzer selbst lesbar.

- Mit `telnet` kann man noch eine ganze Menge mehr tun als sich lediglich auf entfernten Maschinen einloggen (siehe die Manualseiten zu `telnet`).

9.3 Übertragung von Dateien

Es gibt zwei Programme zur Dateiübertragung: `rcp` und `ftp`. Die Verwendung von `rcp` wird nicht empfohlen, da es ein `.rhosts` File mit den entsprechenden Einträgen benötigt. Eine `ftp`–Sitzung des Benutzers `arthur` könnte so beginnen:

```
$ ftp Herz_aus_Gold.Magrathea.Universum
Connected to Herz_aus_Gold.Magrathea.Universum
220 Herz_aus_Gold FTP server (UNIVERSE OS 77.1) ready.
Name (Herz_aus_Gold.Magrathea.Universum:arthur): arthur
331 Password required for arthur.
Password:
230 User arthur logged in.
ftp>
```

Der Name der Maschine, mit der Verbindung aufgenommen werden soll, wird als Argument zum `ftp`–Kommando gegeben. Anschließend muß der Loginname auf dieser Maschine und das zugehörige Paßwort angegeben werden. Danach erwartet `ftp` ftp–Kommandos. Ein `?` listet die möglichen Kommandos auf. Einige wichtige davon sind

`ls` oder `dir` Liste das Verzeichnis auf der entfernten Maschine.

`cd dirname` Wechsle ins Verzeichnis `dirname` auf der entfernten Maschine.

`lcd dirname` Wechsle ins Verzeichnis `dirname` auf der lokalen Maschine.

`binary` Übertrage die Dateien im Binärmodus. Dies ist bei längeren Dateien nützlich, da es schneller ist. Binärmodus sollte daher so oft wie möglich verwendet werden. Allerdings ist das nicht möglich, wenn mindestens eine der beiden Maschinen keine Unix–Maschine ist.

`ascii` Schaltet den Übertragungsmodus von binär wieder auf den voreingestellten Wert zurück.

`hash` Druckt sog. „hash“–Zeichen (`#`) während des Dateitransfers, um die Geschwindigkeit des Transfers anzuzeigen. Die Menge der übertragenen Information, die einem hash–Zeichen entspricht, hängt vom `ftp`–Server ab.

`get fname` Hole Datei `fname` von der entfernten an die lokale Maschine.

`put fname` Lege die Datei `fname` der lokalen Maschine auf die entfernte Maschine.

`mget fileliste` Hole alle Dateien in der `fileliste` von der entfernten auf die lokale Maschine.

`mput filelist` Lege alle Dateien in der `fileliste` von der lokalen auf die entfernte Maschine.

`prompt` Schalte den interaktiven Modus ein bzw. aus. Dies ist nützlich bei Verwendung von `mget` und `mput`, die sonst nämlich bei jeder einzelnen Datei, die sie übertragen sollen, eine Bestätigung verlangen.

`quit` Schließe die Verbindung und beende die `ftp`–Sitzung.

Binärzahlen und Fließkommazahlen in maschineninterner Darstellung können zwischen Maschinen verschiedener Architektur (z.B. zwischen einer DECStation und einer SUN SPARCstation) nicht übertragen werden.[8]

Manche Computerinstallationen am Netz stellen *anonymen ftp* zur Verfügung. An diesen Orten gibt es Archive, in denen freie Software abgelegt wird. Auf die Dateien solcher Archive kann man durch anonymen ftp zugreifen, ohne eine Benutzerkennung auf dieser Maschine zu haben. Dazu wird als Benutzername `anonymous` oder `ftp` abgegeben, als Paßwort die E-Mail–Adresse (!). NIEMALS hier das eigene Paßwort angeben![9] Informationen über anonyme ftp-Server erhält man meistens aus den NetNews (siehe unten).

Bevor man allerdings Dateien von einem weit entfernten ftp–Server holt, sollte man sich in lokalen Archiven umsehen. Anonymer ftp gehört zu den Dingen, die ohnehin stark ausgelastete Netze noch weiter belasten. Auch das viel erwähnte Programm `archie`, das Informationen über anonyme ftp–Server von `archie`–Servern holt, belastet das Netz. An vielen Orten wurden daher sehr gut ausgestattete, lokale Archive installiert, die Dateien wichtiger Archive jede Nacht transferieren. Auf solche Archive sollte man zuerst zugreifen. Im Zweifelsfall kann die Systemverwalterin welche nennen.

9.4 Elektronische Post

Das Netz ermöglicht das Versenden und Empfangen elektronischer Post (E-Mail). Es gibt eine Reihe von Programmen, die den Umgang mit E-Mail ermöglichen. Einfache Programme sind `mail` oder `mailx`, die mit beinahe jedem Unix–System ausgeliefert werden.

[8] Bitte hier auf den Unterschied zwischen einer maschineninternen Fließkommadarstellung, wie sie z.B. durch den `fwrite`–Befehl in C erzeugt wird, und einer Fließkommazahl, die wie Text (z.B. 3.2) abgespeichert wird, achten. Letztere sind natürlich für die Maschine Text und daher auch zwischen verschiedenen Architekturen problemlos zu übertragen.

[9] Falls das versehentlich doch einmal geschieht, sollten Sie sofort Ihr Paßwort ändern, da die Eingabe von Benutzerkennung und Paßwort an ftp-Servern i.d.R. protokolliert werden.

Im `emacs` steht `rmail` zur E-Mail–Bearbeitung zur Verfügung. Dies wird durch `ESC x rmail` im `emacs` gestartet. `C-h m` gibt eine Übersicht über die Kommandos. Andere komfortable Mail–Programme sind `mh` und `elm`. Welche dieser Programme vorhanden sind, ist von System zu System verschieden. Daher werden wir hier auf die Benutzung der verschiedenen Programme nicht eingehen. Es gibt allerdings einige generelle Dinge, die für die Benutzung von E-Mail nützlich zu wissen sind.

Zunächst einmal braucht man die Adresse der Person, der E-Mail geschickt werden soll. Nehmen wir im Moment an, daß diese Person Zugang zum Internet hat. Die E-Mail–Adresse hat dann folgende Form:

```
name@domainname
```

`name` ist der Name der Mailbox, das ist so etwas wie ein Briefkasten für E-Mail. Auf Unix–Systemen wird dafür gewöhnlich der Loginname der Benutzerin verwendet. Einige Systeme erlauben es, den vollen Namen des Benutzers zu verwenden. Es gibt noch andere Mailboxen, z.B. Namen von Mailing–Listen, bei denen die E-Mail, die in dieser Mailbox ankommt, an eine Liste von anderen Mailboxen weiterverteilt wird.

`domainname` ist der Domainname, über den die Mailbox erreicht werden kann. Das muß nicht unbedingt der Name des Rechners sein, an dem der Empfänger meistens arbeitet. Deswegen muß einem dieser Domainname mitgeteilt werden. Manchmal ist der Domainname auch ein Aliasname, der von dem Ort definiert wird, zu dem Mail geschickt werden soll. Das geschieht häufig, um kürzere Mailadressen zu erhalten, die man sich besser merken kann (siehe das `cc:` Feld im Mailkopf unten als Beispiel).

Ein typischer Mailkopf sieht etwa so aus (abgesehen von Modifikationen, die von Mailprogramm und lokaler Konfiguration abhängen):

```
To: juser@foo.bar.edu
cc: arthur@magrathea.universe
Bcc: mein_loginname
Subject: Demonstration

Hier folgt der eigentliche
Text der Mail.
```

`To:` ist der Adressat, `cc:` sind Adressen, die eine Kopie der E-Mail erhalten sollen. Diese Adressen erscheinen im Kopf jeder E-Mail, die verschickt wird. Das `Bcc:` Feld (Blind carbon copy) enthält Adressen, die eine Kopie erhalten sollen, aber nicht in den Köpfen der verschickten Mails auftauchen. Meist wird das verwendet, um sich selbst eine Kopie der Mail zu schicken. Da im lokalen Netz üblicherweise kein Maschinenname in der Mail–Adresse erscheinen muß, genügt es, hier den eigenen Loginnamen einzusetzen. Im voreingestellten Mail-Kopf, den Ihr Mailprogramm automatisch erzeugt, wird vermutlich nur entweder `cc:` oder `Bcc:` auftauchen. Trotzdem können beide verwendet werden. Alle Adreßfelder können mehrere Adressen enthalten. Die einzelnen Adressen werden durch ein Komma voneinander getrennt.

Ist die E-Mail verschickt, so findet keine Benachrichtigung statt, ob die Mail korrekt angekommen ist. Die Tatsache, daß die eigene Kopie ankommt, bedeutet nicht, daß auch andere Empfänger die Mail erhalten haben. Allerdings wird man üblicherweise informiert, falls die Mail nicht ausgeliefert werden konnte. Erhält man innerhalb einiger Tage keine solche Mißerfolgsmeldung, kann man ziemlich sicher sein, daß die E-Mail beim Empfänger angekommen ist, was aber nicht bedeutet, daß die Nachricht auch gelesen wurde. Mit Hilfe des Kommandos `finger` kann manchmal abgefragt werden, ob jemand seine Mail gelesen hat. Siehe Kapitel A.13.

Falls die Auslieferung fehlschlägt, erhält man die Mail mit einem Kommentar über die Ursache des Problems zurück. Einige häufige Gründe sind:

`unknown user` Der angegebene Benutzername paßte zu keiner Mailbox an dem Ort, wohin die Mail geschickt wurde. Möglicherweise haben Sie sich beim Mailboxnamen verschrieben.

`unknown host` Der Domainname konnte nirgends gefunden werden.

`host has been down for ...` Die Maschine, die die Mail für den angegebenen Domainnamen verwaltet, wurde zwar gefunden, reagiert aber derzeit nicht auf Mailtransfers (z.B. weil sie defekt ist).

Soll einer Person E-Mail geschickt werden, die keinen Zugang zum Internet hat, wohl aber zum BITNET, so verwende man einfach die BITNET-Adresse und hänge ein `.bitnet` hinten dran.

Einige Mail–Adressen enthalten `%` Zeichen, z.B.

```
Name%host1@host2.domain
```

Das bedeutet lediglich, daß die Mail durch ein Gateway (hier `host2`) laufen muß und ist für den Sender der Mail unwichtig. So eine Adresse wird wie jede andere Internet–Adresse behandelt.

Mailadressen der folgenden Formen können hingegen sehr wohl Probleme bereiten:

- `hostn! ... host2!host1!name`

  ist eine UUCP–Adresse. Die einfachste Umsetzung in eine Internet–Adresse ist

  ```
  name@host1.uucp
  ```

  Falls das nicht funktioniert (ziemlich wahrscheinlich), muß mindestens ein Gateway in der Mailadresse angegeben werden. Versuchen Sie etwas in der Art

  ```
  name%host1%host2 ...hostn-1@hostn.uucp
  ```

  oder

  ```
  hostn-1!hostn-2!....host2!host1!name@hostn.uucp
  ```

  oder fragen Sie Ihren lokalen Mail–Guru.

- `host::name`

  ist eine DECnet–Adresse. Diese Adressen sind schwer in Internet–Adressen umzusetzen, falls es überhaupt möglich ist. Am besten läßt man sich eine andere Adresse von dieser Person geben!

9.5 NetNews

Es gibt ein weltweites Informationssystem, das `NetNews` oder ganz einfach `News` genannt wird. Dieses System beinhaltet weit mehr als tausend thematisch geordnete Newsgruppen, deren Themen von rein technischen Computerangelegenheiten über politische Information bis hin zu Freizeittips und Witzen reichen. Wie viele Newsgruppen tatsächlich zur Verfügung stehen, hängt von der lokalen Installation ab.

Um News zu lesen, braucht man ein spezielles Programm, einen Newsreader. Es gibt davon ziemlich viele und jeder funktioniert selbstverständlich anders als alle anderen. Man informiere sich, welche Newsreader an der lokalen Installation vorhanden sind und wie sie benutzt werden.

Doch Vorsicht! Liest man zum ersten Mal News, kann es durchaus passieren, daß alle Newsgruppen als abonniert erklärt werden und die freundliche Meldung auftaucht „there are 120.000 unread articles“. Natürlich kann man Newsgruppen abbestellen.

9.6 Das lokale Netz

Das Netz ist nicht nur für Login an entfernte Maschinen oder E-Mail wichtig. Auch an Orten, die gar nicht an einem externen Netz angeschlossen sind, gibt es oft ein lokales Netz zwischen den eigenen Maschinen, um die Arbeit damit angenehmer für die Benutzer zu machen. Das wird vor allem über folgende Mechanismen erreicht:

- Netzwerk–Informations–System (NIS), früher YP (Yellow Pages).[10] NIS hält Informationen über die Benutzer, wie sie etwa im Paßwort–File vorhanden sind, an einem (oder auch mehreren) zentralen NIS–Servern im Netz. Die Dateien sind allen anderen Maschinen zugänglich. Auf diese Weise sind Paßwort und Login–Shell auf allen Maschinen identisch, die in der gleichen NIS–Domain sind und daher ihre Informationen vom gleichen NIS–Server beziehen. Ändert man Paßwort oder Login–Shell auf irgendeiner dieser Maschinen, so wird diese Änderung auf dem NIS–Server gespeichert und ist damit allen Maschinen in dieser NIS–Domain bekannt.

