Kurzeinführung in die Arbeit unter UNIX

Stand: 25. Juli 2001

Klaus Könnecke

Institut
für
Numerische und Angewandte Mathematik

Universität Göttingen

Inhaltsverzeichnis

1 Grundlagen von UNIX 1
- 1.1 Vom Umgang mit den Maschinen . . . 1
- 1.2 Anmelden (*Login*) . . . 1
- 1.3 Pause . . . 2
- 1.4 Abmelden (*Logout*) . . . 2
- 1.5 Der UNIX–Kern . . . 2
- 1.6 Kommandointerpreter (*Shell*) . . . 3
- 1.7 UNIX–Prozesse (Vorder– und Hintergrundprozesse) . . . 3
- 1.8 Allgemeines zu UNIX–Dateien . . . 3
  - 1.8.1 Dateinamen . . . 4
  - 1.8.2 Dateinamenserweiterung . . . 4
  - 1.8.3 Versteckte Dateien (Punkt–Dateien) . . . 4
  - 1.8.4 Dateiarten . . . 5
  - 1.8.5 Erzeugen von Dateien . . . 5
  - 1.8.6 Zugriffsrechte für Dateien . . . 5
- 1.9 Das hierarchische UNIX–Dateisystem . . . 7
- 1.10 Punkt- und Punkt-Punkt-Verzeichnisse . . . 8
- 1.11 Pfadnamen . . . 8
- 1.12 Verzeichnisse (*Directories*) . . . 8
- 1.13 UNIX–Kommandos . . . 9
  - 1.13.1 Einfache Kommandos . . . 9
  - 1.13.2 Zusammengesetzte Kommandos . . . 9
  - 1.13.3 Kommando–Trenner . . . 10
  - 1.13.4 Kommando–Gruppen . . . 10

2 Benutzung der Maus 11

3 Emacs: Text-Editor 12
- 3.1 Starten des Emacs–Editors . . . 13
- 3.2 Verlassen des Emacs–Editors . . . 13
- 3.3 Fehlerbehandlung . . . 13
- 3.4 Laden von Dateien . . . 13
- 3.5 Positionieren in der Datei . . . 14
- 3.6 Löschen und wieder Einsetzen . . . 14
- 3.7 Markieren . . . 14
- 3.8 Suchen . . . 15
- 3.9 Ersetzen . . . 15
- 3.10 Vertauschen . . . 16
- 3.11 Änderung von Groß– und Kleinschreibung . . . 16
- 3.12 Puffer und Fenster . . . 16
- 3.13 Der Minipuffer . . . 16
- 3.14 Hilfe . . . 17

4 Arbeiten mit der Shell 18
- 4.1 Standard–Eingabe und Standard–Ausgabe . . . 18
- 4.2 Umlenken von Standarddateien . . . 18
- 4.3 pipe (Verknüpfen von Kommandos) . . . 19

4.4 Filter . . . 19
4.5 Automatische Ergänzung von Dateinamen (*Wildcards*) . . . 19
4.6 Aliase . . . 20

5 Hilfsprogramme unter UNIX **21**
5.1 Hilfestellungen . . . 21
5.1.1 man (Informationen über das UNIX–Handbuch) . . . 21
5.1.2 whatis (Kurzbeschreibung von UNIX–Kommandos) . . . 21
5.1.3 apropos (Suchen nach Informationen) . . . 21
5.2 pwd (das aktuelle Verzeichnis (*Working Directory*)) . . . 21
5.3 cd (Wechseln in ein neues aktuelles Verzeichnis) . . . 21
5.4 ls (das Auflisten von Dateien) . . . 22
5.5 file (Die Dateiattribute bestimmen) . . . 22
5.6 who (Auflisten aktiver Benutzer) . . . 23
5.7 ps (Prozesse anzeigen) . . . 23
5.8 kill (Abbrechen von Prozessen) . . . 23
5.9 yppasswd (Ändern des Sicherheitswortes (*Password*)) . . . 24
5.10 find (Suchen nach Dateinamen) . . . 24

6 Dateiverwaltung **26**
6.1 mkdir (Erzeugen von Verzeichnissen (*Directories*)) . . . 26
6.2 rmdir (Löschen von Verzeichnissen (*Directories*)) . . . 26
6.3 rm (Dateien löschen) . . . 26
6.4 mv (Dateien umbenennen) . . . 27
6.5 cp (Dateien kopieren) . . . 27
6.6 ln (Verweise (*Links*) auf Dateien) . . . 27
6.6.1 Hard–Link . . . 27
6.6.2 Symbolischer Link . . . 28
6.7 chmod, chown, chgrp (Dateimodus ändern (Zugriffsrechte)) . . . 28
6.8 umask (Unterbinden von Zugriffsrechten) . . . 30

7 Hilfsprogramme für Textdateien **31**
7.1 cat (Verknüpfe und drucke Dateien) . . . 31
7.2 pr (Dateien betiteln und formatieren) . . . 31
7.3 more (Seitenweises Portionieren der Bildschirmausgabe) . . . 31
7.4 lpr (Dateien ausdrucken, Druckkommandos) . . . 32
7.4.1 Druckernamen, Druckerstandorte . . . 33
7.4.2 Druckerwarteschlangen abfragen . . . 33
7.4.3 Drucken von dvi–Dateien, doppelseitig und deckungsgleich . . . 34
7.5 wc (Zählen von Zeilen, Worten und Zeichen) . . . 34
7.6 diff (Vergleichen von Dateien oder Verzeichnissen) . . . 34
7.7 sort (Dateiinhalte sortieren oder vermischen) . . . 35
7.8 grep (Textmuster in Dateien suchen) . . . 36
7.9 cut und paste (Zeilenteile ausschneiden und zusammenfügen) . . . 37
7.10 tee (Ausgabe verdoppeln) . . . 38
7.11 tail (Das Ende einer Datei ausdrucken) . . . 38

8 Compiler **39**
8.1 Allgemeines . . . 39
8.2 Optionen . . . 39
8.3 make (Compileraufrufe automatisieren) . . . 40

9 Arbeiten mit dem Netzwerk **41**
9.1 TCP/IP . . . 41
9.2 Rechneradressen (*Domaine–Name–Service*) . . . 41
9.3 ~~TELNET~~ . . . 41
9.4 ~~FTP~~ . . . 41
9.5 PING . . . 42
9.6 ~~R–Kommunikationsprogramme~~ . . . 42
9.6.1 rwho . . . 42
9.7 S–Kommunikationsprogramme . . . 42
9.7.1 ssh . . . 42
9.7.2 slogin . . . 42
9.7.3 scp . . . 42
9.8 Elektronische Post . . . 43
9.8.1 write . . . 43
9.8.2 mail . . . 43
9.8.3 dxmail . . . 44
9.8.4 rmail (Briefpost–Dienst im EMACS–Editor) . . . 44
9.8.5 Netscape–Mailbox . . . 44

10 Datensicherung **45**
10.1 tar (Erzeugen von Archiv–Dateien) . . . 45
10.2 Komprimieren und Entkomprimieren von Dateien . . . 45
10.3 Komprimieren von Dateien im Emacs–Verzeichnis–Editor (*dired*) . . . 46

Stichwortverzeichnis **46**

1 Grundlagen von UNIX

UNIX ist ein Betriebssystem (*Operating System*) zur Verwaltung der Betriebsmittel eines Rechners. Betriebssysteme erleichtern die Aufgabe der Prozeßverwaltung und des Dateisystems. UNIX ist ein eingetragenes Warenzeichen der USL (*UNIX System Laboratories*). Deshalb benennen andere Hersteller ihre UNIX–Betriebssysteme mit anderen Namen. Das an den Rechnern im Institut für Angewandte und Numerische Mathematik eingesetzte Bestriebssystem heißt Digital–UNIX und ist von der Firma DEC. Es ist wichtig zu wissen, daß UNIX Groß– und Kleinbuchstaben unterscheidet.

1.1 Vom Umgang mit den Maschinen

Ein UNIX–Rechner darf **nur** vom **Systemverwalter** oder vom **Operating** unter Verwendung bestimmter Prozeduren **ein**– oder **ausgeschaltet** werden. Unbefugtes Ausschalten kann großen Schaden an der Konfiguration der Rechner anrichten!

Einschalten der Bildschirme:

Institut	Standort	Wo Bildschirm einschalten
NUMERIK	Räume im Erdgeschoß	**DEC–Rechner:** Schalter am Bildschirm. **PC–Rechner:** großer, blauer Schalter an Zentraleinheit
NUMERIK	Kellerraum	**Rechner x50 - x55 am Schalter hinten rechts an der Zentraleinheit** (Unterbau vom Bildschirm). Sonst Schalter am Bildschirm.
NUMERIK	Vorraum Bibliothek	Schalter am Bildschirm.
MATHE	Terminalraum	Schalter am Bildschirm.

Drucker: **dürfen nur nach Absprache aus– oder eingeschaltet werden**.

Papierstaus an den Druckern: sind beim Operating zu melden und dürfen außerhalb der Dienstzeit des Operating nur von eingewiesenen Mitarbeitern behandelt werden.

Bildschirmschoner: Die Bildschirme werden automatisch abgedunkelt ("Sternenhimmel"), wenn längere Zeit keine Eingabe erfolgte. Durch Bewegen der Maus kann der Bildschirm wieder aktiviert werden.

Power Save: Wenn eingeschaltete Bildschirme der Rechner **u28 - u44** ca. 1 Stunde nicht benutzt werden, tritt ein Stromsparprogramm (*Power Save*) in Kraft (Durch Berühren der Maus keine Bildschirmaktivität). Bildschirm-Power-Schalter **aus-** und wieder **einschalten**.

1.2 Anmelden (*Login*)

Um eine Sitzung an den Rechnern zu starten, muß sich ein Benutzer anmelden, damit das System die Benutzungsechte abprüfen und die individuelle Benutzerumgebung einrichten kann.

Login-Aufforderung: zugewiesenen Benutzernamen eingeben.

Paßwort-Aufforderung: Sicherheitswort (*Password*) eingeben. Das *Password* wird "blind" eingetippt, d.h.es wird auf dem Bildschirm **nicht mitprotokolliert**.

Nach einem korrekten Anmelden erscheinen bei einem Standard-Account automatisch drei Fenster:

dxconsole-window	Enthält wichtige Kurzmitteilungen (Software-Installationen, Ausfallzeiten, usw.)
Session-Mannager-window	Enthält eine Menüleiste, deren Einzel-Menüpunkte per *Mouse-Handling* ausgeführt werden können.
xterm-window	Arbeitsfenster, in dem die *Shell* (Kommandointerpreter) einen *Prompt* (Eingabeaufforderung) vorspielt. Hier können Eingaben getätigt werden, die von der *Shell* ausgeführt werden.

1.3 Pause

Mit dem Kommando **Pause** besteht die Möglichkeit, eine Sitzung **kurzfristig** zu unterbrechen, ohne sich abmelden zu müssen. Kurzfristig bedeutet, **die Abwesenheit des Benutzers am Rechner darf nicht länger als 5 Minuten dauern!** Die Begründung resultiert aus der großen Zahl von Benutzern, die alle eine Chance zum Rechnen bekommen sollen und müssen.

Pause im Pull-down-Menü des Session-Manager-Windows: zuerst **Session**, dann **Pause** anklicken.

1.4 Abmelden (*Logout*)

Um eine Sitzung zu beenden, müssen alle gestarteten Prozesse (z.B. Emacs, Kalender, Netscape, Xterm, usw) **einzeln** geschlossen werden, damit keine CPU-verbrauchenden 'Prozeßleichen' zurückbleiben.

1. Prozesse **einzeln** schließen (**C-d** oder **exit**) oder entsprechender Menüknopf.
2. Session-Manager beenden (im Pull-down-Menü des Session-Manager-Windows erst Session und dann End Session anklicken).

Rechner	**dürfen nicht ausgeschaltet werden!**
Bildschirme	**sollen nur zum Dienstschluß vom Dienstpersonal und nach Dienstschluß vom letzten Benutzer abgeschaltet werden!**

Achtung! `C-z`, falls von anderen Systemen her bekannt, bewirkt unter UNIX lediglich das Stoppen eines Jobs (dieser Prozeß ruht, pausiert; er ist **nicht** entfernt (siehe Abschnitt 5.8) und kann durch Eingabe des Kommandos **fg** oder **bg** fortgesetzt werden (siehe Abschnitt 1.7).

1.5 Der UNIX–Kern

Der Kern von UNIX ist das Kernstück des Betriebssystems. Der Kern überwacht die Prozesse, stellt Speicher und Plattenplatz bereit, kontrolliert die Übertragung zwischen dem Hauptspeicher und den Peripheriegeräten und erfüllt die Anforderungen der Prozesse an das Gesamtsystem. Der Kern führt keine Anweisungen des Benutzers aus. Er stellt seine Dienstleistungen ausschließlich über Systemaufrufe zur Verfügung. Viele Werkzeuge (Tools, Utilities) liegen außerhalb des Kerns und damit in der Verantwortung des Benutzers.

1.6 Kommandointerpreter (*Shell*)

Alle UNIX-Systeme verfügen über einen Kommandointerpreter, die sogenannte **Shell**. Der Kommandointerpreter wird als Shell bezeichnet, weil er wie eine Hülle um den Kern (*kernel*) des Betriebssystems liegt. Während einer UNIX-Sitzung kommuniziert der Benutzer nicht direkt mit dem Betriebssystem, sondern mit der Shell. Die Shell interpretiert die eingegebenen Kommandos und veranlaßt das Betriebssystem die entsprechenden Aktionen durchzuführen. Mit der Entwicklung unterschiedlicher Systeme ist auch eine Reihe verschiedener Shells entstanden.

Die wichtigsten sind die C-Shell, die Bourne-Shell, die Korn-Shell. Im Institut für Numerische und Angewandte Mathematik wird standardmäßig beim Einrichten eines Accounts die Korn-Shell zugewiesen. Eine Einführung in die Programmierung der K-Shell existiert beim Informationssystem WWW auf der **Home-Page** des **Instituts für NAM** unter **Dokumentation**.

Jedes interaktive Betriebssystem stellt seinen Benutzern eine Sammlung vorgefertigter, hilfreicher Programme zur Verfügung. Diese Programme heißen **Systemprogramme** oder **Dienstprogramme** (*Tools, Utilities*). Sie bilden als eine Art Werkzeugkasten die **Toolbox** des Systems. Typische Werkzeuge sind beispielsweise Programme zum Anlegen, Editieren, Kopieren, Sortieren und Löschen von Dateien. Ihr Aufruf erfolgt bei UNIX-Systemen mit Hilfe der Shell.

1.7 UNIX–Prozesse (Vorder– und Hintergrundprozesse)

Die Abarbeitung eines Programms heißt Prozeß. Ein Pozeß enthält Handlungsanweisungen für einen Rechner. Durch die Eingabe eines Kommandos an die Shell wird ein Programm geladen und zur Ausführung gebracht. Damit ist ein Prozeß entstanden, der solange dauert (lebt), bis das Programm beendet ist. Ein (einziges) Programm (Mutterprozeß) kann zu mehreren (parallelen, nebenläufigen) Prozessen (Kind- und Enkelprozesse) führen.

Vorder– und Hintergrundprozesse

Bei einem **Vordergrundprozeß** ist nach dem Starten eines Prozesses das Xterm-Fenster für weitere Eingaben solange blockiert, bis alle Kommandos des gestarteten Prozesses abgearbeitet sind. Erst nach dem Ende des Prozesses erscheint wieder der Prompt für neue Kommandoeingaben.

Ein **Hintergrundprozeß** arbeitet gleichzeitig (parallel) mit der Shell im Vordergrund, d.h. trotz des gestarteten Hintergrundprozesses können im Vordergrund Kommandos ausgeführt werden. Dem Kommando eines Hintergrundprozesses muß am Ende das Sonderzeichen "&" angehängt werden.

cat adressen	Aufruf eines Vordergrundprozesses.
emacs Dateiname &	Aufruf eines Hintergrundprozesses.
jobs	Zeigt gestoppte Jobs und im Hintergrund laufende Jobs an.
fg	Setzt einen gestoppten Job im Vordergrund fort.
bg	Setzt einen gestoppten Job im Hintergrund fort.

Ein Job kann durch **CTRL-Z** und **bg** nachträglich in den Hintergrund gebracht werden. (Anzeigen gestoppter Jobs: siehe Abschnitt 5.8).

1.8 Allgemeines zu UNIX–Dateien

Der Begriff *Datei* ist ein Konzept zur Verwaltung von Daten. Es befreit den Anwender von Kenntnissen über die konkreten physikalischen und elektronischen Eigenschaften von Datenträgern wie Festplatten, Disketten und Magnetbändern und der zu ihnen gehörenden Geräte. Es werden Dateien und indirekt ihre Daten mit einem Dateinamen angesprochen, ohne beispielsweise zu wissen, in welchen Sektoren und Spuren diese auf einer Diskette abgelegt wurden. Die Dateneinheit ist das *Byte*, ein Acht–Bit–Muster.

1.8.1 Dateinamen

Jede Datei hat einen Namen, der bis zu 256 Zeichen lang sein darf. Außer dem Nullbyte (ASCII, dezimal 0), dem Schrägstrich (ASCII, dezimal 47) und dem Doppelpunkt (ASCII, dezimal 58) können alle Zeichen des ASCII–Zeichenvorrats verwendet werden. Da die Shell auch Zeichen mißinterpretieren kann, sollte man sich auf folgende Zeichen im Dateinamen beschränken:

- Klein– und Großschreibung
- Ziffern
- Sonderzeichen Nullbyte, Schrägstrich und Doppelpunkt sind verboten!

1.8.2 Dateinamenserweiterung

Im Gegensatz zu MS–DOS verwendet UNIX keine Extensions (das sind in DOS durch einen Punkt abgetrennte Dateinamensteile, die für das Betriebssystem spezielle Bedeutungen haben). Unter UNIX dürfen beliebig viele Punkte in einem Dateinamen verwendet werden, da der Punkt keine spezielle Bedeutung hat. Die Feststellung, ob eine Datei ausführbar ist, erfolgt in UNIX über andere Mechanismen. Aber Ausnahmen bestätigen die Regel. So benutzen 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 (unvollständiger Auszug):

.c	C–Quellcode–Datei
.cc	C++ Quellcode–Datei
.f	FORTRAN–Quellcode–Datei (manchmal auch .f77 oder .for)
.p	Pascal–Quellcode–Datei
.s	Assembler–Datei
.o	compilierte (aber noch nicht gebundene) Objektdatei
.tex	TEX–Quellcode
.dvi	übersetztes TEX
.ps	Postscript–Dateien (Postscript ist eine Graphiksprache)
.tar	Archiv–Datei
.gz	mittels *gzip* komprimierte Datei
.Z	mittels *compress* komprimierte Datei

1.8.3 Versteckte Dateien (Punkt–Dateien)

Dateinamen, die mit einem Punkt beginnen, sind *unsichtbar* oder *versteckt*, weil sie vom **ls**–Kommando nicht angezeigt werden. Es handelt sich meist um *Startup–Dateien*, deren Kommandos bei einer Anmeldung beim System automatisch abgewickelt werden. Die Unsichtbarkeit ist mehr symbolischer Natur, da das ls–Kommando mit der Option –a auch Punktdateien anzeigt. (**ls –al**)

```
-rwx------ 1 user    72 Aug  3 11:28 .kshrc                # Beispiel: Punktdatei
```

1.8.4 Dateiarten

UNIX versteht unter einer Datei eine *Folge von Bytes*, die gelesen oder geschrieben werden und dabei wie ein Strom an den Werkzeugen vorbeifließen. Dateien haben keine von außen erkennbare *Struktur*. Jede Art von Struktur muß von den beteiligten Prozessen inhaltlich festgelegt werden. Eine nähere Betrachtung zeigt jedoch, daß UNIX zwischen verschiedenen Arten von Dateien unterscheidet: Gewöhnliche Dateien, Dateiverzeichnisse (Directories), blockorientierete und zeichenorientierte Gerätedateien und Symbolische Links.

Das Kommando **ls –l** (*Long Listing*) gibt ganau in der ersten Spalte einer Zeile die Dateiart an:

```
↓
-rwxr--r--  1 user  72  Aug 3 11:28  hypo             Gewöhnliche Datei
drwxr--r--  2 user  72  Aug 3 11:28  eiffel           Dateiverzeichnis (Directory)
brw-------  1 root   7  Aug 3  1991  rz0h             Blockorientierte Gerätedatei
crw-rw-rw-  1 root   0  Aug 3 11:28  tty              Zeichenorientierte Gerätedatei
lrwxr-xr-x  1 root   7  Jan 8  1992  tmp -> var/tmp   Symbolischer Verweis (Link)
```

Gewöhnliche Dateien sind unter UNIX z.B. Textdateien, Objektdateien (mit Maschinensprache als Inhalt), Datenbankdateien. UNIX verbindet mit ihnen keinerlei *von außen* erkennbare Struktur. Lediglich einzelne Dienstprogramme interpretieren bestimmte Zeichen einer Datei als bestimmte Funktionen. So sorgt beispielsweise nicht UNIX (der UNIX–Kern), sondern das **cat**–Kommando dafür, daß bei einer Ausgabe einer Textdatei das ASCII–Zeichen LF (dezimal 010) als NEWLINE (gehe an den Anfang einer neuen Zeile) interpretiert wird.

Dateiverzeichnisse (Directories) enthalten Verweise (Zeiger) auf die zugehörigen Dateien. Die Zeiger verweisen auf sogenannte *Inodes* (*Index Nodes, das ist die Datenstruktur, die das System zur Verwaltung von Dateien verwendet*). In ihnen wird z.B. festgehalten, wann eine Datei angelegt worden ist, wem sie gehört, wieviel Verweise (Links) sie enthält, wie groß sie ist (Anzahl der Bytes) und wo sie auf der Festplatte zu finden ist (Adressen der Plattenblöcke).

Blockorientiert sind in der Regel alle Massen–Datenträger wie Festplatten und Disketten. Ein Block ist traditionell ein ganzzahliges Vielfaches von 512 Bytes und stellt die Transporteinheit zwischen Datenträger und Massenspeicher dar.

Zeichenorientiert sind Geräte wie Tastatur, Bildschirm und Drucker. Bei ihnen ist die Übertragungseinheit zwischen Gerät und Hauptspeicher jeweils ein Zeichen.

Symbolischer Verweis: Ein symbolischer Verweis ist eine Datei, die einen Pfadnamen enthält, der auf die Datei verweist, auf den der symbolische Verweis zeigt. Symbolische Verweise sind dateisystemübergreifend und können auch auf Verzeichnisse zeigen (siehe auch Abschnitt 6.6.1).

1.8.5 Erzeugen von Dateien

Dateien werden auf vielfältige Weise erzeugt. Erzeugungsmöglichkeiten sind z.B. die Benutzung von Editoren (siehe Abschnitt 2) oder Prozesse, deren Standard–Ausgabe in eine Datei umgelenkt wird (siehe Abschnitt 4.2).

1.8.6 Zugriffsrechte für Dateien

Der Dateischutz, den UNIX realisiert, berücksichtigt zum einen, daß es für jede Datei unterschiedliche Arten von Benutzern gibt, und zum anderen, daß auf eine Datei in unterschiedlichen Arten zugegriffen werden kann. UNIX unterscheidet drei Benutzer– und drei Zugriffsarten.

Benutzerarten:

Zu jeder Datei gibt es genau einen Besitzer (*Owner*). Das ist in der Regel auch der Erzeuger der Datei (Dateien können auch ihren Besitzer wechseln).

Der Besitzer einer Datei gehört als UNIX–Benutzer zu einer Gruppe (*Group*), er besitzt eine Gruppenkennung (*Group Identifier, GID*). Die Gruppenzuweisung kann über das Kommando **ls –alg** abgefragt werden (Gruppenzuweisung eines Standard–Accounts = 'users'). Gruppe ist der Rest einer zugewiesenen oder selbstbstimmten Arbeitsgruppe, **ohne den Besitzer der Datei**.

Als dritte Benutzerart gibt es noch 'alle anderen Benutzer' außer den bereits genannten. Sie werden unter dem Begriff *Others* geführt.
Für alle drei Arten werden folgende Buchstabenkürzel verwendet:

Kürzel	Benutzerart	Erklärung
u	**u**ser	genau eine Person, das ist der Besitzer (*Owner*) selbst.
g	**g**roup	Null oder mehr Personen, ohne den Owner, das kann z.B. ein ausgewählter Kreis von Mitarbeitern sein.
o	**o**thers	das sind i.d.R. alle anderen Benutzer.
a	**a**ll	Zusammenfassung der drei Benutzerklassen `user, group` und `others`.

Zugriffsarten:

Auf eine Datei kann lesend, schreibend oder ausführend zugegriffen werden. Ist eine Datei ein Dateiverzeichnis, werden die Zugriffsarten folgendermaßen interpretiert: Ein Verzeichnis zu lesen bedeutet, seine Einträge zu vergleichen. Es zu beschreiben heißt, daß Einträge in ihm erzeugt oder gelöscht werden. Ein Dateiverzeichnis auszuführen macht keinen Sinn. Dafür gibt es hier eine andere Art des Zugriffs. Ein Verzeichnis kann mit dem **cd**–Kommando betreten werden, um eine Liste der angelegten Dateien zu erstellen (**ls**) oder um ein Unterverzeichnis zu erreichen. Dazu muß der der Zugriff auf die Einträge eines Verzeichnisses erlaubt sein. Ein Zugriff kann über Zugriffsrechte erlaubt oder verboten (geschützt) werden.