- Das Netzwerk–File–System (NFS) erlaubt es, Festplattenplatz über das Netz auf verschiedenen Maschinen verfügbar zu machen. Das ist besonders wichtig für das Homedirectory. Mit NFS ist es möglich, daß Ihr Homedirectory auf allen Maschinen am lokalen Netz dasselbe ist. Physikalisch liegt das Homedirectory auf einer Festplatte, die an einer Maschine, einem sog. Fileserver, hängt. Andere Maschinen können diese Platte mounten (vom entsprechenden Befehl `mount`), so daß sie an diesen Maschinen wie eine lokale Platte verfügbar ist. Dieser Mechanismus wird auch für Anwendungsprogramme verwendet, da es wesentlich einfacher ist, ein einzelnes Programm auf einem Fileserver auf dem aktuellen Stand zu halten als viele lokale Kopien. Oft ist nur ganz wenig Software tatsächlich lokal an den einzelnen Maschinen vorhanden, während der Rest auf Platten an einem Fileserver gespeichert und über NFS zugänglich gemacht wird.

[10] NIS/YP und NFS wurden von SUN Microsystems entwickelt und von vielen anderen Herstellern lizensiert. Beide sind heutzutage Standard.

- Drucken kann ein netzwerkweiter Service sein. Dies gestattet, jeden Drucker im lokalen Netz anzusprechen, egal an welcher Maschine er tatsächlich angeschlossen ist.

Ein lokales Netz muß nicht so aussehen wie gerade beschrieben. Allerdings werden Sie es ausgesprochen angenehm finden, wenn es das tut.

10 Sicherheit

10.1 Warum Sicherheit?

Von vielen Benutzern wird Sicherheit als völlig unwichtig und lästig abgetan. Eindringlinge werden als Problem militärischer Installationen betrachtet. Und schließlich: wer könnte denn schon etwas mit meiner Benutzerkennung anfangen?

So einfach ist die Sache leider nicht. Gibt es schon im lokalen Netz Sicherheitsregeln zu beachten, so gilt das in ungleich höherem Ausmaß für Installationen, die an große Netze angeschlossen sind. Doch da die Risiken oft nicht wahrgenommen werden wollen, werden wir hier ein paar der Probleme aufzeigen, vor denen man sich schützen sollte.

- Ein Arbeitskollege überschreibt versehentlich eine Ihrer Dateien, deren Zugriffsberechtigung zu großzügig gesetzt war.

- Die Putzkolonne schaltet versehentlich eine Maschine aus. Beim Reboot zeigt sich, daß die Daten auf der Festplatte nicht mehr gerettet werden können. Die Sicherheitskopie dieser Datei kann nicht gelesen werden, weil die Backup-Routine fehlerhaft ist und nie jemand die Korrektheit dieser Routinen überprüft hat.

- In einer kleinen Firma ist das Systemverwalter-Paßwort Allgemeingut. Eine neue Kollegin startet ein `rm`-Kommando versehentlich als Systemverwalterin in einem Systemverzeichnis. Wenn Sie besonderes Pech haben, muß das Betriebssystem neu installiert werden, doch niemand von Ihnen hat das Wissen, um so weitgehende Systemverwaltungsaufgaben korrekt zu erledigen.

- Ein Kollege ist wütend auf Sie und nützt Ihr unbeaufsichtigtes Terminal, um einer anderen Kollegin beleidigende E-Mail mit Ihrer Benutzerkennung zu schreiben. Anschließend löscht er Ihren neuesten Report.

Diese Probleme können sehr wohl in einer Installation auftreten, die von den großen Netzen weit entfernt ist. In dem Moment, wo es jemanden gibt, der bewußt Schaden anrichten will oder dies aus Unkenntnis tut, müssen Sie Sicherheitsvorkehrungen getroffen haben. Sonst können Sie empfindliche Verluste erleiden.

Wesentlich gravierender wird das Problem, wenn Sie an einem großen Netz angeschlossen sind. Dann nämlich sind Sie offen für weltweite Attacken. Und denken Sie nicht, daß Ihre unbedeutende Computerinstallation für einen Cracker[11] uninter-

[11] Insider verwahren sich sehr gegen den von den Medien bevorzugten Terminus „Hacker" im Zusammenhang mit illegalen Eindringlingen. Diese werden „Cracker" genannt. Als „Hacker" werden all jene bezeichnet, die Spaß daran haben, unter die Oberfläche eines Computersystems zu schauen. Und das ist meistens ganz und gar nicht mit illegalen Tätigkeiten verbunden.

essant ist! Ein schlecht gewartetes System, in das leicht eingedrungen werden kann, ist nämlich ein idealer Stützpunkt für Leute, die ihre Spur verwischen wollen. Und dann gibt es noch eine Menge Cracker, die einfach ihre „Macht“ beweisen wollen, indem sie die Arbeit anderer Menschen zerstören. Sie haben also allen Grund, sich zu schützen.

10.2 Sicherheitsregeln für Benutzer

Der Schutz des Systems umfaßt mehrere Ebenen:

- Schutz der Hardware. Das ist Sache der Systemverwalter und umfaßt Dinge wie die Installation unterbrechungsfreier Stromversorgungen, feuersichere Deponierung von Backup-Bändern u.ä.
- Schutz des Systems auf Superuser-Ebene. Das ist natürlich auch Sache der Systemverwaltung.
- Schutz des Systems auf Benutzerebene. Das geht Sie an.

Als Benutzer können Sie in erster Linie Ihre Benutzerkennung schützen und darauf achten, ob Ihnen Dinge auffallen, die auf mögliche Eindringlinge hinweisen. Die meisten dieser Maßnahmen bedeuten für Sie wenig zusätzlichen Aufwand und gehören bald zur täglichen Routine. Auf einige dieser Maßnahmen wurde in dieser Einführung auch schon hingewiesen.

- Ihre Kennung wird zuallererst durch Ihr Paßwort geschützt. Deswegen sollten Sie unbedingt ein gutes Paßwort wählen (s.a. Kapitel 2.1). Bei vielen Installationen an großen Netzen ist es Crackern möglich, das verschlüsselte Paßwort zu erlangen und dann in aller Ruhe mit einem Paßwort-Suchprogramm zu bearbeiten. Ändern Sie Ihr Paßwort von Zeit zu Zeit. Ändern Sie es auf jeden Fall, wenn Sie glauben, jemand anders kennt Ihr Paßwort! Halten Sie Ihr Paßwort geheim und wählen Sie auf verschiedenen Rechner verschiedene Paßworte.
- Ein weiterer einfacher Schutz sind die Zugriffsberechtigungen (s. Kapitel 5.5). Wenn Dateien nicht von anderen gelesen werden müssen, dann machen Sie sie auch nicht lesbar oder gar schreibbar für andere! Hat jemand in einem Ihrer Verzeichnisse Schreibzugriff, so ist es für diese Person unter bestimmten Voraussetzungen möglich, Programme zu installieren, die unbeschränkten Zugriff auf all Ihre Dateien verschaffen können.
- Verwenden Sie keine `.rhosts`-Dateien.
- Lassen Sie Ihren Bildschirm nicht einfach in öffentlich zugänglichen Räumen stehen, wenn Sie noch eingeloggt sind und kurz weggehen. Es gibt Programme (z.B. `xlock`), die ein Bild auf den Schirm werfen oder ihr Terminal sperren und Eingaben erst wieder zulassen, nachdem Sie Ihr Paßwort eingetippt haben.
- Achten Sie beim Einloggen darauf, ob die Ihnen angezeigte letzte Login-Zeit stimmt. Wenn nicht, könnte Ihre Kennung von jemand anderem benutzt worden sein.

- Achten Sie auf Ihre Dateien. Findet sich darunter etwas, was Sie sicher nicht dorthin getan haben, so wenden Sie sich an Ihre Systemverwalterin. Meistens werden Dateien, die von Crackern hinterlassen werden, als „unsichtbare“ Dateien angelegt, die mit einem Punkt beginnen.

- Legen Sie eigene Backups an und überprüfen Sie von Zeit zu Zeit, ob Sie diese auch wieder lesen können.

Niemand sollte hysterisch werden, wenn es um Computersicherheit geht. Seien Sie sich darüber im klaren, daß der einzig wirklich sichere Computer der ist, der ausgeschaltet in einem Safe steht. Sie können Ihren Computer nicht völlig sicher machen, aber Sie können eine ganze Menge tun, um die Sicherheit zu erhöhen. Es gibt viele Mechanismen, die weit über die hier gegebenen Tips hinausgehen. Wenn Sie sich näher über das Thema Sicherheit informieren wollen, lesen Sie eines der Bücher, auf die im Anhang E hingewiesen wird.

11 Das X–Window–System

Das X–Window–System ist ein frei erhältliches Window–System, das am MIT entwickelt wurde. Es läuft auf vielen verschiedenen Plattformen und gestattet es, sich Fenster über das Netz von einer anderen Maschine öffnen zu lassen. Die Portabilität und die Netzwerkfähigkeit von X machen es anderen Window–Systemen (wie etwa Sunview) überlegen. Es gibt einige graphische Oberflächen (z.B. Openwindows und Motif), die auf X aufbauen und daher ähnlich zu benutzen sind. Die derzeit aktuelle Version von X ist X11, momentan Release 5 (oft als X11R5 bezeichnet). Die folgenden Ausführungen beziehen sich auf X11. In graphischen Oberflächen, die auf X11 aufbauen, haben die Kommandos oft andere Namen und Optionen. Die prinzipielle Funktionsweise ist aber dieselbe.

Loggt man sich an einer X–fähigen Maschine ein, so erfolgt oft schon der Login unter X und nach dem Einloggen erscheinen dann meistens mehrere Fenster. An manchen Systemen muß X erst gestartet werden. Das dafür notwendige Kommando ist oft so etwas wie `xinit` oder `xstart`, eine feste Regel gibt es jedoch nicht.
Es gibt viele Anwendungen unter X:

- der `xterm` Terminal–Emulator (das sind die netten Fenster, in denen man Kommandos absetzen kann); ein ganz einfaches `xterm` wird mit dem Kommando `xterm &` gestartet.

- diverse Window–Manager; diese kümmern sich um die (Un)Ordnung der Fenster sowie deren funktionelle und dekorative Verzierungen.

- einige hübsche kleine Programme, z.B. die Uhr `xclock` oder `xbiff`, ein Programm, das visuell und akustisch über neue E-Mail informiert.

- eine Vielzahl von Programmen für Desktop–Publishing (z.B. FrameMaker oder Publisher), Tabellenkalkulation (z.B. Wingz) etc.

- Graphik–Programme; davon gibt es sehr viele. Einige gebräuchliche, frei erhältliche Programme sind das Zeichenprogramm `xfig`, das Daten–Darstellungsprogramm `xvgr` und das ganz hervorragende Programm `khoros`, das Bildverarbeitung, Datenanalyse und graphisches Programmieren miteinander verbindet. Ebenso ist eine Vielzahl kommerzieller Graphik–Pakete erhältlich.
- mit X kann man sich selbst die graphischen Oberflächen von Programmen zusammenstellen

Viele Dinge um X herum, z.B. welcher Window–Manager verwendet wird oder welche Graphikprogramme zur Verfügung stehen, sind völlig von der lokalen Installation abhängig und werden daher hier nicht diskutiert. Auf den folgenden Seiten werden einige grundlegende Dinge zu X erklärt, so daß es Ihnen möglich sein sollte, sich durch Ihr lokales System durchzukämpfen – auch wenn es sich dabei um einen wahren X-Dschungel handeln sollte!

11.1 Die Rolle der Maus

Sitzt man vor einem X–Window–Bildschirm mit einigen offenen Fenstern, so sieht man, daß das Bewegen der Maus den sog. Mauscursor über den Bildschirm bewegt. Wenn er sich innerhalb eines Fensters befindet, so kann dieses als aktiv, d.h. zur Eingabe bereit, markiert werden. So eine Markierung erfolgt etwa über einen Farbwechsel der Titelleiste. Manche Window–Manager verlangen hingegen, daß ein Fenster durch eine zusätzliche Aktion wie das Drücken eines Mausknopfes aktiviert wird. Wenn kein Fenster aktiv ist und man dennoch eine Eingabe versucht, so geschieht überhaupt nichts.

Der reine Hintergrund, auf dem sich all die Fenster und sonstigen Anwendungen befinden, heißt `Root Window`.

Die Wirkung, die durch das Drücken einer der drei Maustasten erzielt wird, hängt vom Kontext ab, in dem dieser Knopf gedrückt wird. Der Kontext besteht aus

- dem Hintergrund, vor dem sich der Mauscursor bei Drücken des Knopfes befindet: war es vor dem Root Window, vor einem Fensterkörper oder vor einer Titelleiste?
- den Modifikations–Tasten (`Shift`, `Ctrl`, `Meta`), die gleichzeitig mit der Maustaste gedrückt werden.