Kürzel	Zugriffsart	Datei	Dateiverzeichnis
r	**r**ead	Lesen	Einträge vergleichen, Listen anfertigen
w	**w**rite	Schreiben	Einträge erzeugen und löschen
x	**x**ecute	Ausführen	Verzeichnisse betreten, auf Einträge zugreifen

Zugriffsrechte können einzig und allein (abgesehen vom Systemverwalter) vom Besitzer einer Datei vergeben und entzogen werden. Der Besitzer einer Datei ist einerseits **selbst verantwortlich** und andererseits **zur Sicherheit verpflichtet**, was gegenüber anderen Benutzern erlaubt sein soll. Standardzuweisungen sind zu freizügig der anderen Benutzerwelt gegenüber. Dieses zeigt sich z.B. bei der Gruppenzuweisung eines Standard–Accounts. Alle Benutzer bekommen die Gruppe 'users' zugewiesen. Damit ist normalerweise eine Datei für alle Benutzer der Maschine offen (nicht geschützt), falls die Datei für `'group'` offen ist!

Schutzcode:

UNIX kennt drei Zugriffsrechte (*Permissions*), die für jede Benutzerart getrennt angegeben werden. Das Neun–Bit–Muster der Zugriffsrechte heißt Schutzcode der Datei und wird immer in der Reihenfolge `Owner, Group` und `Others` angegeben, wobei innerhalb jeder Benutzerklasse von links beginnend die folgende Reihenfolge gilt: *read, write* und *execute*.

```
drwxr-x---   1  benutzer   gruppe      379 Jul  5   10:17 kurs
-rwxr-x--x   1  benutzer   gruppe      117 Jul  18  09:54 aufg1
```

Gruppen-zugehörigkeit
Eigentümer
Anzahl der Verweise
Dateiname
Veränderungsdatum
Dateigröße in Bytes
Zugriffsrechte für **o***thers (lesen und schreiben nicht erlaubt; ausführen erlaubt)*
Zugriffsrechte für **g***roup (lesen und ausführen erlaubt; schreiben nicht erlaubt)*
Zugriffsrechte für **u***ser (lesen, schreiben und ausführen erlaubt)*
*Dateityp (***d** *= Directory-Datei;* **-** *= normale Datei)*

Abbildung 1: Zugriffsrechte für Dateien oder Verzeichnisse.

Ein Recht zum Lesen, Schreiben und Ausführen besteht immer dann, wenn auch der entsprechende Buchstabe (**r**, **w**, **x**) eingetragen ist. Ein "–" (Minuszeichen) kennzeichnet das Verbot (Schutz) für ein bestimmtes Recht.

1.9 Das hierarchische UNIX–Dateisystem

Die Dateisystemstruktur unter UNIX ist hierarchisch. Ausgangspunkt ist das *root–Verzeichnis*, das immer durch einen führenden "/" (*slash*) gekennzeichnet wird. Von dort aus entwickelt sich das Dateisystem baumartig. So verweist das root–Verzeichnis auf Dateien, die normalerweise wiederum Verzeichnisse sind. Diese können weiter auf Verzeichnisse verweisen, usw. Durch eine Abfolge solcher Verweise läßt sich ein beliebig komplexer Baum als *Dateisystem* verwalten. Diese Baumstruktur läßt sich von jedem Benutzer dynamisch erweitern.

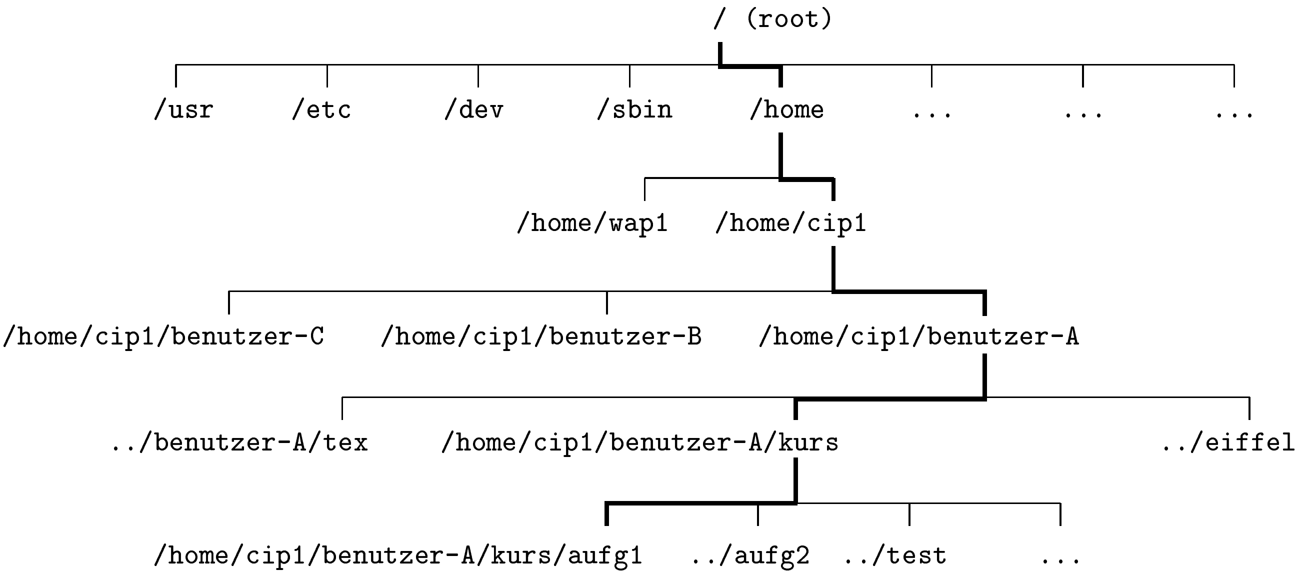

Abbildung 2: Das hierarchische UNIX–Dateiensystem.

Jede Datei hat einen Namen. Eine spezielle Datei, die die Namen anderer Dateien enthält, wird Verzeichnis (*Directory*) genannt. Verzeichnisse werden zur Strukturierung der Dateien auf einem System verwendet. Auch die voreingestellten Eingabe– und Ausgabekanäle sind unter UNIX spezielle Dateien.

1.10 Punkt- und Punkt-Punkt-Verzeichnisse

Zwei Einträge werden automatisch beim Anlegen eines Verzeichnisses (Directory) angelegt und können daraus auch nicht entfernt werden. Es sind dies die Einträge für die Dateien:

`.` (Punkt)
`..` (Punkt–Punkt)

Der Name *Punkt* ist ein Synonym für das jeweilige aktuelle Verzeichnis (*working Directory*) und *Punkt-Punkt* steht für das unmittelbar übergeordnete Verzeichnis (Vorgänger–Directory) des aktuellen Verzeichnisses.

1.11 Pfadnamen

Jeder Benutzer hat ein vom System-Administrator festgelegtes Heimatverzeichnis (*Home–Directory*), das vom Namen her dem *Username* gleicht. Nach jedem Anmelden beim System wird das Heimat–Verzeichnis als aktuelles Arbeitsverzeichnis (*Working-Directory*) gesetzt.

Dateien im aktuellen Verzeichnis (Working–Directory) sind durch Angabe ihres Namens direkt zugreifbar. Dateien, die sich nicht im aktuellen Verzeichnis befinden, müssen über einen Pfadnamen angesprochen werden. Ein Pfadname beschreibt einen Pfad durch das Dateisystem, der zu der gewünschten Datei führt. Für eine Datei ist dadurch der Weg von der root–Directory über die UNIX–Dateisystem–Struktur immer **eindeutig** identifizierbar (siehe Abbildung 2: von root —> aufg1). Pfadkomponenten werden durch einen "/" (Schrägstrich) getrennt. Beim Ansprechen einer Datei kann entweder der absolute oder der relative Pfadname verwendet werden.

Absolute Pfadnamen beginnen mit dem Zeichen "/" für die root–Directory. Dahinter werden immer durch "/" voneinander getrennt alle Verzeichnisnamen bis zum Ziel angegeben, z.B.:

`/home/cip1/benutzer-A/kurs/aufg1` # **aufg1** = Datei in Verzeichnis **kurs**

Relative Pfadnamen gehen vom aktuellen Verzeichnis (*Working-Directory*) aus. Angenommen, **/home/cip1/benutzer–A** sei das aktuelle Verzeichnis. Um über einen relativen Pfad in das Verzeichnis **aufg1** zu kommen, muß folgender Pfadname angegeben werden:

`kurs/aufg1`

In den Pfadkomponenten eines Pfadnamens können auch Ersatzzeichnen verwendet werden (siehe Abschnitt 4.5).

1.12 Verzeichnisse (*Directories*)

Es gibt keine Beschränkung für die Zahl der Unterverzeichnisse (*Subdirectories*). Deswegen bietet es sich an, Unterverzeichnisse intensiv als Mittel zur Strukturierung des Heimatverzeichnisses zu verwenden. So ist es z.B. sinnvoll, Unterverzeichnisse für verschiedene Forschungsprojekte, Programmieraufgaben, Textverarbeitung oder Verwaltungsaufgaben anzulegen.

Folgende Kommandos werden für den Umgang mit Verzeichnissen benötigt:

Kommando	Erklärung	Hinweis
`mkdir dirname`	lege ein neues Unterverzeichnis `'dirname'` an	(siehe Abschnitt 6.1)
`rmdir dirname`	entferne das (leere!) Verzeichnis `'dirname'`	(siehe Abschnitt 6.2)
`cd dirname`	wechsele in das Verzeichnis `'dirname'`	(siehe Abschnitt 5.3)
`cd`	wechsele in das Heimatverzeichnis (Home–Directory)	(siehe Abschnitt 5.3)
`ls`	liste die Dateien des aktuellen Verzeichnisses auf	(siehe Abschnitt 5.4)
`pwd`	zeige den Namen des aktuellen Verzeichnisses an	(siehe Abschnitt 5.2)

1.13 UNIX–Kommandos

Kommandos werden entweder interaktiv eingegeben oder sie sind Teil einer Kommandoprozedur. Ein Kommando wird immer in (wenigstens) einer Kommandozeile formuliert. Erstreckt sich ein Kommando über mehrere Zeilen, sind die jeweiligen Zeilenenden mit einem "\" (Backslash) zu entwerten. Innerhalb eines Kommandos dienen Leer– oder Tabulatorzeichen als Trenner. Kommandos werden in einfache und in zusammengesetzte eingeteilt.

1.13.1 Einfache Kommandos

Ein einfaches Kommando besteht aus einem Kommandonamen, dem eventuell Optionen folgen, an die sich eventuell Objektbezeichnungen anschließen. Die Optionen und die Objektbezeichnungen sind die *Parameter* oder *Argumente* des Kommandos.

Kommandoname *[Optionen]* *[Objektbezeichnungen]* # Syntax

Optionen beeinflussen die Wirkungsweise eines Kommandos. Es ist üblich, aber keine Norm, daß Optionen mit einem Bindestrich beginnen [1]. Mehrere Optionen können hinter einem einzigen Minuszeichen und ohne Trenner zusammengefaßt werden [2] (Das ist üblich, allerdings ebenfalls keine Norm). Die Argumente und die Trenner zerlegen eine Kommandozeile in sogenannte *Token* und können gehäuft auftreten.

Da Optionen bei den einzelnen Kommandos nicht immer die gleiche Bedeutung haben, sollte man sich vor ihrer Verwendung über die *Manual Pages* informieren.

Objektbezeichnungen sind die Argumente, auf die ein Kommando wirkt. In den meisten Fällen werden das wohl Dateien sein.

sort –r adressen # Beispiel: Einfaches Kommando mit drei Token
1) sort Kommando
2) –r Option
3) adressen Objekt

date	# nur Kommando, ~~keine Option~~, ~~kein Objekt~~
ls –al	# Kommando, Zwei Optionen: a und l, ~~kein Objekt~~
cat adressen	# Kommando, ~~keine Option~~, ein Objekt
sort –r adressen	# Kommando, eine Option, ein Objekt

1.13.2 Zusammengesetzte Kommandos

Zusammengesetzte Kommandos sind einfache Kommandos, die durch Sonderzeichen mit Dateien oder anderen Kommandos verbunden sind.

cat a1.c | wc > a2.c # Beispiel: Zusammengesetztes Kommando

Die beiden Sonderzeichen "|" und ">" beeinflussen den Weg, den die Ausgabe des Kommandos **cat** sowie die Ein– und Ausgabe des Kommandos von **wc** nimmt. Die Sonderzeichen werden in Abschnitt 3.3 und 3.2 behandelt.