Hier ist eine Liste von gebräuchlichen Kontexts und ihren Wirkungen. Die resultierenden Aktionen, die in dieser Liste aufgeführt werden, können nur Beispiele sein, da sie in den meisten Fällen vom Window–Manager und lokalen Konfigurationen abhängen.

Root Window Menüs, z.B. für Logins auf anderen Maschinen oder Fenster–Operationen; dies ist völlig von der lokalen Konfiguration abhängig.

Fensterkörper + Meta– oder Alt–Taste kann z.B. für Operationen mit sich überlappenden Fenstern verwendet werden: um etwa ein Fenster aus dem „Fenster–Stapel“ nach oben oder nach unten zu bringen. Dies ist vom Window–Manager abhängig.

Fensterkörper + Ctrl– (oder Strg–) Taste Menüs, mit denen für die einzelnen Fenster Terminaleigenschaften geändert, Schiebebalken (Scrollbars) eingeschaltet oder die Schriftarten verändert werden können.

Titelleiste eines Fensters Bewegen eines Fensters, sowie Ändern der Position des Fensters im Fenster–Stapel; abhängig vom Windowmanager.

Größenversteller in der Titelleiste dient zum Ändern der Fenster–Größe; hängt vom Windowmanager ab, wird aber sehr häufig verwendet (oft ein kleines Kästchen am rechten Rand der Titelleiste). Der genaue Mechanismus ist aber für verschiedene Windowmanager unterschiedlich.

Icon-Marke in der Titelleiste verwandelt das Fenster in ein kleines Kästchen (Icon), das am Rand des Bildschirms erscheint. Hängt vom Windowmanager ab, wird aber sehr häufig verwendet.

Icon der Mausdruck auf einem Icon restauriert das Fenster wieder, von dem der Icon erzeugt wurde.

Am besten probieren Sie gleich einmal aus, welchen Effekt die verschiedenen Maustasten auf den diversen Kontexten auf Ihrem System haben!

11.2 Startup–Dateien in X

Ähnlich wie die Shell hat auch das X–Window–System Startup–Dateien. Diese kontrollieren z.B., welche Farbe und Schriftart ein `xterm` als Voreinstellung hat, welche Anwendungen automatisch nach dem Login gestartet werden, wo System–Meldungen erscheinen sollen usw. Leider gibt es (abgesehen von der Datei `.Xdefaults`) keine Konventionen, wie diese Startup–Dateien heißen. Die Namen hängen daher von der lokalen Installation ab. Die Syntax von `.Xdefaults` ist dem Manual zu `X` zu entnehmen.

11.3 Der Windowmanager

Dieses Ding ist —neben Ihnen— verantwortlich für das Chaos auf Ihrem Bildschirm. Meistens führt der Windowmanager an der Seite des Bildschirms eine Liste aller derzeit laufenden X–Anwendungen. Er stattet die `xterms` mit Titelleiste, Größenversteller und Icon–Marke aus. Oft werden auch mehrere Pull–Down–Menüs zur Verfügung gestellt, die durch Drücken der Mausknöpfe vor dem Root–Window zugänglich werden.

Eine der Aufgaben des Windowmanagers ist es, Fenster zu bewegen und an die richtige Stelle zu setzen. Erscheint ein neues Fenster, so zeigt der Windowmanager meist ein Gitter, das am Maus–Cursor hängt. Mit Hilfe des Mauscursors kann nun dieses Gitter an die gewünschte Stelle des Bildschirmes gebracht werden. Drücken einer Maustaste positioniert das Fenster.

Auch Windowmanager haben Startup–Dateien. Diese Dateien enthalten u.a. die Beschreibung der Menüs, die unter verschiedenen Kontexten verfügbar sind.

11.4 Die Terminalemulation xterm

Der einfachste Weg, ein `xterm` zu starten, ist durch Eingabe des Kommandos

```
xterm &
```

Dabei nicht das `&`-Zeichen für den Start im Hintergrund vergessen, da sonst die Shell, von der aus das Kommando abgesetzt wurde, darauf wartet, daß das neue `xterm` beendet ist, bevor der Prompt wieder erscheint.

Es gibt einige Optionen, mit denen man das Erscheinungsbild des `xterm` steuern kann. Hier eine kleine Liste:

- `-fn Schriftart` das Fenster erscheint mit der angegebenen Schriftart (Font). Leider gibt es keine einfache und bequeme Möglichkeit, sich alle verfügbaren Schriftarten anzusehen. Die Kommandos `xlsfonts`, `xfd fontname` und besonders `xfontsel` können dabei helfen, sofern sie auf Ihrem System vorhanden sind (siehe auch die entsprechenden Manuale). Einige gebräuchliche Schriftarten sind 6x10 (sehr klein), 6x13, 7x14, 10x20 (groß). Diese Fontnamen sind übrigens Abkürzungen; Fontnamen unter X sind nämlich sehr lang. Siehe dazu auch die Manuale zu `X` und `xterm`.
- `-geometry Geometrie` dient zur Angabe von Größe und Plazierung des Fensters. `Geometrie` ist ein Ausdruck der Form „Spalten x Reihen +xoffset +yoffset". Das bedeutet, daß ein `xterm` so viele Zeichen pro Zeile haben soll, wie `Spalten`, und so viele Zeilen, wie `Reihen` angegeben sind. `xoffset` und `yoffset` bestimmen den Ort des Fensters. Sie werden in Pixeln vom linken Bildrand bzw. vom oberen Bildrand aus gemessen. Ersetzt man die `+` Zeichen durch `-`, so wird von rechts bzw. von unten aus gemessen. Es können auch nur die Reihen und Spalten oder nur die Plazierung angegeben werden. Innerhalb der Option dürfen keine Leerzeichen auftauchen.
- `-bg Farbe` setzt die Hintergrund–Farbe. Farben können mit besonderen Farbnamen angegeben werden. Die Farbnamen können durch `showrgb` aufgerufen werden.
- `-fg Farbe` setzt die Farbe des Fenstervordergrundes.
- `-n Name` Der Name des Fenster im iconifizierten Zustand.
- `-T Titel` Der Titel des Fensters, der in der Titelleiste erscheint.
- `-e Kommando` `Kommando` wird ausgeführt, wenn das Fenster aktiviert wurde. Diese Option muß als letzte angegeben werden.

```
xterm -fn 7x14 -g 80x60 -n Manual -T Manual -e man xterm &
```

startet ein langes Fenster mit einem etwas größeren Font, das als Icon und in der Titelleiste „Manual" genannt wird. In diesem Fenster wird die Manualseite für `xterm` aufgerufen. Einige Windowmanager können einen Teil dieser Optionen ignorieren. Die Optionen `fn`, `geometry`, `bg` und `fg` werden von fast allen X–Anwendungen erkannt.

11.5 Einige andere X–Anwendungen

Die folgenden Anwendungen haben einige gemeinsame Optionen. Mit `-bg` und `-fg` können Hintergrund– und Vordergrund–Farbe gewählt werden. Die `-geometry`–Option legt die Geometrie der Anwendung fest. Im Gegensatz zu `xterm` wird die Größe in der Regel in Pixeln angegeben.

xbiff kleiner Briefkasten, der die Farbe ändert und piepst, wenn E-Mail ankommt.

xload graphische Anzeige der Maschinenauslastung.

xclock eine „analoge" Uhr. Mit der Option `-d` wird eine digitale Anzeige daraus, wobei zusätzlich das Datum angezeigt wird.

oclock "analoge", runde Uhr.

xcalc wissenschaftlicher Desktop–Rechner.

xman eine angenehm zu benutzende X–Variante des `man`–Kommandos.

11.6 Benutzung von X im Netz

Stellen Sie sich ein `xterm` vor, in dem Sie einen `rlogin` an eine andere Maschine laufen haben. Sie wollen nun auf dieser anderen Maschine eine X–Anwendung starten, aber diese natürlich auf dem Bildschirm, vor dem Sie sitzen, dargestellt haben. Diese Situation ist ziemlich häufig gegeben, wenn spezielle Programme nur an einigen Maschinen verfügbar sind. Damit das funktioniert, müssen Sie folgendes tun:

- Teilen Sie der entfernten Maschine mit, wo die X–Anwendung dargestellt werden soll. Dazu setzen Sie die Variable `DISPLAY` auf der entfernten Maschine auf den Display–Namen Ihrer lokalen Maschine. Der Display–Name ist

  ```
  Maschinenname:0
  ```

  falls an der Maschine nur ein Bildschirm hängt. Bei mehreren Bildschirmen ändert sich lediglich die Ziffer am Ende des Display–Namens. Das Setzen der `DISPLAY` Variable würde in ksh und bash durch folgendes Kommando auf der entfernten Maschine geschehen:

  ```
  export DISPLAY=lokaler_Maschinenname:0
  ```

- Falls die Angabe des korrekten Displays noch nicht genügt, müssen Sie der entfernten Maschine explizit erlauben, Fenster auf Ihrem Bildschirm zu öffnen. Dies geschieht durch Absetzen des Kommandos

  ```
  xhost Name_der_entfernten_Maschine
  ```

  auf der lokalen Maschine.

Sind diese beiden Schritte durchlaufen, so genügt es, das Kommando für die X–Anwendung auf der entfernten Maschine zu geben. Oft genügt auch nur der erste Schritt.

Es gibt einige besonders dumme Programme, die sich unter X beschweren, wenn man sie auf der eigenen, lokalen Maschine ausführen will. Zunächst sollte man nachsehen, ob die Variable `DISPLAY` richtig gesetzt ist (das gilt auch für weniger dumme Programme) und sie eventuell richtig setzen. Hilft das immer noch nichts, muß man das `xhost` Kommando auf der lokalen Maschine mit dem lokalen Maschinennamen angeben. Es ist lächerlich, aber Computer sind halt nun einmal dumm und manche Programme eben auch.

A Wichtige Unix–Kommandos

Wir listen hier in alphabetischer Reihenfolge einige wichtige Unix–Kommandos zusammen mit den gebräuchlichsten Optionen auf. Natürlich kann solch eine Aufzählung in keiner Weise die Manualseiten ersetzen und erhebt auf keinen Fall Anspruch auf Vollständigkeit. Sie soll Sie lediglich mit wichtigen Kommandos bekanntmachen und andere erwähnen, ohne deren Benutzung genau zu erklären. Die Beschreibungen gelten sowohl für System-V–Derivate als auch für BSD–Systeme, sofern nichts anderes angegeben ist.

A.1 awk

`awk` ist ein Programm, um Muster zu erkennen und zu verarbeiten. Es ist mächtiger als `sed`, aber nicht so vielseitig wie `perl`. `awk` gehört zur Standardausrüstung eines Unix.

A.2 cat

```
cat Filename
```

gibt den Inhalt der Datei `Filename` am Bildschirm aus. Die Ausgabe kann auch in eine andere Datei umgeleitet werden:

```
cat file1 > file2
```

schreibt den Inhalt der Datei `file1` in die Datei `file2`.

```
cat file1 file2 file3 > file4
```

hängt die ersten drei Dateien aneinander an und leitet die resultierende Ausgabe in die Datei `file4` um. Aber Vorsicht! `cat file1 file2 > file1` zerstört die Datei `file1`, da sie zuerst für den Schreibvorgang geöffnet und damit auf Länge Null reduziert wird, bevor das `cat`–Kommando ausgeführt wird.

A.3 cc

ist der C–Compiler Ihres Systems. Die einfachste Art der Benutzung ist:

```
cc file.c
```

Damit wird die Datei `file.c` compiliert und eine ausführbare Datei namens `a.out` erzeugt. Genaueres steht in Kapitel 8.

A.4 cd

```
cd dirname
```

ändert das aktuelle Verzeichnis. Es wird dann `dirname`. Siehe auch Kapitel 5.1.

A.5 chmod

ändert die Zugriffsberechtigungen einer Datei oder eines Verzeichnisses. Siehe Kapitel 5.5.

A.6 compress, uncompress und zcat

Diese Kommandos dienen dazu, Dateien zu komprimieren und wieder aus dem komprimierten Zustand zu restaurieren. Sie werden folgendermaßen verwendet:

```
compress filename
uncompress filename.Z
zcat filename.Z
```

`compress` reduziert die Größe der Datei `filename` und speichert das komprimierte Resultat in der Datei `filename.Z` ab, sofern dies möglich ist.

`uncompress` stellt die komprimierte Datei in ihrer ursprünglichen Form wieder her. Die komprimierte Datei wird dabei gelöscht.

`zcat` ist dasselbe wie `uncompress` mit dem Unterschied, daß das komprimierte File erhalten bleibt. Die Ausgabe erfolgt bei `zcat` in stdout.

Um ein komprimiertes Archiv Ihres Unterverzeichnisses `C` anzulegen, können Sie folgende Befehlssequenz benutzen:

```
tar cvf - C | compress > C.tar.Z
```

Dadurch erhalten Sie eine Datei `C.tar.Z`. Das Minuszeichen im `tar`–Kommando veranlaßt `tar` dazu, die Ausgabe nach stdout zu schreiben. Stdout wird dann mit Hilfe der Pipe | als stdin in das Kommando `compress` umgeleitet. Da `compress` die Eingabe aus stdin erhält, kann es keinen Namen für die `.Z`–Datei finden. Daher erfolgt am Ende des Kommandos eine explizite Umleitung in die gewünschte Datei. Um den Inhalt dieses Archives wiederherzustellen, verwendet man die Kommandozeile

```
zcat C.tar.Z | tar xvf -
```

Das Minuszeichen informiert `tar`, daß die Eingabe über stdin erfolgt (über die Pipe).

Diese Programme können durch das deutlich effizientere `freeze` ersetzt werden (siehe Abschnitt A.14).

A.7 cp

```
cp Quelldatei Zieldatei
cp Dateiliste Verzeichnis
```

`cp` kopiert Dateien und Verzeichnisse. Siehe Kapitel 5.4.

A.8 date

zeigt Datum und Uhrzeit an.

A.9 diff

```
diff file1 file2
```

zeigt die Differenzen zwischen den Dateien `file1` und `file2` an.

A.10 echo

kopiert seine Argumente nach stdout. `echo` wird oft verwendet, um den Wert von Shell–Variablen auszugeben (zur Erinnerung: Shell–Variablen werden mit `$` referenziert).

```
$ echo a
a
$ a=b
$ echo $a
b
$
```

Das Kommando `a=b` ist in csh ungültig und muß durch `set a=b` ersetzt werden.

A.11 file

```
file Dateiliste
```

klassifiziert die Dateien in der `Dateiliste` nach ihrem Inhalt. Mögliche Attribute sind z.B. ascii text, executable, C source code u.ä.

A.12 find

```
find Verzeichnis-Liste Ausdruck
```

sucht Dateien, die sich in einem Verzeichnis der `Verzeichnis-Liste` befinden und die Anforderungen von `Ausdruck` erfüllen. `Ausdruck` kann dabei auch Kommandos enthalten, die auf gefundene Dateien angewendet werden sollen. `find` hat eine ziemlich komplexe Syntax, so daß wir hier nicht näher darauf eingehen. Zudem kann es ein sehr zeitaufwendiges Kommando sein. Aus diesem Grund sollte man NIEMALS einen `find` von einem Verzeichnis aus starten, das weit oben in der Verzeichnis–Hierarchie steht wie etwa `/` oder `/home`. In solch einem Fall würde sich `find` durch sämtliche Verzeichnisse unterhalb dieses weit oben angesiedelten Verzeichnisses durcharbeiten, um alle Dateien in allen Unterverzeichnissen auf die Kriterien von `Ausdruck` hin zu überprüfen. Je nach Größe und Organisation der betroffenen Filesysteme kann ein solcher `find` mehrere Stunden dauern und die entsprechenden Maschinen unbenutzbar machen.

A.13 finger

```
finger name@host
```

`finger` liefert Informationen über Benutzer, z.B. ob sie ihre Mail gelesen haben, welche Telefonnummer sie haben etc. Wird nur ein Loginname angegeben, sieht `finger` auf der lokalen Maschine (manchmal auch auf dem lokalen Netz) nach. Wird nur `@host` angegeben, so wird Information über alle Benutzer ausgegeben, die derzeit an `host` eingeloggt sind. `host` kann dabei ein beliebiger Rechner am Internet sein. Mit der Option `-l` erhält man ausführlichere Information. Einige Systeme haben diesen nützlichen Service aus Sicherheitsgründen abgestellt.

A.14 freeze, melt und fcat

```
freeze filename
melt filename.F
fcat filename.F
```

`freeze` ist ein effizienteres Kompressionsprogramm als `compress`. Die Dateien werden bei `freeze` durch komprimierte Dateien, die durch die Endung `.F` gekennzeichnet werden, ersetzt. Wird kein Dateiname angegeben, so wird die Eingabe von stdin gelesen. Restaurieren erfolgt durch das Kommando `unfreeze` oder (gleichwertig) `melt`. Wird die Ausgabe nach stdout gewünscht, so verwendet man `fcat`; dabei bleibt (wie bei `zcat`) die komprimierte Datei erhalten.

A.15 grep

```
grep Muster Dateiliste
```

sucht nach `Muster` in den Dateien der `Dateiliste`, welche auch Verzeichnisse enthalten kann. In letzterem Fall werden sämtliche Dateien der entsprechenden Verzeichnisse durchsucht. Angenommen, Sie haben eine Datei mit Telefonnummern namens `telnum` und möchten die Telefonnummer eines „Peter“ heraussuchen. Sie erhalten Sie mit dem Kommando

```
grep Peter telnum
```

Mit Hilfe der Option `-i` können Sie die Unterscheidung von Groß– und Kleinbuchstaben abschalten.

A.16 head

```
head -n filename
```

gibt die ersten `n` Zeilen der Datei `filename` auf dem Bildschirm aus. Die Voreinstellung für `n` ist 10.

A.17 kill

```
kill PID-Liste
```

beendet die Prozesse, die durch die `PID-Liste` gekennzeichnet sind, wobei `PID` für Prozeß–Identifikationsnummer steht. Die `PID` wird durch Ausführen des `ps` Kommandos erhalten. Sie können nur Ihre eigenen Prozesse killen. Falls das `kill` Kommendo dazu aber nicht genügt, versuchen Sie `kill -9` anstelle von `kill`.

In Shells, die Job–Control unterstützen, können Sie

```
kill %jobnummer
```

für gestoppte Jobs oder Jobs im Hintergrund verwenden. `Jobnummer` ist die Nummer des zu killenden Jobs und wird vom Kommando `jobs` angezeigt.

A.18 ln

```
ln file link
```

erzeugt einen sog. harten Verweis (Link) zu einer existierenden Datei, d.h. diese Datei kann jetzt mit zwei Namen aufgerufen werden: mit dem ursprünglichen Namen und mit dem soeben geschaffenen Verweisnamen. Beide haben denselben Status. Wird einer der beiden Namen gelöscht, so ist die Datei noch immer unter dem anderen Namen zugänglich. Auf neueren Systemen können auch sog. symbolische Verweise mit Hilfe der `-s` Option erzeugt werden. Dieser Verweis hat nicht denselben Status wie der ursprüngliche Dateiname. Der symbolische Verweis ist eine eigene Datei, die den Namen der Datei enthält, auf die der Verweis zeigt. Wird daher das ursprüngliche File gelöscht, so zeigt der symbolische Verweis auf ein nicht mehr existierendes File und auf die Daten kann nicht mehr zugegriffen werden. Im Gegensatz zum harten Verweis können symbolische Verweise jedoch auch auf ein Verzeichnis erfolgen.

A.19 lpr und andere Druckerkommandos

Die Kommandos `lpr`, `lpq` und `lprm` gelten unter BSD, die entsprechenden Kommandos unter System V sind `lp`, `lpstat` und `cancel`.

```
lpr -Ppname file-liste
```

`lpr` schickt die Dateien der `file-liste` auf den Drucker, der `pname` heißt. Informieren Sie sich über die Druckernamen Ihrer Installation. Typische Namen für Drucker sind „lp“ für einen Zeilendrucker und „ps“ für einen PostScript–fähigen Laserdrucker. Wird die Umgebungsvariable `PRINTER` auf den Namen eines Druckers gesetzt (siehe Kapitel 6.4), so wird dieses der voreingestellte Drucker. Verwenden Sie nun `lpr` ohne Angabe eines Druckers, so werden die zu druckenden Dateien automatisch an den Drucker geschickt, der in `PRINTER` gesetzt wurde.

Eine PostScript–Datei darf NIE zu einem nicht PostScript–fähigen Drucker (z.B. zu einem Zeilendrucker) geschickt werden. Das Resultat wäre nämlich der Ausdruck aller PostScript–Kommandos, nicht aber der gewünschte Text oder die erwünschte Graphik. Geschieht dies versehentlich, so sollten Sie unbedingt diesen Druckjob mit Hilfe der Kommandos `lpq` und `lprm` aus der Druckerschlange nehmen (siehe unten), da sonst eine Menge Papier verschwendet wird.

Die Jobs in der Druckerschlange eines bestimmten Druckers können mit

```
lpq -Ppname
```

aufgelistet werden. Dabei wird auch die Jobnummer jedes Druckjobs angezeigt. Wollen Sie nun einen Job aus der Druckerschlange entfernen, so verwenden Sie das Kommando

```
lprm -Ppname job-nummer
```

wobei `job-nummer` die Jobnummer Ihres Druckauftrages laut `lpq` ist. Selbstverständlich können Sie nur Ihre eigenen Druckaufträge löschen.

A.20 ls

```
ls file-liste
```

gibt Information über die Dateien in `file-liste` aus. Dabei darf `file-liste` auch Verzeichnisse enthalten, so daß alle Dateien dieser Verzeichnisse aufgeführt werden. Wird die `file-liste` weggelassen, so erscheint eine Liste der Dateien im aktuellen Verzeichnis. Folgende Optionen für `ls` sind oft nützlich:

`-a` zeige auch Dateien, die mit einem Punkt beginnen

`-l` zeige ausführliche Informationen über die Dateien einschließlich der Zugriffsberechtigungen

`-F` zeige einen `/` nach jedem Verzeichnis, einen `*` nach jeder ausführbaren Datei und (auf Systemen, die das unterstützen) ein `@` nach symbolischen Verweisen.

A.21 mail, Mail und mailx

`mail` ist ein sehr einfaches Programm zum Lesen und Verschicken von E-Mail. Es stellt nur elementarste Möglichkeiten zum Editieren beim Schreiben von Mail zur Verfügung. `Mail` (BSD) und `mailx` (SYSTEM V) sind etwas besser. Wir empfehlen aber, ein komfortableres Mail–System zu verwenden, z.B. `rmail`, `elm` oder `mh`. Der Gebrauch von `rmail` ist in der `Emacs`–Referenzkarte in Anhang B beschrieben.

A.22 make

ist ein nützliches Werkzeug, um den Compilationsprozeß bequemer zu gestalten. Es ermöglicht, daß automatisch nur immer diejenigen Dateien neu compiliert werden, die sich seit der letzten Compilation geändert haben, sowie die Dateien, die von geänderten Dateien abhängen. Das Kommando `make` verarbeitet die Datei `Makefile` oder auch `makefile`, die Anweisungen für die Compilation enthalten, z.B. die Abhängigkeiten von Dateien. Wegen der genauen Syntax und einer ausführlichen Beschreibung wende man sich an die Manuale oder an ein Buch über `make`. Für die GNU–Version von `make` gibt es eine sehr gute Dokumentation (info–Eintrag im `emacs`), die auch als Einführung in andere `make`–Versionen dienen kann.

A.23 man

```
man Kommando
```

gibt die Manualseiten für `Kommando` aus. Die `-k` Option gibt alle Überschriften der Manuale aus, in denen der Suchbegriff `Kommando` gefunden wurde. Nähere Information über Hilfestellungen in Unix ist im Kapitel 4 zu finden.

A.24 mkdir

```
mkdir Verzeichnis-Liste
```

erzeugt die in der Liste angegebenen Verzeichnisse.

A.25 more

```
more Datei-Liste
```

gibt die Dateien in der Liste Seite für Seite aus. Auf manchen alten System-V-Maschinen muß statt dessen `pg` verwendet werden.

A.26 mv

```
mv vorhandene_Datei neue_Datei
```

benennt eine Datei um. Siehe auch Kapitel 5.4.

A.27 nice

erniedrigt die Priorität eines Prozesses, um die Maschine für andere Aufgaben nicht zu sehr zu blockieren. Siehe Kapitel 3.3.

A.28 nohup

```
nohup Kommando
```

stellt sicher, daß der durch `Kommando` gestartete Prozeß nicht beim Logout beendet wird. Wenn keine Ausgabedateien angegeben werden, werden sowohl stderr als auch stdout in die Datei `nohup.out` umgeleitet. `nohup` startet Prozesse automatisch mit etwas erniedrigter Priorität.

Wird csh verwendet, so wird durch das Starten eines Prozesses im Hintergrund bereits sichergestellt, daß er beim Logout nicht beendet wird. Daher ist das `nohup`–Kommando in csh unnötig. Dies gilt aber nicht für tcsh.

A.29 ps

informiert über den Status aktiver Prozesse. Ohne Angabe von Optionen erhält man lediglich Information über die eigenen Prozesse, die vom selben Terminal wie das `ps` Kommando gestartet wurden. In einem Window–System stellt aber jedes Fenster ein eigenes Terminal dar. Um alle eigenen Prozesse zu überblicken, verwendet man daher (in BSD)

```
ps -ux
```

Um Prozesse anderer Benutzer ebenfalls angezeigt zu bekommen, verwendet man (in BSD)

```
ps -aux
```

Beschränkung auf Prozesse, die gerade wirklich am laufen sind, liefert die Option `-r`. Wünscht man, daß alle Zeilen ganz ausgeschrieben werden und nicht nach 80 Zeichen abgebrochen werden, kann dies durch Angabe von `ww` bei den Optionen erreicht werden.