[1] Die AT&T–Entwicklung (System–V) verlangt Bindestriche. Die Berkley–Entwicklung benötigt keine Bindestriche. Die an den Rechnern des Instituts für Angewandte und Numerische Mathematik eingesetzten Systeme UNIX und OSF der Firma DEC lassen beide Möglichkeiten bei bestimmten Optionen zu.

[2] Leider ist das nicht einheitlich unter UNIX, z.B. Compilerkommandos benötigen **immer** einen eigenen Bindestrich vor jeder Option.

1.13.3 Kommando–Trenner

Unter einem Kommando–Trenner versteht man das Zeichen, das zwei nebeneinanderstehende Kommandos voneinander trennt. Folgende Zeichen erfüllen diesen Zweck:

NEWLINE	LF–Zeichen (ASCII, dezimal 010), ausgelöst durch die RETURN–Taste
;	(Semikolon) Kommandos, die durch NEWLINE oder Semikolon voneinander getrennt sind, werden hintereinander in der angegebenen Reihenfolge abgearbeitet.
|	(Pipeoperator) Kommandos, die durch ein Pipesymbol voneinander getrennt sind, bilden ein einziges zusammengesetztes Kommando.
&	(Hintergrundoperator) Das Kommando (einfach oder zusammengesetzt) vor dem Hintergrundoperator wird im Hintergrund gestartet.

cat < b.txt | sort; wc < a.txt # Beispiel: Zwei Kommandos in einer Kommandozeile

1.13.4 Kommando–Gruppen

Mit runden Klammern können Kommandogruppierungen vorgenommen werden.

Beispiele:

a; b &	**a** wird im Vordergrund abgearbeitet, **b** im Hintergrund.
(a; b) &	**a** und **b** werden (sequentiell) im Hintergrund abgearbeitet.
(a; b) & (c; d) &	**(a; b)** werden parallel zu **(c; d)** im Hintergrund abgearbeitet.
(ls –l ; cat *) > sammeln	Produziert eine gemeinsame Standard–Ausgabe. Ohne Zusammenfassung würde die Kommandozeile folgendermaßen aussehen: **ls –l > sammeln ; cat * >> sammeln**

2 Benutzung der Maus

Die Maus ist ein Zeigerinstrument und für die Benutzung von fensterorientierten Benutzeroberflächen unerläßlich. Sie ermöglicht eine *direkte Manipulation*, d.h. es können auf dem Bildschirm zur Verfügung gestellte Objekte aktiv berührt (Schalter, Kommandos, Menü–Attribute) oder bewegt (Fenster, Ikonen) werden.

Die Maus besitzt drei Druckknöpfe, mit denen folgende Aktionen ausgeführt werden können:

Aktivität	**Ausführung**
Positionieren (*Point*)	Positionieren des Mauszeigers für die nächste Aktion durch Bewegen der Maus auf der Mausunterlage.
Klick (*Click*)	Druckknopf drücken u. wieder loslassen, **ohne die Maus zu bewegen**.
Drücken (*Press*)	Druckknopf drücken und niederhalten, **ohne die Maus zu bewegen**.
Doppelklick (*Double Click*)	Druckknopf zweimal sehr schnell hintereinander drücken, **ohne die Maus zu bewegen**.
Ziehen (*Drag*)	Druckknopf drücken, niederhalten, und den Maus–Zeiger durch Bewegen auf ein neues Objekt positionieren.
Shift Klick (*Shift Click*)	Positionieren eines Objekts, drücken und halten der Shift–Taste, Druckknopf MB1 drücken.

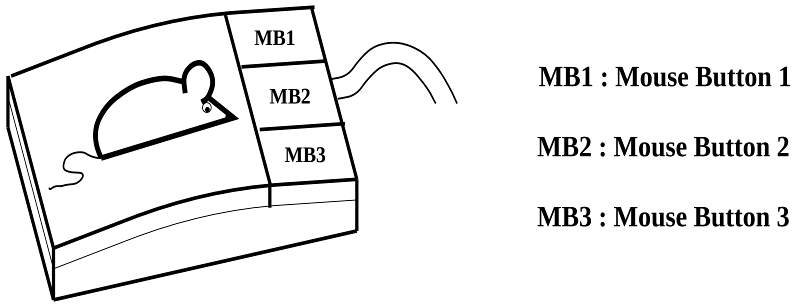

Abbildung 3: Eine Maus mit der Anordnung der Maus–Druckknöpfe (Mouse–Buttons) für Rechtshänder (Standard–Einstellung). Die Maus läßt sich auch für Linkshänder programmieren: Option–Menu im Session Manager Fenster, bei Pointer den Schalter 'Left Handed' wählen.

Durch Bewegen der Maus (rollen der Kugel auf dem *Mousepad*) bewegt sich auch der Mauszeiger auf dem Bildschirm. Er kann innerhalb der Grenzen des Root–Fensters jede Position einnehmen. Für unterschiedliche Manipulationen (Ausführung von Funktionen) nimmt der Mauszeiger unterschiedliche Zeigerformen an.

Je nach Voreinstellung des Window–Managers kann ein Fenster durch Positionieren des Mauszeigers oder durch zusätzliches Anklicken mit MB1 aktiviert werden. Nur ein aktives Fenster kann Eingaben entgegennehmen. Ein aktives Fenster wird durch eine andersfarbene Titelleiste und Laufmarke angezeigt. Wenn in einem Fenster *'irgend etwas'* nicht arbeitet, ist erst der Mauszeiger zu überprüfen, ob er sich innerhalb der Fenstersgrenzen dieses Fensters befindet.

3 Emacs: Text-Editor

Ein wesentlicher Teil eigener Arbeit besteht normalerweise darin, selbst mit Hilfe eines Editors einen Text zu speichern, bzw. den Inhalt einer bestehenden Datei zu verändern. Es stehen unter UNIX verschiedene Editoren zur Verfügung.

Ein geeigneter Editor heißt Emacs. Emacs ist mehr als nur ein Editor. Er ist eine Programmierumgebung, kann zum Lesen von E–Mails verwendet werden und hat ein komfortables Informationssystem. Bei regelmäßiger Benutzung wird man viele Möglichkeiten zur Verwendung von Emacs entdecken. Der einfache Umgang wird hier in kleiner Auswahl stichwortartig beschrieben.

Kommandos an Emacs können auf verschiedene Arten eingegeben werden:

- Durch Auswählen mit der Maus (Klicken der linken Maustaste) in der schwarz unterlegten Menüleiste am Kopf des Emacs–Fensters. Jede Operation über die Menüleiste ist an ein Emacs–Kommando gebunden. Die Menüleiste erklärt sich selbst.
- Durch Eingabe von Spezialzeichen (*Keys*). Es handelt sich dabei um Tasteneingaben, die in Kombination mit der **CTRL**–Taste oder der **META**–Taste gedrückt werden. Da viele Terminals keine **META**–Taste besitzen, wird ein Meta–Zeichen durch die **ESC**–Taste simuliert. **Hinweis:** Nur oft benutzte (gebräuchliche) Kommandos im Emacs besitzen Tastenbindungen.
- Durch Eingabe eines Kommandonamens im Minipuffer in der Form `M-x` *[Kommando]* (siehe Abschnitt 3.13).

Es bleibt dem Benutzer überlassen, für welche Form der Kommandoeingabe oder Kombination er sich entscheidet. In der weiteren Beschreibung des Emacs bedeuten:

C-<z>	Festhalten der CTRL–Taste und anschließende Eingabe eines Zeichens.
M-<z>	Kurzes Drücken der ESC–Taste und anschließende Eingabe eines Zeichens.

Mit **SPC** (*space*) wird die Leertaste bezeichnet.

Emacs stellt verschiedene Modi (*major modes*) zur Verfügung, die das Editieren sehr erleichtern. Beim Aufruf einer Datei zum Editieren wird auf die Dateinamensendung wie z.B. `.c`, `.tex`, `.f`, `.p` geachtet und automatisch der passende Modus gestartet. In den einzelnen Modi werden zum Teil selbstständig Überprüfungen (z.B. geschweifte Klammerpaare) und zum Teil modusspezifische Ausführungen über Kommandos (z.B. strukturiertes Einrücken des Codes) durchgeführt. Informationen über den aktuellen Modus erhält man mit dem Kommando `C-h m`. Der von Emacs gestartete Modus wird in der Modus–Zeile unter dem gestarteten Fenster angezeigt (z.B. LaTeX ADVANCE, C ADVANCE, Fortran ADVANCE).

Viele Kommandos sind für die einzelnen Modi übergreifend, d.h. sie führen in jedem Modus bei gleicher Tastenbindung (*Key*) den selben Befehl aus. Es gibt aber auch Kommandos, die bei gleichem Key unterschiedliche Befehle innerhalb der verschiedenen Modi ausführen, wie die nachfolgende Beispieltabelle zeigt.

Tastenbindung	Modus	Ausführung
`C-a`	`alle Modi außer 2`	gehe an den Anfang einer Zeile.
`C-a`	`Calendar mode`	gehe an den Anfang einer Woche.
`C-a`	`Shell mode`	gehe an den Anfang einer Zeile, aber nach dem Prompt (sofern die Zeile einen Prompt aufweist).

3.1 Starten des Emacs–Editors

2. `emacs dateiname &` ein Emacs–Prozeß wird im Hintergrund angelegt.

   Das Zeichen '&' im Aufruf bewirkt, daß der Editor als unabhängiger Prozeß im Hintergrund gestartet wird (siehe Abschnitt 1.5).

3.2 Verlassen des Emacs–Editors

`C-x C-c` oder Emacs–Menüleiste: File–Menu –> Kommando: *Exit Emacs*

3.3 Fehlerbehandlung

`C-g`	Abbrechen eines teilweise getippten oder ausgeführten Kommandos.
`C-l`	Reorganisieren (*recenter*) eines Fensters, wobei die Zeile mit dem Cursor Mittelpunktzeile des Fenstersneuaufbaus wird.
`C-x u`	Rückgängigmachen (*undo*) des letzten Kommandos. Bei Mehrfacheingabe werden die Kommandos in umgekehrter Eingabereihenfolge rückgängig gemacht.
`M-~`	(Tilde) Löschen der Änderungsmarke (*modificationflag*), wenn Änderungen nicht gerettet werden sollen. Als sichtbares Zeichen werden in der Modus-Zeile die Sternchen '**' in `'--'` umgewandelt.
`M-x recover-file`	Wiederherstellen einer durch System-crash verlorenen Datei. Als Eingabe wird der durch *auto-save* gekennzeichnete Dateiname verlangt (`#dateiname.typ#`). Emacs legt selbstständig Sicherheitskopien von Dateien an, die sich in einem Veränderungszustand befinden. Die Standardeinstellung der Variablen *auto–save–interval* beträgt 300 eingegebene Zeichen.

3.4 Laden von Dateien

`C-x C-f`	Laden einer Datei (*find-file*). Der Name wird nachfolgend im Minipuffer erfragt.
`C-x C-s`	Abspeichern der aktuellen Datei (*save-buffer*).
`C-x C-w`	Schreibe den aktuellen Puffer (*write-file*) in eine anzugebende Datei.
`C-x C-v`	Ersetze die aktuelle Datei mit einer anderen (*find-alternatefile*).
`C-x i`	Einfügen einer anderen Datei **vor** dem Cursor (*insert-file*).
`C-x d`	Starte den Verzeichnis–Editor (*dired*). Folgende Kommandos unter Eingabe des 1. Zeichens sind im Verzeichnis–Editor möglich (unvollständiger Auszug):

~	(Tilde) Markiert alle Dateien mit einer Löschmarke, bei denen Datei-Namensteile in Tilden eingeschlossen sind (*Dateiname.Typ.~1~*).
#	Markiert alle Dateien mit einer Löschmarke, die als *auto-save*–Dateien gekenzeichnet sind (#*Dateiname.Typ*#).
d`-elete`	Markiert eine gewählte Datei zum Löschen.
u`-ndelete`	Entfernt eine Lösch-Markierung.
x`-ecute`	Ausführen des Löschvorgangs für die markierten Dateien.
f`-ind`	Ausgewählte Datei vorspielen.
R`-ename`	Ausgewählte Datei umbenennen.
c`-opy`	Ausgewählte Datei kopieren.
v`-iew`	Betrachten einer Datei (*readonly*, kein Editieren möglich).
Z	Kommando zum Komprimieren/Entkomprimieren (zip).
?	Informiert über Kommandos des Verzeichnis–Editors.

3.5 Positionieren in der Datei

Zum Positionieren können die vier Pfeiltasten verwendet werden. Schneller und frei versetzen läßt sich der Cursor über die Maus mit einem Klick von MB1 (linke Maustaste).