Die Ausgabe von `ps` erfolgt in mehreren Spalten. Einige interessante sind (BSD):

`USER` Besitzer des Prozesses

`PID` Prozeß–Identifikations–Nummer

`%CPU` Prozentsatz der CPU–Zeit, die der Prozeß gerade verbraucht

`%MEM` Prozentsatz des benutzen Hauptspeichers

`SZ` Größe des Prozesses

`RSS` aktuelle Größe des Prozesses im Hauptspeicher

`TIME` bisher verbrauchte CPU–Zeit in Sekunden

`COMMAND` das Kommando, mit dem der Prozeß gestartet wurde

Optionen und Ausgabe von `ps` unterscheiden sich auf System-V-Maschinen deutlich vom hier besprochenen BSD–Fall. Die zu `aux` analogen Optionen sind `ef`. Die Option `-u` gefolgt von einem Benutzernamen informiert über alle Prozesse dieses Benutzers. Für weitere Optionen wende man sich an das Manual.

A.30 rm

```
rm Datei-Liste
```

löscht die Dateien in der Liste. Es gibt keine Möglichkeit, den Effekt dieses Befehls ungeschehen zu machen! Die Option `-i` bewirkt, daß bei jeder Datei noch einmal nachgefragt wird, ob sie wirklich gelöscht werden soll.

A.31 rmdir

```
rmdir Verzeichnis-Liste
```

löscht die leeren Verzeichnisse in der Liste. Verzeichnisse, die noch Dateien enthalten, werden nicht gelöscht.

A.32 ruptime und rup

Diese beiden Kommandos geben den Status der Maschinen am lokalen Netz an.

A.33 rwho und rusers

Diese Kommandos zeigen an, wer am lokalen Netz eingeloggt ist. Das Kommando `rusers` braucht etwas länger. `rwho` ist an manchen Installationen nicht verfügbar, da es zu große Netzlast verursacht. Wegen Optionen siehe die Manualseiten.

A.34 sed

ist ein nicht–interaktiver Editor. Genaue Beschreibung in den Manualseiten.

A.35 sort

sortiert Dateien in ASCII-Sequenz oder anderen Ordnungen, die durch Optionen ausgewählt werden. Lexikalische Ordnung wird z.B. durch die Option `-d` erhalten. Es kann ebenso ausgewählt werden, nach welchem Eintrag in jeder Zeile sortiert werden soll. Siehe die Manuale.

A.36 tail

```
tail Dateiname
```

gibt die letzten Zeilen der Datei `Dateiname` aus.

A.37 talk

```
talk loginname@hostname
```

bittet die Benutzerin `loginname` am Rechner `hostname`, mit Ihnen ein „Gespräch“ über das Netz zu beginnen. Ist die Benutzerin auf derselben Maschine eingeloggt wie Sie, so genügt die Angabe von `loginname` in der Kommandozeile. Ein Gespräch über `talk` wird mit `C-c` beendet. Zwischen verschiedenen Architekturen ist es oft nicht möglich, `talk` zu verwenden. Mit Hilfe des Kommandos `mesg` kann ein Benutzer vermeiden, daß er durch `talk`–Anfragen gestört wird. Der Initiator des `talk` erhält dann die Meldung `Your party is refusing messages`.

A.38 tar

```
tar Key Optionen Datei-Liste
```

erzeugt eine Archiv–Datei oder restauriert Dateien einer Archiv–Datei. Der `Key` entscheidet, ob ein Archiv angelegt oder von einem Archiv gelesen werden soll:

- **c** erzeuge eine Archiv–Datei
- **x** extrahiere Dateien aus einer Archiv–Datei
- **t** liste den Inhalt der Archiv–Datei

Die wichtigsten Optionen sind:

- **v** geschwätzig; listet die bearbeiteten Dateien auf
- **f** gibt an, daß das nächste Argument der Name einer Datei ist, von der gelesen oder auf die geschrieben werden soll (je nachdem , was zum `Key` paßt); wird anstelle eines Dateinamens ein Minuszeichen angegeben, so wird stdin bzw. stdout verwendet.
- **h** folge bei der Erzeugung einer Archivdatei auch symbolischen Verweisen und füge diese Dateien mit in das Archiv ein.

Um ein Archiv Ihres C–Unterverzeichnisses in Ihrem Homedirectory zu erzeugen, würden Sie das folgende Kommando verwenden:

```
tar -cvf C.tar C
```

Dies liefert eine Archivdatei namens `C.tar`. Mit Hilfe des Kommandos

```
tar -xvf C.tar
```

wird der Inhalt des Archives wieder in seiner ursprünglichen Form restauriert.

A.39 tee

```
tee Datei-Liste
```

kopiert stdin nach stdout sowie die angegebenen Dateien. Um an Dateien anzuhängen, verwende man die `-a` Option.

```
a.out | tee outfile
```

gibt die Resultate der ausführbaren Datei `a.out` am Bildschirm aus, schreibt sie aber gleichzeitig in die Datei `outfile`.

A.40 uptime

gibt den Status Ihrer Maschine aus: wie lange sie läuft, wie viele Benutzer eingeloggt sind und wie stark die Maschine ausgelastet ist.

A.41 wc

```
wc Datei-Liste
```

zählt Zeilen, Wörter und Buchstaben der Dateien in der Liste und gibt das Resultat aus. Will man auswählen, was gezählt werden soll, so geschieht dies mit den Optionen `l` (Zeilen), `w` (Wörter) und `c` (Buchstaben). Wird keine Option angegeben, so wird alles gezählt.

A.42 which

```
which Dateiname
```

zeigt den absoluten Pfadnamen der Datei an. `which` sucht aber nur in den Verzeichnissen nach der Datei, die in der Variable `PATH` angegeben sind.

A.43 who

zeigt an, welche Benutzer an Ihrer Maschine eingeloggt sind.

B GNU Emacs Referenz–Karte

In dieser Referenzkarte bedeutet `C-k` „drücke die Control- (Strg-) Taste und die `k`–Taste gleichzeitig“. `M-k` bedeutet „drücke die Meta-Taste und das `k` gleichzeitig“. Die Meta–Taste kann mit `Meta`, mit ◇ oder auch mit `Alt` beschriftet sein. Falls das alles nicht funktioniert, kann die `ESC`–Taste verwendet werden. In diesem Fall drückt man zuerst kurz die `ESC`–Taste und danach die entsprechende Taste.

Diese Referenzkarte ist eine Übersetzung der englischen Referenzkarte der Free Software Foundation, für die folgendes Copyright gilt:

B.1 Aufrufen und Verlassen von Emacs

Um Emacs aufzurufen, tippt man einfach den Namen:	`emacs`
Siehe unten, wie eine Datei zum Editieren eingelesen wird	
stoppe Emacs	`C-z`
verlasse Emacs auf Dauer	`C-x C-c`

B.2 Dateien

Einlesen einer Datei in Emacs	`C-x C-f`
Abspeichern einer Datei auf der Platte	`C-x C-s`
Einfügen einer anderen Datei in diesen Puffer	`C-x i`
ersetze diese Datei mit der, die Sie wirklich wollten	`C-x C-v`
schreibe den Puffer in eine anzugebende Datei	`C-x C-w`
starte Dired, den Verzeichnis–Editor	`C-x d`

B.3 Hilfestellungen

Das Hilfssystem ist einfach. Tippen Sie `C-h` und folgen Sie den Anweisungen. Benutzen Sie Emacs zum ersten Mal, starten Sie mit `C-h t` das **Tutorial**. (Diese Karte setzt das Tutorial voraus).

loswerden des Hilfe-Fensters	`C-x 1`
weiterblättern im Hilfe-Fenster	`ESC C-v`
apropos: zeige Kommandos, die Ausdruck enthalten	`C-h a`
zeige Funktion, mit der eine Taste belegt ist	`C-h c`
beschreibe eine Funktion	`C-h f`
gib Modus-spezifische Information	`C-h m`

B.4 Beheben von Fehlern

Abbrechen eines teilweise getippten oder ausgeführten Kommandos	`C-g`
Restauriere eine durch einen Systemabsturz verlorene Datei	`M-x recover-file`
lösche eine ungewollte Änderung	`C-x u` oder `C-_`
restauriere den ursprünglichen Inhalt eines Puffers	`M-x revert-buffer`
Neuaufbau des Bildschirms	`C-l`

B.5 Inkrementelle Suche

suche vorwärts	`C-s`
suche rückwärts	`C-r`
suche regulären Ausdruck	`C-M-s`

Wiederholte Verwendung von `C-s` oder `C-r` setzt die Suche in der entsprechenden Richtung fort.

beende inkrementelle Suche	`ESC`
lösche Wirkung des letzten Buchstabens	`DEL`
Abbrechen der aktuellen Suche	`C-g`

Sucht Emacs noch, so bricht `C-g` den noch nicht erfolgten Teil der Suche ab, ansonsten den gesamten Suchvorgang.

B.6 Bewegen

Cursor–Bewegung:

Größe der Bewegung	**rückwärts**	**vorwärts**
Buchstabe	`C-b`	`C-f`
Wort	`M-b`	`M-f`
Zeile	`C-p`	`C-n`
gehe zum Zeilenanfang (oder -ende)	`C-a`	`C-e`
Satz	`M-a`	`M-e`
Absatz	`M-[`	`M-]`
Seite	`C-x [`	`C-x ]`
sexp	`C-M-b`	`C-M-f`
Funktion	`C-M-a`	`C-M-e`
gehe zum Pufferanfang (oder -ende)	`M-<`	`M->`

Bildschirm–Bewegung:

blättere zum nächsten Schirm	`C-v`
blättere zum vorhergehenden Schirm	`M-v`
blättere nach links	`C-x <`
blättere nach rechts	`C-x >`

B.7 Killen und Löschen

zu killende Größe	**rückwärts**	**vorwärts**
Buchstabe (löschen, nicht killen)	`DEL`	`C-d`
Wort	`M-DEL`	`M-d`
Zeile (bis zum Ende)	`M-0 C-k`	`C-k`
Satz	`C-x DEL`	`M-k`
sexp	`M-- C-M-k`	`C-M-k`

kill **Bereich** (Region)	`C-w`
kill bis zum nächsten Erscheinen von *Zeichen*	`M-z` *Zeichen*
Wiedergabe (yank) der zuletzt gekillten Größe	`C-y`
ersetze letzte Wiedergabe mit vorausgehendem Kill	`M-y`

B.8 Markieren

setze Marke hier	`C-@` oder `C-SPC`
vertausche Punkt und Marke	`C-x C-x`
setze Marke *arg* **Worte** weiter	`M-@`
markiere **Absatz**	`M-h`
markiere **Seite**	`C-x C-p`
markiere **sexp**	`C-M-@`
markiere **Funktion**	`C-M-h`
markiere ganzen **Puffer**	`C-x h`

B.9 Ersetzen mit Nachfrage

interaktiv einen Textstring ersetzen	`M-%`
unter Verwendung regulärer Ausdrücke	`M-x query-replace-regexp`

Gültige Antworten beim Ersetzen mit Nachfrage sind

ersetze dieses, gehe zum nächsten	`SPC`
ersetze dieses, aber gehe nicht weiter	`,`
springe zum nächsten ohne zu ersetzen	`DEL`
ersetze alle übrigen passenden Ausdrücke	`!`
zurück zum letzten passenden Ausdruck	`^`
verlasse Ersetzen mit Nachfrage	`ESC`
beginne rekursives Editieren (verlassen mit `C-M-c`)	`C-r`

B.10 Mehrere Fenster

lösche alle anderen Fenster	`C-x 1`
lösche dieses Fenster	`C-x 0`
teile Fenster vertikal	`C-x 2`
teile Fenster horizontal	`C-x 5`
blättere anderes Fenster vorwärts	`C-M-v`
springe mit Cursor ins andere Fenster	`C-x o`
vergrößere Fensterlänge	`C-x ^`
verkleinere Fensterbreite	`C-x {`
vergrößere Fensterbreite	`C-x }`
verkleinere Fensterlänge	`M-x shrink-window`
wähle Puffer in anderem Fenster	`C-x 4 b`
Einlesen einer Datei im anderen Fenster	`C-x 4 f`
Zusammenstellung von Mail im anderen Fenster	`C-x 4 m`
starte Dired im anderen Fenster	`C-x 4 d`
finde ein Tag im anderen Fenster	`C-x 4 .`

B.11 Formatieren

Einrücken der aktuellen **Zeile** (Modus–abhängig)	`TAB`
Einrücken des **Bereiches** (Modus–abhängig)	`C-M-\`
Einrücken von **sexp** (Modus–abhängig)	`C-M-q`
rigoroses Einrücken des Bereiches um *arg* Spalten	`C-x TAB`
Einfügen eines Zeilenumbruchs	`C-o`
schiebe Rest der Zeile vertikal nach unten	`C-M-o`
lösche Leerzeilen um aktuelle Position (Point)	`C-x C-o`
lösche sämtlichen Leerraum um aktuelle Position	`M-\`
setze genau ein Leerzeichen an die aktuelle Position	`M-SPC`
fülle **Absatz**	`M-q`
fülle **Bereich**	`M-g`
setze die Füllspalte	`C-x f`
setze das Präfix, mit dem jede Zeile starten soll	`C-x .`

B.12 Änderung von Groß– und Kleinschreibung

schreibe ganzes Wort groß	`M-u`
schreibe ganzes Wort klein	`M-l`
schreibe Wortanfang groß	`M-c`
schreibe ganzen Bereich groß	`C-x C-u`
schreibe ganzen Bereich klein	`C-x C-l`
schreibe Anfangsbuchstaben im Bereich groß	`M-x capitalize-region`

B.13 Der Minipuffer

Folgende Tastenbelegungen sind im Minipuffer definiert:

ergänze soweit wie möglich	`TAB`
ergänze bis zu einem Wort	`SPC`
ergänze und führe aus	`RET`
zeige mögliche Ergänzungen	`?`
Abbruch des Kommandos	`C-g`

Tippen Sie `C-x ESC`, um das letzte Kommando im Minipuffer zu editieren und zu wiederholen. Dann sind die folgenden Tastenbelegungen definiert:

vorhergehendes Minipuffer–Kommando	`M-p`
nächstes Minipuffer–Kommando	`M-n`

B.14 Puffer

wähle anderen Puffer	`C-x b`
Liste alle Puffer	`C-x C-b`
kille einen Puffer	`C-x k`

B.15 Vertauschen

vertausche **Buchstaben**	`C-t`
vertausche **Worte**	`M-t`
vertausche **Zeilen**	`C-x C-t`
vertausche **sexps**	`C-M-t`

B.16 Überprüfung der Rechtschreibung

prüfe aktuelles Wort	`M-$`
prüfe alle Wörter im Bereich	`M-x spell-region`
prüfe gesamten Puffer	`M-x spell-buffer`

(Anm.: hier wird generell ein englisches Wörterbuch verwendet)

B.17 Tags

finde Tag	`M-.`
finde nächstes Vorkommen des Tag	`C-u M-.`
gib neues Tags-File an	`M-x visit-tags-table`
regexp-Suche in allen Dateien der Tags–Tabelle	`M-x tags-search`
Ersetzen mit Nachfrage in allen Dateien	`M-x tags-query-replace`
Fortsetzen der letzen Tags–Suche oder Ersetzen mit Nachfrage	`M-,`

B.18 Shells

Ausführen eines Shell–Kommandos	`M-!`
Anwenden eines Shell–Kommandos auf den Bereich	`M-|`
filtere Bereich durch ein Shell–Kommando	`C-u M-|`
starte eine Shell im Fenster `*shell*`	`M-x shell`

B.19 Rmail

blättere vorwärts	`SPC`
blättere rückwärts	`DEL`
Anfang der Nachricht	`.` (dot)
nächste nicht–gelöschte Nachricht	`n`
vorhergehende nicht–gelöschte Nachricht	`p`
nächste Nachricht	`M-n`
vorhergehende Nachricht	`M-p`
lösche Nachricht	`d`
lösche Nachricht und aktualisiere	`C-d`
mache Löschen der Nachricht ungültig	`u`
antworte auf Nachricht	`r`
schicke Nachricht an jemanden weiter (forward)	`f`
schicke Mail	`m`
hole neu angekommene Mail	`g`
verlasse Rmail	`q`
gebe Nachricht in ein anderes Rmail-File aus	`o`
gebe Nachricht im Unix-Mail-Format aus	`C-o`
zeige Zusammenfassung der Kopfzeilen	`h`

B.20 Reguläre Ausdrücke

Folgendes hat spezielle Bedeutung innerhalb regulärer Ausdrücke:

irgendein Einzelbuchstabe	`.` (dot)
keine oder mehrere Wiederholungen	`*`
eine oder mehrere Wiederholungen	`+`
keine oder eine Wiederholungen	`?`
irgendein Buchstabe im Set	`[` ... `]`
irgendein Buchstabe außerhalb des Sets	`[^` ... `]`
Zeilenanfang	`^`
Zeilenende	`$`
zitiere spezielles Zeichen *c*	`\`*c*
alternatives oder	\—
Gruppieren	\(... \)
*n*te Gruppe	`\`*n*
Pufferanfang	`\‘`
Pufferende	`\’`
Wortabbruch	`\b`
nicht Anfang oder Ende eines Wortes	`\B`
Wortanfang	`\<`
Wortende	`\>`
irgendein Wort–Syntax–Buchstabe	`\w`
irgendein nicht–Wort–Syntax–Buchstabe	`\W`
Buchstabe mit Syntax*c*	`\s`*c*
Buchstabe mit Syntax anders als *c*	`\S`*c*

B.21 Register

kopiere Bereich in Register	`C-x x`
füge Registerinhalt ein	`C-x g`
speichere aktuelle Position im Register	`C-x /`
gehe zur abgespeicherten Position	`C-x j`

B.22 Info

starte das Info–Dokumentationssystem	`C-h i`

Bewegung innerhalb eines Knotens (Node):

blättere vorwärts	`SPC`
blättere rückwärts	`DEL`
Anfang des Knotens	`.` (dot)

Bewegung zwischen Knoten:

nächster Knoten	`n`
vorhergehender Knoten	`p`
gehe nach **oben**	`u`
wähle Menüpunkt beim Namen	`m`
wähle den $n$ten Menüpunkt durch Nummer (1–5)	n
folge dem Querverweis (zurück mit `l`)	`f`
zurück zum letzten gesehenen Knoten	`l`
zurück zum Verzeichnis-Knoten	`d`
gehe zu irgendeinem Knoten mit Namen	`g`

Andere:

starte Info–**Tutorial**	`h`
liste Info–Kommandos	`?`
verlasse Info	`q`
suche in Knoten nach regulärem Ausdruck	`s`

B.23 Tastatur–Makros

beginne Definition eines Tastatur–Makros	`C-x (`
beende Definition des Tastatur–Makros	`C-x )`
Ausführen des zuletzt definierten Tastatur–Makros	`C-x e`
hänge an letzten Tastatur–Makro an	`C-u C-x (`
benenne letzten Tastatur–Makro	`M-x name-last-kbd-macro`
füge Lisp–Definition in Puffer ein	`M-x insert-kbd-macro`

B.24 Kommandos im Umgang mit Emacs Lisp

Auswertung von **sexp** vor aktueller Position	`C-x C-e`
Auswertung des aktuellen **defun**	`C-M-x`
Auswertung des **Bereiches**	`M-x eval-region`
Auswertung des ganzen **Puffers**	`M-x eval-current-buffer`
Lesen und Auswerten des Minipuffers	`M-ESC`
Wiederausführen des letzten Minipuffer–Kommandos	`C-x ESC`
Lesen und Auswerten der Emacs Lisp Datei	`M-x load-file`
Laden vom Standard–System–Verzeichnis	`M-x load-library`

B.25 Einfache Anpassungen

Hier sind einige Beispiele für globale Tastaturbelegungen in Emacs Lisp. Beachten Sie, daß nicht mit `"\M-#` gearbeitet werden kann, sondern `"\e#` verwendet werden muß.

```
(global-set-key "\C-cg" 'goto-line)
(global-set-key "\e\C-r" 'isearch-backward-regexp)
(global-set-key "\e#" 'query-replace-regexp)
```

Beispiel einer Variablenzuweisung in Emacs Lisp:

```
(setq backup-by-copying-when-linked t)
```

B.26 Schreiben von Kommandos

```
(defun <command-name> (<args>)
  "<documentation>"
  (interactive "<template>")
  <body>)
```

Beispiel:

```
(defun this-line-to-top-of-screen (line)
  "Reposition line point is on to the top of
the screen.  With ARG, put point on line ARG.
Negative counts from bottom."
  (interactive "P")
  (recenter (if (null line)
              0
            (prefix-numeric-value line))))
```

Das Argument zu `interactive` ist ein String, der spezifiziert, wie die Argumente bei interaktivem Aufruf der Funktion erhalten werden sollen. Tippen Sie `C-h f interactive` für mehr Information.

C Vi Referenz–Karte

C.1 Maßeinheiten

Kommandos im Kommandomodus sind oft mit Maßeinheiten verbunden, die angeben, in welchem Bereich das Kommando wirksam sein soll. Hier erläutern wir die verschiedenen Begriffe für Maßeiheiten in `vi`.

Buchstabe	jeder Buchstabe einschließlich Leerzeichen, `TAB`, Satzzeichen und Kontrollzeichen
Wort	mehrere Buchstaben, die an beiden Seiten durch ein oder mehrere Satzzeichen, Leerzeichen, `TAB`s, Ziffern oder Zeilenumbrüche begrenzt werden. Eine Gruppe von Satzzeichen ist ein Wort.
leer begrenztes Wort	im Unterschied zum Wort gehören angrenzende Satzzeichen zum leer begrenzten Wort dazu, d.h. es wird durch sogenannten Leerraum (Whitespace), also Leerzeichen, `TAB` oder Zeilenumbrüche, begrenzt.
Zeile	Ansammlung von Zeichen, die durch einen Zeilenumbruch begrenzt werden. Eine logische Zeile ist nicht unbedingt mit einer Zeile am Bildschirm identisch. Wurde am Zeilenende kein `RETURN` eingegeben, so setzt `vi` die logische Zeile fort. Manche Konfigurationen von `vi` trennen logische Zeilen automatisch in Bildschirmzeilen.
Satz	eine Ansammlung von Zeichen, die durch einen Punkt oder ein Ausrufezeichen gefolgt von zwei (!) Leerzeichen oder einem Zeilenumbruch begrenzt wird.
Absatz	Ansammlung von Zeichen, der mindestens eine Leerzeile folgt.

Wird vor einer Maßeinheit eine Zahl angegeben, so wird diese Zahl als Wiederholungsfaktor für die Maßeinheit interpretiert.

C.2 Starten und Verlassen von vi

`vi`	startet `vi` ohne eine zu editierende Datei
`vi filename`	editiere die Datei `filename`
`vi -r filename`	restauriere die Datei `filename` nach einem Systemabsturz
`:wq`	Abspeichern der Änderungen und Verlassen von `vi`
`ZZ`	Abspeichern der Änderungen und Verlassen von `vi`
`:q!`	Wegwerfen der Änderungen und Verlassen von `vi`

C.3 Bewegen

Denken Sie daran, daß Sie sich im Kommandomodus befinden müssen, um den Cursor mit Hilfe der folgenden Kommandos bewegen zu können. Diese Kommandos legen auch Maßeinheiten fest, die zusammen mit anderen Kommandos verwendet werden können. Jedes Kommando kann durch Voranstellen einer Zahl mehrfach ausgeführt werden.

Beachten Sie, daß sich die Tasten `h, j, k, l`, die in `vi` die Bedeutung von Cursortasten haben, direkt unter Ihren Fingern befinden.

`h`	ein Buchstabe nach links
`j`	eine Zeile nach unten
`SPACE, l`	ein Buchstabe nach rechts
`k`	eine Zeile nach oben
`w`	ein Wort nach rechts
`W`	ein leer begrenztes Wort nach rechts
`b`	Wort nach links
`B`	leer begrenztes Wort nach links
`e`	Ende des Wortes rechts
`E`	Ende des leer begrenzten Wortes rechts
`0`	Zeilenanfang (kann nicht mit einem Wiederholungsfaktor versehen werden)
`RETURN`	Anfang der nächsten Zeile
`-`	Anfang der vorhergehenden Zeile
`$`	Zeilenende
`(`	Satzanfang
`)`	Satzende
`{`	Absatzanfang
`}`	Absatzende
n`G`	zu Zeile *n* (ohne *n* zur letzten Zeile)
`H`	zum Bildschirm–Beginn
`M`	zur Bildschirm–Mitte
`L`	zum Bildschirm–Ende

Die nächsten Kommandos bewegen den Cursor um größere Strecken. Diese Kommandos können nicht als Maßeinheiten verwendet werden. Zur Erinnerung: `C-x` bedeutet, daß die `Ctrl`- bzw. `Strg`-Taste gleichzeitig mit der Taste `x` gedrückt werden muß.

`C-d`	eine halbe Bildschirmseite nach unten
`C-u`	eine halbe Bildschirmseite nach oben
`C-f`	eine Bildschirmseite vorblättern
`C-b`	eine Bildschirmseite zurückblättern

C.4 Einfügen von Text

Mit den folgenden Kommandos bleibt man solange im Eingabemodus, bis durch Tippen von `ESC` in den Kommandomodus gewechselt wird.