Weitere Positionierungsfunktionen für:

Cursor:

vorwärts	rückwärts	
`C-f`, →	`C-b`, ←	einzelnes Zeichen (→, ← = Pfeiltasten).
`M-f`, `kp-1`	`M-b`	einzelnes Wort (kp-1 = Taste 1 im Ziffernblock).
`C-n`, ↓	`C-p`, ↑	1 Zeile weiter (↓, ↑ = Pfeiltasten).
`C-e`, `kp-2`	`C-a`	Zeilenende oder -anfang (kp-2 = Taste 2 im Ziffernblock).
`M-e`	`M-a`	Satzende oder -anfang.
`M-}`	`M-{`	Absatzende oder -anfang.
`M->`	`M-<`	Sprung zum Pufferende oder -anfang der Datei.

Bildschirm:

`C-v`, `Next`–Taste	Blättern um eine Bildschirmseite vorwärts
`M-v`, `Prev`–Taste	Blättern um eine Bildschirmseite rückwärts.

3.6 Löschen und wieder Einsetzen

1) vorwärts	2) rückwärts				
`C-d`	`DEL`	Zeichen löschen	1) auf	2) vor	der Cursorposition.
`M-d`	`M-DEL`	Wort löschen	1) auf	2) vor	der Cursorposition.
`C-k`	`M-0 C-k`	Zeile löschen	1) auf	2) vor	der Cursorposition.
`M-k`	`C-x DEL`	Satz löschen	1) auf	2) vor	der Cursorposition.

`Komma`–Taste	Zeichen löschen auf der Cursoposition (**nur im Ziffernblock).**
`Bindestrich`–Taste	Wort löschen auf der Cursoposition (**nur im Ziffernblock).**
`C-w`	Löschen eines markierten Abschnitts.
`M-z char`	Löschen bis zum nächsten Vorkommen des einzugebenden Zeichens.
`C-y`	Wiedergabe (*yank*) der zuletzt gelöschten Größe an der Cursorposition.
`M-y`	Ersetze letzte Wiedergabe mit vorausgehendem Löschen.
`M-SPC`	Löschen von Leerzeichen (Blanks, Tabs) ab Cursorposition gegen das Zeilenende. Ein Leerzeichen bleibt bestehen.
`C-x C-o`	Löschen aller aufeinanderfolgende Leerzeilen ab Cursorposition. Eine Leerzeile bleibt bestehen.

3.7 Markieren

`C-@ oder C-SPC`	Setzen einer Marke an der aktuellen Cursorposition
`C-x C-x`	Vertausche Punkt und Marke.
`M-@`	Markiere bis zum Ende des nächsten Wortes.
`M-h`	Markiere Absatz.

3.8 Suchen

Inkrementelles Suchen (*Incremental search*): Nach der Initialisierung wird die zu suchende Zeichenkette im Minipuffer erfragt. Es wird nach **jedem** eingegebenen Zeichen **sofort** auf die nächste **bis dahin** übereinstimmende Textstelle positioniert.

`C-s, PF3`	Suchen vorwärts (PF3 = Taste im Ziffernblock).
`C-r`	Suchen rückwärts.

Wiederholte Eingabe von `C-s` oder `C-r` setzt die Suche in der entsprechenden Richtung fort. Doppelklick von `C-s` oder `C-r` führt den durch Unterbrechung zuletzt eingegebenen Suchbegriff erneut aus.

`ESC`	Beende inkrementelle Suche.
`DEL`	Löscht das letzte Zeichen der Suchzeichenkette im Minipuffer.
`C-g`	Abbrechen der aktuellen Suche.

Nichtinkrementelles Suchen (*Nonincremental search*) Nach der Initialisierung wird die zu suchende Zeichenkette erfragt. Der Suchvorgang wird mit `<RETURN>` ausgeführt. Die Suchfunktion wird nur **einmal** ausgeführt!

`C-s RET` *Zeichenkette* `RET`	Suchen vorwärts.
`C-r RET` *Zeichenkette* `RET`	Suchen rückwärts.

Eine Wiederholung, ohne die Suchzeichenkette noch einmal eingeben zu müssen, ist durch folgende Kommandofolge möglich (nur wenn zwischen zwei Suchkommandos kein anderer Befehl eingegeben wurde: `C-x, ESC, ESC, RETURN`; oder über *minibuffer history list*).

`M-x occur RET` *Zeichenkette* `RET`	Auflisten aller Zeilen von Cursorposition bis Dateiende in einem '`*Occur*`'–Fenster, in denen der eingegebene Suchbegriff vorkommt.
`C-c C-c`	Springt auf die Zeile in Editier–Fenster, die mit dem Cursor im '`*Occur*`'–Fenster ausgewählt wurde.

3.9 Ersetzen

Es wird ein Vorgang eingeleitet, durch den wiederholt eine vorhandene Zeichenfolge (*oldstring*) durch eine neue (*newstring*) ersetzt werden kann. Das Ersetzen erfolgt nur **ab** der Cursorposition in Richtung Dateiende.

`M-%` *oldstring* `RETURN` *newstring* `RETURN`

Mit folgenden Eingaben kann der Ersetzungsvorgang gesteuert werden:

`y, SPC`	Ersetze dieses, gehe zum nächsten Vorkommen.
`,`	(Komma) Ersetze dieses, aber gehe nicht weiter.
`n, DEL`	Springe zum nächsten Vorkommen, ohne zu ersetzen.
`!`	Alle (weiteren) Vorkommen ohne Abfrage ersetzen.
`.`	(Punkt) Ersetze dieses Vorkommen, dann abbrechen der Ersetz–Funktion.
`^`	(Dach) Zurück zum letzten passenden Vorkommen.
`q, ESC`	Verlasse Ersetzen mit Nachfrage.

3.10 Vertauschen

`C-t`	Vertausche den Buchstaben vor dem Cursor mit dem Buchstaben der Cursorposition.
`M-t`	Vertausche das Wort vor oder auf der Cursorposition mit dem nachfolgenden Wort.
`C-x C-t`	Vertausche die Zeile, in der der Cursor steht, mit der vorhergehenden Zeile.

3.11 Änderung von Groß– und Kleinschreibung

`M-u`	Schreibe Wort ab Cusorposition groß.
`M-l`	Schreibe Wort ab Cusorposition klein.
`M-c`	Schreibe Zeichen auf Cusorposition groß.
`C-x C-u`	Schreibe den markierten Bereich in Kleinbuchstaben.
`C-x C-l`	Schreibe den markierten Bereich in Großbuchstaben.

3.12 Puffer und Fenster

Jeder Text, den Emacs bearbeitet, wird in einem internen Puffer gehalten. Dadurch ist es möglich, mit Emacs mehrere Dateien parallel zu bearbeiten, ohne daß Emacs mehrfach aufgerufen werden muß. Für die Bearbeitung mehrerer Puffer gleichzeitig läßt sich das Emacs–Fenster in mehrere Teilfenster aufteilen. Am unteren Ende jedes Fensters befindet sich die *Mode-Line*, in der der Name des dazugehörigen Puffers (normalerweise der `Dateiname`) und andere zusätzliche Informationen angezeigt werden.

Beim Benutzen mehrerer Emacs–Fenster beziehen sich die Eingaben im Minipuffer immer auf das Fenster, in dem sich der Cursor befindet (*Selected–Window*) und damit auf den Puffer, dessen Inhalt das Selected–Window zeigt (*Currend–Buffer*).

`C-x 2`	Splittet das aktive Fenster in zwei horizontale Fenster.
`C-x 5`	Splittet das aktive Fenster in zwei vertikale Fenster.
`C-x o`	(other) Wählt ein anderes Fenster als aktives Fenster aus.
`C-x 0`	(Null) Löscht das aktive Teilfenster.
`C-x 1`	Löscht alle anderen Teilfenster außer dem aktiven Fenster.
`Buffers`	Pufferkommandos: Emacs–Menüleiste: Buffers–Menu

Der Emacs verwendet nur **einen** gemeinsamen Löschpuffer. Dadurch ist es möglich, unabhängig davon, wieviele Teilfenster eröffnet sind, Textteile innerhalb der Puffer (Dateien) auszutauschen.

3.13 Der Minipuffer

Der Minipuffer (*Minibuffer*) bietet die Möglichkeit, die zu einem Emacs–Kommando benötigten Argumente einzugeben. Diese Argumente können Dateinamen, Puffernamen, Lisp–Funktionsnamen, Emacs–Kommandonamen und vieles andere mehr sein. Editieren, das Komplettieren von Kommandos und die sogenannte *minibuffer history list* sind weitere Vorzüge.

Folgende Tastenbelegungen sind im Minipuffer definiert:

`TAB`	Ergänze so weit wie möglich.
`SPC`	Ergänze bis zu einem Wort.
`RET`	Ergänze und führe aus.
`?`	Zeige mögliche Ergänzungen.
`C-g`	Abbruch des Kommandos.

Emacs–Kommandos können auch unter ihrem Namen im Minipuffer eingegeben werden. Mit der Tastenkombination `M-x` wird eine Kommandoeingabe aktiviert. Das eingegebene Kommando wird mit der `RETURN`–Taste gestartet.

M-x	*Kommando* `RETURN`	# Syntax
M-x	**list–command–history** `RETURN`	# Beispiel

Die im Minipuffer eingegebenen Kommandos werden in der *minibuffer history list* gespeichert. Kommandos aus dieser Liste können später wieder abgerufen werden.

`C-x ESC ESC    spielt das letzte eingegebene Kommando vor`

Beim Aufruf der *list-command-history* sind folgende Tastenbelegungen definiert:

- `M-p` Vorhergehendes Minipuffer–Kommando.
- `M-n` nächstes Minipuffer–Kommando.

Jedes Kommando kann über den Minipuffer eingegeben werden. Oft benutzte (gebräuchliche) Kommandos besitzen Tastenbindungen und sind auch über die Emacs–Menüleiste per Maus zu benutzen. Die folgende Tabelle zeigt den Unterschied zwischen Tastenbindung und ausführlichem Kommandonamen.

Tastenbindung	Eingabe über Minipuffer
`C-x C-c`	`M-x save-buffers-kill-emacs`
`C-x d`	`M-x dired`
`C-k`	`M-x kill-line`

Bei der Eingabe über das Kommando `M-x` kennt Emacs die Möglichkeit, abgekürzt eingegebene Kommandonamen zu vervollständigen (*Completion*). Wenn eine Abkürzung zum Vervollständigen nicht eindeutig möglich ist, wird automatisch ein weiteres Fenster eröffnet, in dem alle zum bisherigen Textmuster vervollständigten und passenden Kommandonamen angezeigt werden.

Vervollständigen für inkrementelles Suchen rückwärts (*isearch–backward*) der Zeichenkette `'heute'`

`M-x, 'ise', <Tab>, 'b', <Ret>, 'heute'` # Beispiel

Erklärung:

Aktivieren des Minipuffers:	`M-x`
Eingabe des abgekürzten Kommandonamens:	`'ise'` (*isearch*)
Vervollständigen des Kommandonamens:	`<TAB>-Taste`
Eingabe des abgekürzten Kommandonamens:	`'b'` (*backward*)
Vervollständigen und Ausführen der Kommando:	`<RET>-Taste`
Eingabe der zu suchenden Zeichenkette:	`'heute'`

3.14 Hilfe

Emacs bietet umfangreiche Hilfestellungen an, um beispielweise Informationen für die verschiedenen Kommandos, Modes oder Tastaturbelegungen zu erklären. Einsicht gibt das Help–Menu in der Emacs–Menüleiste.

4 Arbeiten mit der Shell

Rechner haben keine naturgegebene Fähigkeit, die eingegebenen Kommandos zu entschlüsseln. Die meisten Betriebssysteme stellen einen Kommandointerpreter bereit, der diese Funktion durchführt. Der Kommandointerpreter von UNIX heißt *Shell*. Die Shell umfaßt vielfältige Möglichkeiten, sehr leistungsfähige Kommandos zu spezifizieren.

4.1 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 Standard–Ausgabekanal. Beides sind lediglich vordefinierte Dateien. Der Eingabekanal (Standard–Eingabe) wird i.d.R. mit der Eingabe über Tastatur identifiziert, der Ausgabekanal (Standard–Ausgabe) mit dem Bildschirm.

In einer Umgebung mit verschiedenen logischen Terminals, wie etwa einer Fensterumgebung, wo jedes Fenster ein eigenes logisches Terminal darstellt, sind Standard–Eingabe und –Ausgabe für jedes Fenster extra definiert. Standard–Eingabe und –Ausgabe können in verschiedene Dateien umgelenkt werden.

Für Fehlermeldungen existiert eine weitere interne Datei. Diese Datei benutzt ebenfalls den Bildschirm als Standard–Ausgabe.

4.2 Umlenken von Standarddateien

Die Standard-Eingabe und die Standard-Ausgabe sind normalerweise mit dem Terminal verbunden (Primärzuweisung). Da aber die Standard-Eingabe und die Standard-Ausgabe von der Shell zugeordnet werden, ist es möglich, sie von der Shell umlenken zu lassen. Eine Eingabe kann so also aus einer Datei kommen und/oder die Ausgabe kann in einer Datei abgelegt werden.