Folgende Kommandos fügen Text ein:

`i`	vor dem Cursor
`I`	vor dem ersten nichtleeren Zeichen der Zeile
`a`	an der aktuellen Cursorposition
`A`	am Ende der Zeile
`o`	öffne Zeile unterhalb der aktuellen Zeile
`O`	öffne Zeile oberhalb der aktuellen Zeile

C.5 Löschen

In diesem und den folgenden Abschnitten steht M für eine Maßeinheit (s. Kapitel C.3), der ein Wiederholungsfaktor n vorausgehen kann. Beachten Sie, daß `d` gefolgt von `RETURN` zwei Zeilen löscht, nämlich die aktuelle Zeile sowie die folgende. Um nur die aktuelle Zeile zu löschen, verwende man `dd`.

n`x`	lösche *n* Buchstaben beginnend beim aktuellen
n`X`	lösche *n* Buchstaben vor dem aktuellen
`d`*M*	lösche den durch *M* bezeichneten Text
n`dd`	lösche *n* Zeilen
`D`	lösche bis zum Zeilenende

Beispiele:

`d0`	lösche bis zum Zeilenanfang
`dW`	lösche bis zum Ende des leer begrenzten Wortes
`5dd`	lösche 5 Zeilen beginnend mit der aktuellen
`dG`	lösche bis zum Ende der Datei
`d1G`	lösche bis zum Anfang der Datei

C.6 Ändern

Bei Eingabe der ersten vier Kommandos der folgenden Liste erscheint am Ende der angegebenen Maßeinheit ein `$`-Zeichen. Alles vom Cursor bis zu diesem `$` wird durch Ihre Eingabe ersetzt, bis Sie `ESC` tippen, um in den Kommandomodus zurückzukehren. Falls Sie bei der Eingabe über das `$`-Zeichen hinaus tippen, so wird `vi` diese Zeichen einfügen und somit den Text nach dem `$`-Zeichen nicht mehr überschreiben.

n`s`	ersetze *n* Buchstaben
`c`*M*	ändere durch *M* spezifizierten Text
n`cc`	ändere *n* Zeilen
`C`	ändere bis zum Ende der Zeile
n`r`*c*	ersetze *n* Buchstaben durch den einzelnen Buchstaben *c* (kehrt in Kommandomodus zurück)
`R`	ersetze (überschreibe) Text, bis `ESC` getippt wird

C.7 Suchen nach Ausdrücken

Im folgenden ist *regexp* ein regulärer Ausdruck (siehe Anhang D), der eine einfache Buchstabenreihe sein kann.

/*regex* `RETURN`	vorwärts nach *regex* suchen
?*regex* `RETURN`	rückwärts nach *regex* suchen
`n`	wiederhole ursprüngliche Suche in derselben Richtung
`N`	wiederhole ursprüngliche Suche in entgegengesetzter Richtung
`/RETURN`	wiederhole ursprüngliche Suche vorwärts
`?RETURN`	wiederhole ursprüngliche Suche rückwärts

C.8 Ersetzen von Ausdrücken

Ein Ersetzungskommando hat folgende Syntax:

`:[Adresse]s/Such-Ausdruck/Ersetzungs-Ausdruck/g`

`Adresse`	besteht aus zwei Zeilennummern, die durch ein Komma getrennt sind. Die Zeilennummern können durch einen Punkt für die aktuelle Zeile, ein `$` für die letzte Zeile oder durch eine Marke (siehe Abschnitt C.11) ersetzt werden.
`Ersetzungs-Ausdruck`	ein regulärer Ausdruck, der eine einfache Buchstabenreihe sein kann
`g`	steht für globales Ersetzen; ohne das `g` wird nur die erste mögliche Ersetzung in jeder Zeile durchgeführt.

C.9 Merken von Text

In `vi` steht „yank“ für „merken, aber nicht löschen“ (also ganz anders als im Emacs, wo „yank“ für die Wiedergabe einer gekillten Sequenz steht!). Wir verwenden hier einfach den Ausdruck „merken“. Derartig gemerkter Text kann irgendwo anders in der Datei wiedereingesetzt werden. Um gemerkten Text für spätere Zwecke aufzubewahren, kann man einen benannten Puffer verwenden. Dazu wird dem Merk–Kommando ein `"x` vorangestellt, wobei `x` der Puffername (ein Buchstabe von a bis z) ist.

`y`M	merke durch M angegebenen Text
n`yy`	merke n Zeilen
`Y`	merke bis zum Zeilenende

C.10 Wiedereinfügen von Text

Nachdem Text gelöscht oder gemerkt wurde, kann er in der Datei wieder eingefügt werden („put“). Falls kein Puffer als Zwischenspeicher angegeben wurde, so dürfen zwischen dem Löschen bzw. Merken und dem Wiedereinfügen nur Cursorbewegungen stattfinden. Um Text aus dem Puffer `x` wieder einzufügen, wird dem Kommando `"x` vorangestellt.

`P`	Wiedereinfügen des Textes vor dem Cursor
`p`	Wiedereinfügen des Textes nach dem Cursor

C.11 Markieren

`m`*x*	setze Marke *x*, wobei *x* ein Buchstabe von a bis z ist
’x	gehe mit dem Cursor an den Anfang der Zeile mit Marke *x*
‘x	gehe mit dem Cursor zur Marke *x*
`''`	gehe mit dem Cursor zur vorigen Position

C.12 Shell–Befehle

`:sh`	starte eine Subshell; in den Editor zurück kommt man mit `C-d` oder `exit`
`:!Kommando`	starte Shell und führe `Kommando` aus
`!!Kommando`	starte Shell, führe `Kommando` aus und ersetze die aktuelle Zeile der editierten Datei mit der Ausgabe von `Kommando`
`:r!Kommando`	füge die Ausgabe von `Kommando` an der aktuellen Cursorpostion der editierten Datei ein

C.13 Verschiedene Kommandos

`u`	mache letzte Änderung rückgängig; funktioniert aber nur einmal
`J`	verbinde die aktuelle mit der folgenden Zeile
`.`	wiederhole das letzte verändernde Kommando
`:w Datei`	schreibe Änderungen nach `Datei` (die aktuelle Datei, falls keine `Datei` abgegeben wurde)
`:e Datei`	editiere `Datei`; erst mit `:w` die Änderungen am aktuellen File abspeichern, bevor das neue eingelesen wird!
`:f`	zeige Name und Status des aktuellen Files an
`:r Datei`	füge Datei an der aktuellen Cursorposition ein
`C-g`	zeige Dateiname, aktuelle Zeilennummer, Gesamtzeilenzahl und den Prozentsatz der Datei, der sich vor dem Cursor befindet
`C-v`	füge nächsten Buchstaben „wörtlich“ ein. Das wird im Eingabemodus zur Eingabe von Kontrollzeichen u.ä. verwendet
`~`	ändere Klein– zu Großschreibung und umgekehrt

D Einfache reguläre Ausdrücke

Reguläre Ausdrücke werden in verschiedenen Unix-Utilities und Editoren verwendet, um nach Ausdrücken zu suchen und/oder diese zu ersetzen. Dieses Kapitel beschränkt sich auf einfache reguläre Ausdrücke wie sie etwa in `vi` verwendet werden. Wer sich für den vollen Umfang der Möglichkeiten regulärer Ausdrücke interessiert, sollte einmal einen Blick in die Manualseiten zu `egrep` oder in die entsprechenden Abschnitte des Anhangs B werfen. Allerdings ist die Verwendung regulärer Ausdrücke in Emacs etwas anders als hier beschrieben. Daher sollte man auch die entsprechenden Kapitel im Emacs–Manual konsultieren.

Ein regulärer Ausdruck kann eine einfache Ansammlung von Buchstaben sein oder eine Kombination von Zeichen, die bestimmte Gruppen von Ausdrücken beschreibt. Das wird dadurch erreicht, daß einigen Zeichen spezielle Bedeutung zugewiesen wird. Will man nun eines dieser Sonderzeichen wörtlich in einem Suchausdruck verwenden, so muß man es zitieren, indem das Escape-Zeichen der regulären Ausdrücke, ein \, dem Sonderzeichen vorangestellt wird. \\ steht für den wörtlichen Rückwärts–Schrägstrich \.

Die Möglichkeit, mit Hilfe von Sonderzeichen eine ganze Gruppe von Ausdrücken zu beschreiben, ist bereits von der Dateinamen–Ergänzung der Shell her bekannt. Dort haben `*`, `?`, `[` und `]` eine spezielle Bedeutung. Reguläre Ausdrücke sind komplizierter, aber auch mächtiger.

In den meisten Fällen müssen reguläre Ausdrücke zwischen zwei gleichen Zeichen, den Begrenzern, geklammert werden. Die Begrenzer können jedes Zeichen sein, solange sie das erste Zeichen eines regulären Ausdrucks sind. D.h., daß das erste Zeichen eines regulären Ausdrucks automatisch als Begrenzer gilt und dann innerhalb dieses Ausdruckes ein Sonderzeichen darstellt. Oft wird dafür ein Schrägstrich `/` verwendet, der auch in den folgenden Beispielen diese Funktion erfüllen wird.

Der einfachste reguläre Ausdruck ist eine ganz gewöhnliche Buchstabenreihe. Sie steht nur für sich selbst. Der Ausdruck `/abc/` paßt daher zu jedem Ausdruck wie `abc`, `aabcc`, `qabch5` etc.

Sonderzeichen für reguläre Ausdrücke sind:

- **`.` (Punkt)** steht für irgendein einzelnes Zeichen (wie das `?` in der Dateinamen–Ergänzung der Shell)

- `[ ]` wie in der Shell. Dies repräsentiert ein Einzelzeichen, das sich in einer Liste in den Klammern (wie in `[abc]`) oder in einem gegebenen Bereich befindet (wie in `[a-c]`). Folgt der linken Klammer unmittelbar ein `^`, so wird jedes Einzelzeichen repräsentiert, das sich nicht in der Liste oder dem angegebenen Bereich befindet. Die Sonderzeichen `\`, `*`, `$` (siehe unten) verlieren innerhalb der Klammern ihre spezielle Bedeutung. Das `^` ist nur als erstes Zeichen innerhalb der Klammer ein Sonderzeichen. Ein Minuszeichen wird zur Angabe eines Bereichs verwendet und ist daher innerhalb der Klammern ein Sonderzeichen, außer als erstes Zeichen nach `[` oder `[^`. Die rechte Klammer `]` ist natürlich auch ein Sonderzeichen.
- `*` Ein Stern nach einem regulären Ausdruck, der ein Einzelzeichen repräsentiert, bedeutet, daß dieser Ausdruck keinmal oder mehrere Male erscheinen kann. `ab*c` würde also zu `ac`, `abc`, `abbc` etc. passen.
  Ein Stern nach einem Punkt steht für jeden beliebigen Ausdruck. Dies ist äquivalent zum Stern in der Shell.
- `^` Ein regulärer Ausdruck, der mit einem Dach `^` beginnt, steht nur für Ausdrücke am Zeilenanfang.
- `$` Ein regulärer Ausdruck, der mit einem `$` endet, steht nur für Ausdrücke am Zeilenende.
- `\(` **und** `\)` Zitierte runde Klammern werden verwendet, um Teile eines regulären Ausdrucks in Gruppen zusammenzufassen. Auf diese Gruppen kann man sich später mittels zitierter Ziffern beziehen.

Die beiden nächsten Zeichen haben in `vi` und `sed` spezielle Bedeutung beim Suchen und Ersetzen:

- Ein `&` im Ersetzungs-Ausdruck steht für den Ausdruck, der durch den Suchterm gefunden wurde.
- Im Suchstring steht eine zitierte Ziffer (`\n`) für den `n`ten Ausdruck, der in diesem Suchstring durch Klammern gruppiert wurde.
- Im Ersetzungs-Ausdruck steht eine zitierte Ziffer (`\n`) für den `n`ten Ausdruck, der im Suchstring mit Hilfe von zitierten Klammern zu einer Gruppe zusammengefaßt und bei der Suche gefunden wurde.

Um Mehrdeutigkeiten zu vermeiden, gibt es die folgenden Regeln:

- Ein regulärer Ausdruck paßt immer zum längsten möglichen Ausdruck.
- Ein leerer regulärer Ausdruck (das sind einfach zwei gleiche Zeichen hintereinander) steht für den zuletzt verwendeten regulären Ausdruck.

E Literatur

E.1 Samuel's englische Literaturliste

Ein Großteil der Unix–Literatur ist englisch geschrieben. Die folgende Liste von Samuel Ko habe ich aus den NetNews erhalten und hier mit seiner Einwilligung eingefügt. Sie gibt einen sehr guten Überblick über die englischsprachige Literatur zu Unix. Einige dieser Bücher sind auch ins Deutsche übersetzt worden, z.B.

- Das Unix V Nutshell–Buch
  Autor: Tim O'Reilly
  Verlag: Addison-Wesley

- Das BSD Unix Nutshell–Buch
  wie oben

Hier also nun Samuel's Liste:

Last Updated: August 12 1992 Copyright 1992 Samuel Ko (kko@sfu.ca)

1. General Unix Texts

   - Title: Unix for the Impatient
     Author: Paul Abrahams and Bruce Larson
     Publisher: Addison Wesley
     Edition: 1992
     Comments: **** Highly Recommended ****
     A new, comprehensive, in-depth reference to Unix. You can't find something better.
   - Title: The Unix Operating System
     Author: Kaare Christian
     Publisher: Wiley
     Edition: 2nd ed. 1988
     Comments: A classic overview of Unix commands. Very good.
   - Title: A Practical Guide to the Unix System
     Author: Mark Sobell
     Publisher: Benjamin / Cummings
     Edition: 1990
     Comments: Similar to Christian's book, slightly easier to read. There is a new edition for System V Release 4.
   - Title: The Waite Group's Unix System V Primer
     Authors: Mitchell Waite, Donald Martin and Stephen Prata
     Publisher: Hayden
     Edition: 1991
     Comments: A very good tutorial-type book for SVR4.
   - Title: Mastering SunOs
     Authors: Brent Heslop and David Angell
     Publisher: Sybex
     Edition: 1990
     Comments: A good, comprehensive hand-on text to SunOs and OpenWin.
   - Title: Peter Norton's Guide to Unix
     Author: Peter Norton and Harley Hahn
     Publisher: Bantam Computer
     Edition: 1991
     Comments: One of the many books by Peter Norton. Good coverage, but some errors.
   - Title: Unix in a Nutshell for System V
     Author: -
     Publisher: O'Reilly
     Edition: 1990
     Comments: A fine quick reference. There is an updated edition covering SVR4 and Solaris 2. Also an edition for 4.3. BSD.
   - Title: Life with Unix - A Guide for Everyone
     Author: Don Libes and Sandy Ressler

Publisher: Pre Hall
Edition: 1990
Comments: **** Highly Recommended ****
An everything-you-want-to-know-about-Unix book. It includes info you might not find elsewhere.

- Title: Learning Unix
  Author: James Gardner
  Publisher: Sams
  Edition: 1991
  Comments: With disks containing MSDOS simulation of Unix (MSK Tools) ...
  A good tutorial / reference book for those without constant access to Unix.
- Title: The Unix C Shell Field Guide
  Author: Gail Anderson and Paul Anderson
  Publisher: Prentice Hall
  Edition: 1986
  Comments: The C-Shell bible - everything you need to know to use Unix effectively.
- Title: Unix C Shell - Desk Reference
  Author: Martin Arick
  Publisher: QED Technical
  Edition: 1992
  Comments: Another text on C-Shell.
- Title: The Design of the Unix Operating System
  Author: Maurice Bach
  Publisher: Prentice Hall
  Edition: 1986
  Comments: An excellent reference on the internals of System V.
- Title: The Design and Implementation of the 4.3 BSD Unix Operating System
  Authors: Samuel Leffler et al
  Publisher: Addison-Wesley
  Edition: 1989
  Comments: An authoritative description of the design of BSD Unix.

2. Unix Editors

- Title: GNU EMACS Manual
  Author: Richard Stallman
  Publisher: Free Software Foundation
  Edition: 6th edition 1988
  Comments: The official manual of GNU Emacs.
- Title: Learning GNU Emacs
  Authors: Debra Cameron and Bill Rosenblatt
  Publisher: O'Reilly
  Edition: 1992
  Comments: **** Highly Recommended ****
  Probably the best documentation on Emacs.
- Title: Desktop Guide to Emacs
  Authors: Ralph Roberts and Mark Boyd
  Publisher: Sams
  Edition: 1991
  Comments: Another good book on emacs.

- Title: Learning the vi Editor
  Author: Linda Lamb
  Publisher: O'Reilly
  Edition: 1990
  Comments: A good guide to vi and ex commands. With a quick reference card. Other good vi stuff (including an online tutorial) is obtained by anonymous ftp from cs.uwp.edu (in pub/vi).

3. Unix Networking and Communications

- Title: Unix Networking
  Authors: Edited by Stephen Kochan and Patrick Wood
  Publisher: Hayden
  Edition: 1989
  Comments: Highly technical on different issues of networking.
- Title: Managing NFS and NIS
  Author: Hal Stern
  Publisher: O'Reilly
  Edition: 1991
  Comments: A comprehensive, technical guide on network file and info systems for sys admins.
- Title: The Waite Group's Unix Communications
  Authors: Bart Anderson, Barry Costales & Harry Henderson
  Publisher: Sams
  Edition: 2nd ed. 1991
  Comments: **** Highly Recommended ****
  An excellent book on elm, rn and nn.
- Title: Managing UUCP and Usenet
  Authors: Tim O'Reilly and Grace Todino
  Publisher: O'Reilly
  Edition: 1992
  Comments: Well written on the setting-up and the maintenance of UUCP and Netnews.
- Title: Using UUCP and Usenet
  Authors: Grace Todino and Dale Dougherty
  Publisher: O'Reilly
  Edition: 1991
  Comments: Well written on how to use uucp and netnews.
- Title: Internetworking with TCP/IP Vols I and II
  Author: Douglas Comer
  Publisher: Prentice-Hall
  Edition: 1991
  Comments: A detailed discussuion of the architecture of the Internet and its protocols. Vol I is readable by anyone but Vol II is very technical.
- Title: Zen and the Art of Internet
  Author: Brendan Kehoe
  Publisher: -
  Edition: 1st ed. 1992
  Comments: A comprehensive overview of the Internet for beginners. It's obtainable by anonymous ftp from wuarchive.wustl.edu (in mirrors/msdos/books) or other Simtel20 mirrors. The second edition is commercial.

4. Unix System Administration

   - Title: The Unix System Administration Handbook
     Authors: Evi Nemeth, Garth Snyder and Scott Seebass
     Publisher: Prentice-Hall
     Edition: 1989
     Comments: classic, also known as the sys-admin bible.
   - Title: Unix System V Release 4 Administration
     Authors: David Fiedler, Bruce Hunter and Ben Smith
     Publisher: Hayden
     Edition: 1991
     Comments: **** Highly Recommended ****
     A must for any intermediate / advanced users of Unix.
   - Title: Essential System Administration
     Author: Aeleen Frisch
     Publisher: O'Reilly
     Edition: 1991
     Comments: **** Highly Recommended ****
     A must for serious users / sys admins of Unix.
   - Title: Unix System - Advanced Administration and Management Handbook
     Author: Bruce Hunter and Karen Hunter
     Publisher: MacMillan
     Edition: 1991
     Comments: Another good book on system administration.

5. Unix Security

   - Title: Practical Unix Security
     Authors: Simson Garfinkel and Gene Spafford
     Publisher: O'Reilly
     Edition: 1991
     Comments: **** Highly Recommended ****
     Simply the best book in this field.
   - Title: Unix System Security - A Guide for Users and System Administrators
     Author: David Curry
     Publisher: Addision Wesley
     Edition: 1992
     Comments: Comprehensive coverage with pointers to further info.
   - Title: Unix System Security
     Author: Rik Farrow
     Publisher: Addison Wesley
     Edition: 1991
     Comments: Fine but not as technical as other books.
   - Title: Site Security Handbook
     Authors: Edited by P. Holbrook and J. Reynolds
     Publisher: -
     Edition: 1991
     Comments: This is rfc1244.txt which is available by anonymous ftp from nic.ddn.mil (in rfc directory). *(Anm.: Die RFCs sind auch auf vielen deutschen ftp-Servern vorhanden, z.B. auf ftp.informatik.tu-muenchen.de)*

6. Programming

- Title: The Unix Programming Environment
  Authors: Brian Kernighan and Rob Pike
  Publisher: Prentice-Hall
  Edition: 1984
  Comments: A classic book on Unix programming.
- Title: The C Programming Language
  Authors: Brian Kernighan and Dennis Ritchie
  Publisher: Prentice Hall
  Edition: 1988
  Comments: This book is a MUST for any C programers.
- Title: C - A Reference Guide
  Authors: Samuel Harbison and Guy Steel
  Publisher: Prentice Hall
  Edition: 1991
  Comments: An authoritative reference to C progamming language. A good companion to Kernighan and Ritchie.
- Title: The Waite Group's New Primer C Plus
  Authors: Mitchell Waite and Stephen Prata
  Publisher: Sams
  Edition: 1990
  Comments: A really good introduction to C for beginners.
- Title: Practical C Programming
  Author: Steve Oualline
  Publisher: O'Reilly
  Edition: 1991
  Comment: Yet another book on C.
- Title: Using C on the Unix System
  Author: David Curry
  Publisher: O'Reilly
  Edition: 1990
  Comments: This book is directed to (would-be) system programmers.
- Title: C Programming in the Berkeley Unix Environment
  Author: Nigel Horspool
  Publisher: Prentice Hall
  Edition: 1986
  Comments: A C book for 4.3 BSD.
- Title: The Waite Group's C++ Programming
  Author: John Berry
  Publisher: Sams
  Edition: 1992
  Comments: A fine on C++ (for Unix and Dos).
- Title: Programming Perl
  Authors: Larry Wall and Randal Schwartz
  Publisher: O'Reilly
  Edition: 1991
  Comments: The authoritative guide to Perl - the programming language for for serious Unix users.
- Title: Managing Projects with make
  Authors: Steve Talbott and Andrew Oram
  Publisher: O'Reilly

Edition: 2nd ed. 1991
Comments: An unique book on using make and handling Makefile.

7. TEX

- Title: The TeXbook
  Author: Donald Knuth
  Publisher: Addison Wesley
  Edition: 1990
  Comments: The bible of TeX users.
- Title: TeX for the Impatient
  Author: Paul Abrahams
  Publisher: Addison-Wesley
  Edition: 1991
  Comments: A more user-friendly handbook on TeX.
- Title: LATEX- A Document Preparation System
  Author: Leslie Lamport
  Publisher: Addison Wesley
  Edition: 1986
  Comments: The bible for LATEXusers.

8. X Windows

- Title: X Window System Users' Guide
  Authors: Valerie Quercia and Tim O'Reilly
  Publisher: O'Reilly
  Edition: 1990 (MIT or Motif)
  Comments: A useful tutorial type book to X11R4
- Title: The X Window System - A User's Guide
  Author: Niall Manfield
  Publisher: Addison Wesley
  Edition: 1990
  Comments: Another good tutorial-type book to using X.

9. Dictionary

- Title: The New Hacker's Dictionary
  Author: Eric Raymond
  Publisher: MIT Press
  Edition: 1991
  Comments: This book corresponds to version 2.9.6 of the on-line jargon file. The latest (at the time of writing) is version 2.9.10 (jargon2910.ascii.Z) which is available by ftp from wuarchive.wustl.edu (in mirrors/gnu).

E.2 Deutsche Literaturhinweise

Hier also nun einige deutsche Literaturangaben. Dabei handelt es sich sowohl um Bücher, die ursprünglich auf deutsch erschienen, sowie um Übersetzungen von Büchern, die in der englischen Liste nicht erwähnt werden. Achten Sie auch auf Übersetzungen der Bücher in der englischen Liste.

1. Netzwerke

- Titel: Computer–Netzwerke
  Autor: Andrew S. Tanenbaum

Verlag: Wolfram's Fachverlag, 1990
Kommentar: Ein 800-Seiten -Klassiker, an dem Leute, die sich ernsthaft mit Netzwerken befassen müssen, nicht vorbeikommen. Neuere Entwicklungen werden aber nicht abgedeckt. Trotzdem sehr empfehlenswert!

- Titel: Programmieren von Unix-Netzen
  Autor: W. Richard Stevens
  Verlag: Hanser 1992
  Kommentar: Ein gutes Buch über Netzwerk-Protokolle und Implementierung von TCP/IP-Anwendungen. Setzt moderate Kenntnisse in Unix und natürlich C voraus. Die deutsche Version leidet zwar unter den üblichen Übersetzungsfehlern, ist aber deutlich billiger als die englische Ausgabe.
- Titel: TCP/IP und NFS in Theorie und Praxis
  Autor: Michael Santifaller
  Verlag: Addison-Wesley, 1990
  Kommentar: Eine gute Einführung in TCP/IP und NFS für Systemverwalter und Entwickler verteilter Anwendungen. Nicht so technisch wie die Bücher von Comer.
- Titel: Lokale Netze
  Autor: Franz-Joachim Kauffels
  Verlag: Datacom, 1989
  Kommentar: Kein technisches Buch, sondern ein Überblick über die verfügbaren Protokolle und den technischen Stand bei lokalen Netzwerken. Die Informationen sind gut aufbereitet.

2. Programmierung

- Titel: Der C-Experte
  Autor: Andrew Koenig
  Verlag: Addison-Wesley, 1989
  Kommentar: Ein sehr gutes Buch für Leute, die gerade C gelernt haben und in die typischen C–Fettnäpfchen treten. Es befaßt sich mit den häufigsten Fehlern bei der C-Programmierung. Enthält Aufgaben mit Lösungen.
- Titel: Das C-Puzzle-Buch
  Autor: Verlag: Kommentar: Erläutert anhand von „Rätseln" (was tut dieses Programm?) die Vertracktheiten von C. Gute Ergänzung zu Koenigs Buch.

3. X

- Titel: X und Motif
  Autor: K. Gottheil u.a.
  Verlag: Springer 1992
  Kommentar: gute Einführung in die Programmierung des Motif-Toolkits und des X-Window-Systems.

4. TEX und LATEX

- Titel: Einführung in TEX
  Autor: Norbert Schwarz
  Verlag: Addison-Wesley
  Kommentar: Benutzerfreundliches Buch für TEX–Novizen.
- Titel: TEX für Fortgeschrittene
  Autor: Wolfgang Appelt
  Verlag: Addison-Wesley
  Kommentar: Programmiertechniken in TEX werden verständlich dargestellt. Ergänzt das Buch von Knuth.

- Titel: LaTeX–Eine praktische Einführung
  Autor: Helmut Kopka
  Verlag: Addison-Wesley
  Kommentar: Sehr zu empfehlen! Das Buch für Leute, die ein gutes technisches Dokument in LaTeX schreiben wollen, ohne allzuviel Ahnung von Rechnern zu haben.
- Titel: LaTeX–Erweiterungsmöglichkeiten
  Autor: Helmut Kopka
  Verlag: Addison-Wesley, 1991
  Kommentar: Für Benutzer, die an LaTeX rumbasteln möchten.

Index