Shell-Umlenkzeichen

- `>` Lenkt die **Ausgabe** in eine Datei um. Besteht die Datei schon, wird ihr Inhalt überschrieben. Existiert unter dem gewählten Dateinamen noch keine Datei, legt die Shell eine neue Datei an.
- `>>` Lenkt die **Ausgabe** in eine Datei um und hängt die **Ausgabe** am Ende des bestehenden Inhalts an. Existiert noch keine Datei unter dem gewählten Dateinamen, wird eine neue Datei angelegt.
- `<` Lenkt die **Eingabe** um, d.h. übernimmt die Eingabe aus der angegebenen Datei.

Beispiele:

`cat kap1 kap2 > buch`	Verknüpft die beiden Dateien `'kap1'` und `'kap2'` und überträgt sie durch Umlenkung in die Datei `'buch'`.
`cat index >> buch`	Durch Umlenkung wird der Inhalt der Datei `'index'` an das Ende der Datei `'buch'` angehängt.
`sort < unsort`	Durch Umlenkung erhält das Programm **sort** die Eingabedaten aus der Datei `'unsort'`. Ausgabe = Standard–Ausgabe.
`sort < unsort > sortiert`	Durch Umlenkung bezieht das Programm **sort** seine Eingabedaten aus der Datei `'unsort'` und schreibt den sortierten Inhalt über Umlenkung in die Datei `'sortiert'`.

4.3 pipe (Verknüpfen von Kommandos)

Unter einer pipe versteht man ein Objekt, das nach dem FIFO–Prinzip arbeitet. FIFO steht für *First In First Out* und meint, daß aus einer solchen Datei immer nur in genau der Reihenfolge gelesen werden kann, in der vorher hineingeschrieben worden ist, wobei das Gelesene aus der Datei entfernt wird. Es entsteht anschaulich das Verhalten einer Röhre (Pipeline), durch die Bytes hindurchfließen. Diese Veranschaulichung hat dem Verfahren den Namen gegeben.
Pipes ermöglichen, daß die Standard–Ausgabe eines Kommandos als Standard–Eingabe eines folgenden Kommandos verwendet wird. Eine Pipe–Kommandofolge wird durch das Symbol "|" (Pipe–Verbinder) verbunden.

cat unsortiert.dat | sort # Beispiel: pipe

Die Standard–Ausgabe des Kommandos **cat** wird zur Standard–Eingabe des Kommandos **sort**. Als Ergebnis dieses verknüpften Kommandos erfolgt eine sortierte Ausgabe der Datei `'unsortiert.dat'` auf dem Bildschirm.

4.4 Filter

Man nennt Programme, die zwischen zwei Pipesymbolen stehen können *Filter*. Sie lesen ihre Information von der Standard–Eingabe und schreiben das Ergebnis in die Standard–Ausgabe.

cat a.txt | grep "UNIX" | wc > a.erg # Beispiel 1: Filter

cat a.txt | sort | lpr –Peg # Beispiel 2: Filter

Beispiel 1: Das **cat**–Kommando liest die Datei `'a.txt'` und schreibt seine Ausgabe über eine Pipe zum **grep**–Kommando. Dieses filtert alle Zeilen mit der Zeichenfolge "`UNIX`" heraus und schreibt sie über eine Pipe zum Word–Count–Kommando **wc**, das die Zeilen, Wörter und Zeichen seiner Eingabe zählt und sein Ergebnis in die Datei `'a.erg'` schreibt.

Beispiel 2: Hier wird das **sort**–Kommando als Filter benutzt.

4.5 Automatische Ergänzung von Dateinamen (*Wildcards*)

Manchmal sollen mehrere Dateien angesprochen werden, aber man möchte nicht alle Dateinamen einzeln eingeben. Oder es sollen lange Dateinamen abgekürzt eingegeben werden. Dafür kennt UNIX Sonderzeichen (*Wildcards*), die als Dateinamen–Suchmuster für einzelne Zeichen oder ganze Gruppen eingesetzt werden. Das Verfahren heißt Namensexpandierung (*Filename Expansion*).

Die Shell vergleicht die Suchmuster mit dem Namen sichtbarer, also nicht mit Punkt beginnender Dateien des jeweiligen aktuellen Dateiverzeichnisses und ersetzt das angegebene Suchmuster durch die Dateinamen, für die der Vergleich erfolgreich ist.

Anmerkung: Es gibt **kein** Sonderzeichen, das zu einem führenden Punkt in einem Dateinamen paßt. Bei Punkt–Dateien ist der führende Punkt explizit zu schreiben.

`?`	Paßt zu genau einem Zeichen in einem Dateinamen an genau der Position, an der es im Suchmuster steht.
`*`	Paßt zu 0 oder mehr Zeichen eines Dateinamens an genau der Position, an der es im Suchmuster steht.
`[Zeichenkette]`	Paßt zu genau einem Zeichen aus der Klammer an genau der Position, an der im Suchmuster die Klammer steht.
`[Zeichen1--Zeichen2]`	Wie die Bedeutung von `[Zeichenkette]`, nur daß in der Klammer Zeichenintervalle (mit Bindestrich und in ASCII–Reihenfolge) stehen dürfen.
`!`	"!" als erstes Zeichen in der Klammer wirkt als Negation: Alle Zeichen außer denen in der Klammer werden gewertet.

Beispiele:

`a?b`	Steht für alle Dateien, deren Namen aus drei Zeichen bestehen, die mit einem "**a**" beginnen und mit einem "**b**" enden.
`a*b`	Steht für alle Dateien, deren Namen mit dem Zeichen "**a**" beginnen und mit dem Zeichen "**b**" enden. Für die im Dateinamen angegebene *wildcard* "*****" folgen Zeichen, die in Art und Anzahl beliebig sein können.
`*`	'*' allein steht für alle Dateien im aktuellen Verzeichnis.
`test*`	Steht für alle Dateien mit dem Namensbeginn **'test'**, deren Restname aus einer beliebigen Zeichenfolge bestehen kann.
`rm *`	Löscht alle Dateien in der aktuellen Directory, die nicht mit einem Punkt beginnen (**Löscht unwiderruflich und ohne Warnung!**).
`ls -l [de]*`	Gibt eine Liste der Dateiennamen aus, deren Anfangszeichen mit einem "**d**" oder "**e**" beginnen.
`rm test/[0-5]*.c`	Löscht alle Dateien in dem Unter-Verzeichnis **'test'**, deren Namen mit einer Ziffer zwischen "**0**" und "**5**" beginnen, im Restnamen beliebig viele Zeichen enthalten und deren Deiteinamenergänzungen mit **'.c'** enden.
`rm a.[!0-9]`	Löscht alle Datein mit einer einzeichigen Dateinamensergänzung, die keine Ziffer ist.

4.6 Aliase

Häufig gebrauchte Shell–Kommandos will man gerne abkürzen. Zu diesem Zweck können *Aliase* in der Datei '.kshrc' im Heimatverzeichnis des Benutzers definiert werden.

Beispiele:

`alias` **ll** `"ls -l"`	Das Kommando **ll** erzeugt eine lange Liste des aktuellen Verzeichnisses.
`alias` **rmi** `"rm -i *"`	Das Kommando **rm** verlangt über die eingebaute Option `'-i'` eine Bestätigung zum Löschen jeder Datei.
`alias` **c-dir** `"cd /home/cip1/name/name_c;pwd"`	Das Kommando **c-dir** wechselt vom Heimatverzeichnis 'name' des Benutzers in das Unterverzeichnis für C–Programme 'name_c' und gibt mit **pwd** als Kontrolle den Namen des jetzt neuen aktuellen Verzeichnisses aus.

5 Hilfsprogramme unter UNIX

5.1 Hilfestellungen

5.1.1 man (Informationen über das UNIX–Handbuch)

Eine Beschreibung der Shell–Kommandos und beispielsweise von Systemaufrufen steht dem Benutzer standardmäßig *On–Line* zur Verfügung. Damit ist gemeint, daß man unter anderem auf das UNIX–Handbuch (Manual Pages) per Dienstprogramm zugreifen kann.

man *Suchbegriff* # Syntax

gibt die Information über den angegebenen Suchbegriff aus, wobei automatisch das Kommando **more** (siehe Abschnitt 7.3) durchlaufen wird.

Beispiele:

`man man`	Gibt seitenweise Informationen über das **man**–Kommando aus.
`man ps`	Gibt seitenweise Informationen über das **ps**–Kommando aus.
`man mtools`	Gibt Befehle mit Kurzbeschreibungen zum Handling von DOS–Dateien aus.

5.1.2 whatis (Kurzbeschreibung von UNIX–Kommandos)

whatis *Kommando* # Syntax

liefert eine Kurzbeschreibung von dem eingegebenen Kommandonamen.

5.1.3 apropos (Suchen nach Informationen)

Um festzustellen, in welchen Handbuch–Einträgen eine Information über ein bestimmtes Stichwort bereitliegt, kann das Kommando:

apropos *Stichwort* # Syntax

benutzt werden. Es werden alle Handbuch–Einträge angezeigt, deren Titel *Stichwort* enthält.

5.2 pwd (das aktuelle Verzeichnis (*Working Directory*))

Informationen über das aktuelle Verzeichnis mit seinem vollen Pfadnamen liefert das Kommando **pwd** (**p***rint* **w***orking* **d***irectory*).

pwd # Syntax

5.3 cd (Wechseln in ein neues aktuelles Verzeichnis)

Ein Verzeichnis (Directory) läßt sich einfach wechseln. Mit dem Kommando **cd** (**c***hange* **d***irectory*) kann man sich anschaulich in der Verzeichnis–Struktur bewegen.

cd *[directory]* # Syntax

Folgende Ersatzzeichen können innerhalb von Pfadnamen verwendet werden:

.	(Punkt) ist das Ersatzzeichen der aktuellen Directory (Working Directory).
..	steht für ein Verzeichnis im Dateibaum unmittelbar über dem aktuellen Verzeichnis.
~	steht für das Heimatverzeichnis (Home–Directory).
~*Username*	steht für das Heimatverzeichnis (Home–Directory) des angegebenen Benutzers.

Beispiele:

`cd`	(Ohne Argumentenangabe!) Kehrt in das Heimatverzeichnis zurück.
`cd ~`	Kehrt in das Heimatverzeichnis zurück.
`cd otto/aufgaben`	Wechselt vom aktuellen Verzeichnis über das Unterverzeichnis `'otto'` in das Unterverzeichnis `'aufgaben'`.
`cd ../name_c`	Wechselt vom aktuellen Verzeichnis in das unmittelbar darüberliegende Verzeichnis und von dort in das Unterverzeichnis `'name_c'`.

5.4 ls (das Auflisten von Dateien)

Mit dem **ls**-Kommando werden Inhalte von Dateiverzeichnissen ausgegeben. Über Optionen kann festgelegt werden, welche zusätzlichen Informationen zu den im Verzeichnis enthaltenen Dateien ausgegeben werden sollen.

Optionen (unvollständiger Auszug)

`-l`	Erzeugt eine langformatige Auflistung.
`-t`	Erzeugt eine sortierte Liste entsprechend dem Veränderungsdatum.
`-a`	Listet **alle** Dateien im angegebenen Verzeichnis auf (auch Punktdateien).
`-g`	Erzeugt eine Auflistung mit Gruppenzugehörigkeit (nur sinnvoll mit **-l** Option!).
`-r`	Dreht die Reihenfolge der Ausgabe um.

Beispiele:

`ls`	Gibt eine Liste des aktuellen Verzeichnisses aus (Kurzform, ohne Punktdateien).
`ls *.c`	Listet alle Dateien des aktuellen Verzeichnisses auf, deren Dateinamenergänzungen `'.c'` lauten (Kurzform, ohne Punktdateien).
`ls -t /bin`	Listet die Dateien des Verzeichnisses `'/bin'` sortiert nach dem neuesten Veränderungsdatum auf (Kurzform, ohne Punktdateien).
`ls -al /etc`	Listet **alle** Dateien des Verzeichnisses `'/etc'` auf (langformatig, mit Punktdateien).

5.5 file (Die Dateiattribute bestimmen)

Mit dem Kommando

file *Dateiname* # Syntax

wird der Typ der angegebenen Dateien bestimmt. Für Directory- und Spezialdateien ist das **file**–Kommando exakt. Bei anderen Dateitypen kann das **file**–Kommando den Dateityp nur "vermuten".

file * # Beispiel: file, * = alle Dateien

Ergebnis:

`/usr/users/meier/mail:`	directory.
`liste-1-2:`	ASCII text.
`unix-und-c:`	English text.
`liste-alle:`	empty (leere Datei).
`texpreview.dvi:`	data.
`wochentag.c:`	c program text.

5.6 who (Auflisten aktiver Benutzer)

who # Syntax

listet alle am selben Rechner angemeldeten Benutzer auf. Das **who**–Kommando ordnet seine Ausgabezeilen in der Reihenfolge der Terminalmummern.

Beispiele:

`who`	Listet alle beim Rechner angemeldeten Benutzer auf.
`who am i, whoami`	Gibt den eigenen login-Namen aus. (Beide Formen werden akzeptiert)

5.7 ps (Prozesse anzeigen)

Informationen über den Status aktiver Prozesse liefert das **ps**–Kommando. Die Ausgabe von **ps** erfolgt in mehreren Spalten. Einige interessante sind:

`USER`	Besitzer des Prozesses.
`PID`	Prozeß–Identifikations–Nummer.
`%CPU`	Prozentsatz der CPU–Zeit, die der Prozeß gerade verbraucht hat.
`%MEM`	Prozentsatz des benutzten 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 (unvollständiger Auszug):

`-l`	Erzeugt eine langformatige Ausgabe.
`-a`	Druckt Informationen über alle Prozesse.
`-#`	Listet Informationen über den Prozeß mit der angegebenen Prozeßnummer auf.
`u`	Listet Benutzer orientierte Informationen über Prozesse auf. Unterscheidet sich zur Option 'l' in den Ausgabefeldern. (Erklärung für Optionen mit und ohne Bindestrich siehe Fußnote 2, Seite 9)
`w`	Benutzt 132 Spalten Ausgabe in einer Zeile statt 80 Spalten.
`x`	Listet Informationen über Prozesse aller Terminals auf.

Beispiele:

`ps`	Erzeugt eine Liste der eigenen Prozesse.
`ps -l`	Erzeugt eine langformatige Liste der Angaben über die eigenen Prozesse.
`ps axuw`	Erzeugt eine Liste aller eigenen Prozesse auf dem System.
`ps 4711`	Druckt Angaben über den Prozeß mit der Nummer `'4711'`.

5.8 kill (Abbrechen von Prozessen)

Manchmal ist es erforderlich, Prozesse vorzeitig zu beenden. Dabei hängt es davon ab, ob es sich um einen Vordergrund– oder Hintergrundprozeß handelt. Vordergrundprozesse sind mit dem Terminal verbunden. Sie werden durch das Auslösen des sogenannten Terminal–Interrupts vorzeitig beendet. Die Eingabe von `C-c` (CTRL–Taste und "c" gleichzeitig drücken) löst so einen Interrupt aus.

Hintergrungprozesse sind nicht mit dem Terminal verbunden. Das hat zur Folge, daß sie mit dem Terminal–Interrupt nicht mehr erreichbar sind. Um auch sie vorzeitig zu beenden, ist das **kill**–Kommando zu verwenden.

kill *[-signalnummer] processid* # Syntax

Zum Abbrechen eines Prozesses wird die Prozeßidentifikationsnummer (PID) benötigt (abzufragen über das **ps**–Kommando). Außerdem muß man "Eigentümer" des Prozesses oder Systemverwalter sein.

Beispiele:

`kill 4711`	Abbrechen des Prozesses mit der Prozeßidentifikationsnummer `'4711'`.
`kill -9 4712`	Abbrechen des Prozesses `'4712'`. **Hinweis:** Prozesse, die das normale Abbruchsignal abfangen oder ignorieren, können nur endgültig abgebrochen werden, wenn ihnen die Signalnummer "9" gesendet wird!

Das Kommando:

kill %*jobnummer* # Syntax

beendet gestoppte Jobs oder laufende Jobs im Hintergrund. *Jobnummer* ist eine Kennzahl, die bei Eingabe des Kommandos **jobs** in eckigen Klammern angezeigt wird.

5.9 yppasswd (Ändern des Sicherheitswortes (*Password*))

Das **yppasswd**–Kommando wird zum Ändern des persönlichen Sicherheitswortes benutzt.

yppasswd # Syntax

Ein Paßwort muß 8 Zeichen lang sein und soll mindestens ein Sonderzeichen und eine Ziffer enthalten. **Doppelpunkt (:) ist verboten!**

Aus Sicherheittsgründen müssen diese Konventionen eingehalten werden. Zeichenfolgen mit mehr als 8 Zeichen werden abgeschnitten (die längere Eingabe ist also umsonst).

Ein Sicherheitswort muß "blind" eingetippt werden, da bei der Eingabe der Echo-Mechanismus (Ausgabe auf dem Bildschirm) abgeschaltet wird.

5.10 find (Suchen nach Dateinamen)

Das **find**–Kommando ist hilfreich, eine Datei, deren Namen bekannt ist, in der Verzeichnishierarchie zu lokalisieren. Die Suche beginnt bei dem angegebenen Verzeichnis und erstreckt sich über die hier beginnende Teilhierarchie (*Subdirectories*) nach unten.

find *Startdirectory* **-name** *Dateiname* # Syntax

Optionen (unvollständiger Auszug)

`-atime n`	Diese Bedingung ist erfüllt, wenn auf die betrachtete Datei in den letzten `'n'` Tagen zugegriffen wurde.
`-name dateiname`	Spezifiziert einen Dateinamen unter Benutzung der üblichen Shell-Metazeichen. Falls Metazeichen eingesetzt werden, müssen sie mit dem Fluchtsymbol (siehe Beispiel unten) versehen sein, um an das **find**–Kommando weitergereicht werden zu können.
`-print`	Diese Option ist mittlerweile Standard geworden, sie braucht nicht mehr angegeben zu werden!
`-size n`	Definiert die Größe einer Datei in Blöcken.
`-user`	Definiert den Eigentümer einer Datei.

Beispiele:

`find ./name/otto -user peter`	Gibt alle Dateien aus, die dem Benutzer `'peter'` gehören und die im Unterbaum `'/name/otto'` gespeichert sind.
`find /otto -size +100 -exec ls -l {}\;`	Findet alle Dateien ab dem Startverzeichnis `'/otto'`, die größer als 100 Blöcke sind, und listet sie mit dem **ls**–Kommando auf. Im Kommando wird das Argument '{}' durch das aktuelle Verzeichnis ersetzt. Das Ende des Kommandos wird durch ein mit dem Fluchtsymbol '\' versehenes Semikolon angezeigt.
`find /otto -atime +100`	Gibt alle Dateien ab dem Startverzeichnis `'/otto'` aus, auf die in den letzten 100 Tagen nicht zugegriffen wurde.
`find . -name "*.c"`	Gibt alle Dateien ab dem aktuellen Verzeichnis und dem dazugehörigen Unterbaum aus, die die Dateinamenergänzung `'.c'` enthalten.

`find .`
`find . -print`
`find . -name "*"`
	Diese 3 Kommandos erzeugen immer das selbe Resultat: Auflistung aller Dateien der aktuellen Directory.

6 Dateiverwaltung

Eine wichtige Aufgabe des Benutzers ist es, eine unumgängliche Dateiverwaltung zu betreiben. Dazu gehört von Zeit zu Zeit alte Dateien zu löschen, Dateien umzubenennen, Dateien zu kopieren, Zugriffsrechte einiger Dateien zu ändern oder eine übersichtlich, funktionsbezogene Verzeichnisorganisation anzulegen. UNIX bietet dazu folgende Werkzeuge an:

6.1 mkdir (Erzeugen von Verzeichnissen (*Directories*))

Mit dem **mkdir**–Kommando (**m**a**k**e **dir***ectory*) werden Verzeichnisse erzeugt. Dazu wird die Schreiberlaubnis des Vorgänger–Verzeichnisses benötigt. Mit dem Erzeugen eines Verzeichnisses werden die '.' und '. .' –Dateien automatisch eingetragen.

mkdir *Verzeichnisname* # Syntax

Beispiel:

`mkdir name_tex`	Aus dem aktuellen Verzeichnis heraus wird ein neues Unterverzeichnis mit dem Namen `'name_tex'` angelegt.

6.2 rmdir (Löschen von Verzeichnissen (*Directories*))

Verzeichnisse werden mit dem Kommando **rmdir** (**r**e**m***ove* **dir***ectory*) gelöscht. Dazu muß das Verzeichnis leer sein. Nach Definition ist ein Verzeichnis leer, wenn es nur noch die Einträge '.' und '. .' enthält. Um ein Verzeichnis zu löschen, muß das Schreibrecht im Vorgänger–Verzeichnis des zu löschenden Verzeichnisses gesetzt sein.

rmdir *Verzeichnisname* # Syntax

6.3 rm (Dateien löschen)

Das **rm**–Kommando (**r**e**m***ove*) erlaubt das Löschen normaler Dateien. Um eine Datei löschen zu können, benötigt der Benutzer das Schreibrecht für das Verzeichnis, in dem die Datei angelegt ist. Das Schreibrecht für die Datei ist nicht nötig. Wenn eine Datei schreibgeschützt ist, fragt das System nach, ob die Datei wirklich gelöscht werden soll. Gelöschte Dateien sind **nicht** mehr rekonstruierbar, außer, es werden vom Operating regelmäßig Sicherungskopien durchgeführt.

rm *[Optionen] Dateiname* # Syntax

Beispiele:

`rm meinedatei`	Löscht die Datei `'meinedatei'` (**Ohne Rückfrage, unwiderruflich!**).
`rm a1 a2 a3`	Löscht in einem Kommando die Dateien `'a1', 'a2'` und `'a3'` (**Ohne Rückfrage, unwiderruflich!**).
`rm *`	**Vorsicht!!** Löscht **alle** normalen Dateien im aktuellen Verzeichnis. Punkt–Dateien bleiben erhalten (**Ohne Rückfrage, unwiderruflich!**).
`rm -i kap*`	Die Option '–i' verlangt bei jeder Datei mit dem Namensbeginn `'kap'` eine Bestätigung, ob sie gelöscht werden soll oder nicht.
`rm -r buchdir`	Löscht das Verzeichnis `'buchdir'`, alle Einträge und den gesamten Unterbaum dieses Verzeichnisses. Da die Option '–r' den gesamten Baum löscht, der zum angegebenen Verzeichnis gehört, sollte diese Option **sehr vorsichtig** eingesetzt werden (**Ohne Rückfrage, unwiderruflich!**).

6.4 mv (Dateien umbenennen)

Das **mv**–Kommando (**m**o**v**e) erlaubt Dateiumbenennungen. Es bewegt eine oder mehrere Dateien von einem Platz der Verzeichnis–Struktur an eine andere Stelle. Liegen beide Stellen im selben Dateisystem, ist die Bewegung lediglich ein Umbenennen der Datei. Wenn die beiden Positionen allerdings in zwei verschiedenen Verzeichnissen liegen, müssen die Daten der Datei in das andere Verzeichnis verschoben werden.

mv *[Optionen] vorhandener_Name neuer_Name* # Syntax
mv *[Optionen] Datei... Directory* # Syntax

Beispiele:

`mv quelle ziel`	Umbenennen der Datei `'quelle'` in `'ziel'`. Eine bereits existierende Datei `'ziel'` wird dabei überschrieben!
`mv -i quelle ziel`	Falls beim Umbenennen die Datei `'ziel'` schon existiert, wird über die Option '–i' nachgefragt, ob diese Datei gelöscht werden soll.
`mv otto neudir`	Verschiebt die Datei `'otto'` in das Verzeichnis `'neudir'`. Die Datei hat nachher den selben Basisnamen, aber einen anderen Pfadnamen.
`mv dir1 dir2`	Schiebt den Inhalt des Verzeichnisses `'dir1'` mit allen Unterverzeichnissen in das Verzeichnis `'dir2'`. `'dir1'` existiert danach nicht mehr! Falls `'dir2'` schon existiert, wird `'dir1'` Unterverzeichnis von `'dir2'`. Falls nicht, wird `'dir2'` neu erzeugt. `'dir2'` darf **kein** Unterverzeichnis von `'dir1'` sein (Fehlermeldung)!

6.5 cp (Dateien kopieren)

Das **cp**–Kommando (**c**o**p**y) erstellt eine Kopie einer Datei. Der Unterschied zwischen **mv** und **cp** ist, daß **mv** die Quelldatei löscht, wohingegen **cp** sie bestehen läßt.

cp *[Optionen] alte_Datei neue_Datei* # Syntax
cp *[Optionen] Datei... Directory* # Syntax
cp *[Optionen] Directory... Directory* # Syntax

Beispiele:

`cp otto otto.test`	Erstellt eine Kopie der Datei `'otto'` unter dem Namen `'otto.test'`. Die Datei `'otto'` wird von der Operation nicht verändert. Eine bereits existierende Datei `'otto.test'` wird dabei (ohne Rückfrage) überschrieben!
`cp altbuch/* neubuch`	Kopiert alle Dateien aus dem Verzeichnis `'altbuch'` in das Unterverzeichnis `'neubuch'`. Das Unterverzeichnis `'neubuch'` muß existieren!
`cp -r dir1 dir2`	Kopiert alle Dateien des Verzeichnisses `'dir1'` sowie die Unterverzeichnisse rekursiv nach `'dir2'`, so daß die Datei/Verzeichnisstruktur erhalten bleibt.

6.6 ln (Verweise (*Links*) auf Dateien)

6.6.1 Hard–Link

Ein Eintrag in einem Dateiverzeichnis besteht aus einem Dateinamen und einem Zeiger auf eine Dateibeschreibung (Inode), in der unter anderem aufgeführt ist, wo auf dem Datenträger sich die Datei befindet. Ein Verzeichniseintrag ist lediglich ein Verweis, man sagt dazu ein *Link*, auf eine Datei. Damit besteht die Möglichkeit, einen derartigen Verweis in mehr als ein Verzeichnis aufzunehmen. Eine

Datei, die physikalisch nur **einmal** vorhanden ist, ist dann in mehreren Verzeichnissen eingetragen. Verweisen also zwei Namen im Dateisystem auf den gleichen Inode, so existieren zwei *Hard–Links* auf diese Datei. Wird einer der beiden Namen durch **rm** gelöscht, so wird nur der Link auf diese Datei gelöscht. Die Datei ist immer noch unter dem anderen Namen zugänglich.

ln *Dateiname Directory* # Syntax
ln *Dateiname Linkname (zweiter Name)* # Syntax

Mit dem Kommando:

ls –li | sort # Beispiel

kann die angelegte Inode angezeigt werden. Das nachgeschaltete **sort**–Progamm zeigt Dateien mit gleicher Inode direkt hintereinander an.

Feld 1	Feld 2	Feld 3	Feld 4	Feld 5
239746	-rw-r--r--	1	user	72 Aug 3 11:28 hypo
239751	-rw-r--r--	**2**	user	37 Jul 30 10:17 oh
239751	-rw-r--r--	**2**	user	37 Jul 30 10:17 otto

Das 1. Feld jeder Zeile zeigt die zum Namen gehörende Inode und das 3. Feld (Linkzähler) die Anzahl der Hard–Links (normalerweise: **1**).

6.6.2 Symbolischer Link

Nützlicher als der Hard–Link ist der symbolische Verweis (*Symbolic Link*), da dieser nicht datenträgerspezifisch gebunden ist (Inodes sind immer nur eindeutig für Dateien desselben Dateisystems). Symbolische Links sind dateisystemübergreifende Verweise und sie werden durch eine entsprechend gekennzeichnete Datei realisiert. In dieser Datei steht der absolute Pfadname der Datei, auf die verwiesen wird. Ein weiterer Vorteil ist, daß sie auch auf Verzeichnisse (Directories) zeigen können. Außerdem geht der Verweis nicht verloren, wenn die ursprüngliche Datei durch eine neue Version ersetzt wird.

ln –s *[Pfad]Dateiname Symbname* # Syntax

Da ein symbolischer Link selbst eine Datei ist, kann er durch das Kommando **rm** *Symbname* entfernt werden.

```
lrwxr-xr-x 1 root        9 Jan 8 1992 spool -> var/spool
lrwxr-xr-x 1 root        7 Jan 8 1992 tmp -> var/tmp
```

Ein symbolischer Link zeigt in der 1. Spalte (Dateityp) ein **l** (**l***ink*) und im Dateinamenfeld den Verweis *Symbname –> [Pfad]Dateiname* an.

6.7 chmod, chown, chgrp (Dateimodus ändern (Zugriffsrechte))

Drei Dinge darf ein Benutzer mit einer Datei tun: lesen, darauf schreiben und ausführen. Die Zugriffsrechte (siehe Abbildung 1) legen nun fest, wer was mit dieser Datei tun darf. Sie können durch die Kommandos **chmod** (**ch***ange* **mod***e*), **chown** (**ch***ange* **own***er*) und **chgrp** (**ch***ange* **gr***ou***p**) durch den Benutzer oder den Super–User geändert werden. Das Kommando **chmod** wird zum Ändern der Zugriffsrechte für Dateien und Verzeichnisse benutzt. Es kann in einer etwas umständlichen relativen *Operator*– oder einer knappen absoluten *Zahlencode*– Schreibweise benutzt werden.

Zugriffsrechte über Operatoren		Zugriffsrechte über Zahlencode		
Operator	Ausführung	Kürzel	Erlaubnis	Zahlencode
-	Entzug für ein Recht.	r	lesen	4
+	Erlaubnis für ein Recht.	w	schreiben	2
=	Weise ein Recht zu.	x	ausführen	1
		-	enzieht alle Rechte	0

Hinweis: Zwischen einer relativen Operatoren– und einer absoluten Zahlencode–Zuweisung ist ein deutlicher Unterschied! Wenn für die Datei `'aufg1.c'` im folgenden Beispiel kein Schreibrecht vorhanden war, bleibt dieses Recht bei der Zuweisung **a-x** unberührt, während bei einer Zuweisung **666** Lese– und Schreibrecht für alle Gruppen explizit gesetzt wird. Entsprechend sind auch die Unterschiede in den anderen Beispielen mit dem Kommando **chmod**.

Beispiele:

Operatoren–Zuweisung	Zahlencode–Zuweisung	**Erklärung**
`chmod a-x aufg1.c`	`chmod 666 aufg1.c`	**a**(*lle*) = **user, group, other** wird das Recht zum **Ausführen** für die Datei `'aufg1.c'` entzogen (**6=4+2**).
`chmod g=rwx test.f`	`chmod 770 test.f`	**user** behält alle Rechte, **group** bekommt alle Rechte, **other** hat keine Rechte für die Datei `'test.f'` (**7=4+2+1**).
`chmod go+x labor1`	`chmod 711 labor1`	**user** behält alle Rechte, **group** und **other** bekommen das Recht zum **Ausführen** für die Datei `'labor1'` zugesprochen.
`chmod a-w .`	`chmod 555 .`	Verhindere die Erzeugung einer Datei im aktuellen Verzeichnis (**5=4+1**).
`chmod go-rwx dirneu`	`chmod 700 dirneu`	Das Unterverzeichnis `'dirneu'` behält für **user** alle Rechte. Für **group** und **other** werden keine Rechte eingerichtet.

Eigentümer ändern	**Erklärung**
`chown koch aufg1.c`	**Achtung: Nur mit Superuser–Privileg möglich!** Ändere den Eigentümer der Datei `'aufg1.c'` in `'koch'`.

Gruppe ändern	**Erklärung**
`chgrp labor test*`	**Achtung: Wenn man nicht Owner ist oder nicht zur neuen Gruppe gehört, benötigt man Superuser–Privileg!** Übereigne alle Dateien, deren Namen mit den Zeichen 'test' beginnen, der Gruppe 'labor'.

6.8 umask (Unterbinden von Zugriffsrechten)

Beim Erzeugen von Dateien legen Prozesse automatisch Zugriffsrechte für die erzeugten Dateien fest, die aus Sicht des Eigners für **g**roup und **o**thers meistens zu viel erlauben.

umask *ugo* # Syntax

Dieses Kommando unterbindet für den laufenden Prozeß und alle seine Nachkommen das automatische Setzen bestimmter Rechte.

ugo steht für eine dreistellige Zahl. Die erste Ziffer beeinflußt die Rechte für den **u**ser, die zweite die für **g**roup und die dritte die für **o**thers.

Jede dieser Ziffern kann einen Wert zwischen **0** und **7** mit folgender Bedeutung annehmen.

Achtung: Bei **umask** bedeuten die Ziffern genau das Gegenteil wie bei der Zahlencode–Zuweisung mit dem Kommando **chmod**.

Zugriffsrechte beim Kommando: umask	
Zahlencode	Ausführung
4	unterbindet lesen (**r***ead*)
2	unterbindet schreiben (**w***rite*)
1	unterbindet ausführen (*e***x***ecute*)
0	unterbindet **nichts** (alle Rechte erlaubt!)

Beispiele:

`umask 000` Unterbindet nichts (**0**) (alle Rechte erlaubt!)

`umask 027` Für **u**ser wird nichts unterbunden (**0**).
Für **g**roup wird das **Schreiben** unterbunden (**2**).
Für **o**thers wird **alles** unterbunden (**7 = 4 + 2 + 1**).

alias cdfor "cd /home/cip1/name/name_for; umask 077; pwd" # Beispiel

Dieses Abkürzungskommando 'cdfor' führt ein Wechsel vom Heimatverzeichnis 'name' in das Unterverzeichnis 'name_for' (= FORTRAN–Dateien) durch und unterbindet über das Kommando **umask 077** den Zugriff Lesen, Schreiben und Ausführen dieser Dateien für **g**roup und **o**thers. Außerdem wird nach dem Wechsel der Pfad des aktuellen Verzeichnisses durch **pwd** angezeigt.

7 Hilfsprogramme für Textdateien

7.1 cat (Verknüpfe und drucke Dateien)

cat *Dateiliste* # Syntax

(*con***cat***enate*) ist ein sehr vielseitiges Kommando. Normalerweise wird es zur Ausgabe von Dateien auf dem Bildschirm eingesetzt. Es kann aber auch mehrere Dateien in einer vorgegebenen Folge zu einer Datei zusammenfügen (aneinanderhängen). Seine volle Wirkung entfaltet es erst im Zusammenhang mit der *Umlenkung* von Standard–I/O und mit *Pipes*.

Beispiele:

`cat otto`	Druckt die Datei `'otto'` auf dem Bildschirm aus.
`cat labortest | more`	Druckt die Datei `'labortest'` seitenweise auf dem Bildschirm aus.
`cat kap1 kap2 kap3`	Verknüpft die angegebenen Dateien und drucke sie hintereinander auf dem Bildschirm aus.
`cat kap1 kap2 kap3 > alle`	Verknüpft die angegebenen Dateien und übertrage sie über Umlenkung in die Datei mit dem Namen `'alle'` (ohne Bildschirmausgabe!).
`cat /dev/null/ > Datei.neu`	Erzeugt eine leere Datei unter dem Namen `'Datei.neu'`.

Hinweis: Eine "durchlaufende" Bildschirmausgabe kann mit `CTRL-s` angehalten und mit `CTRL-q` wieder gestartet werden.

7.2 pr (Dateien betiteln und formatieren)

Mit dem **pr**–Kommando werden ASCII—Dateien zum Ausdrucken vorbereitet. **pr** versieht Dateien mit Kopf– und Fußzeilen, unterteilt eine Datei in Spalten und paßt sie an verschiedene Seitenlängen und Weiten an. Standard ohne Option ist eine einspaltige Ausgabe mit 66 Zeilen pro Seite und einer kurzen Überschrift (Datum und Zeit, Dateiname, Seitenzahl). Ohne Dateiangabe wird die Standard–Eingabe verarbeitet. Die Optionen zur Kontrolle des Ausgabeformats sind über *Manual–Pages* nachzulesen.

pr *[Optionen][Datei ...]* # Syntax

Beispiele:

`pr buch | lpr  -Peg`	Bereite die Datei `'buch'` mit kurzer Überschrift auf und drucke sie im konventionellen Format auf dem Standarddrucker aus.
`ls | pr -6`	Bereite die Ausgabe des ls–Programms auf in eine sechsspaltige Einteilung mit Überschrift. Ausgabe = Standard–Ausgabe (Bildschirm).
`pr +10 kap1 > kap1.neu`	Aufbereiten der Datei `'kap1'` ab Seite 10 und übertragen in die Datei `'kap1.neu'`.

7.3 more (Seitenweises Portionieren der Bildschirmausgabe)

Das Kommando **more** gibt Daten seitenweise auf dem Bildschirm aus (1 Seite = Größe des Fensters).

more *Dateiname* # Syntax

more reicht von der Standard-Eingabe so viele Zeilen zur Standard-Ausgabe weiter, wie im aktuellen Fenster darstellbar sind. Die unterste Zeile im Fenster zeigt invers Informationen über die auszugebenden Daten mit Namen und Prozentzahl an.

more kennt innerhalb eines Aufrufes sehr nützliche Kommandos, die das Arbeiten wesentlich erleichtern. **more**–Kommandos mit einem Stern können mit einer vorangesetzten Zahl versehen werden.

`h`		Information aller **more**–Kommandos.
`=`		Gibt Dateiname und Status aus.
`f, Leertaste; b`	*	Vorwärts; rückwärts N Zeilen, default = 1 Fensterseite.
`j, RETURN-Taste; k`	*	Vorwärts; rückwärts N Zeilen, default = 1 Zeile.
`/pattern; ?pattern`	*	Vorwärts– ; rückwärts–Suchen nach einer Zeichenfolge.
`q, ZZ`		Bricht die Ausgabe ab.

Beispiele:

`more otto.dat`	Gibt seitenweise die Datei `'otto.dat'` im aktuellen Bildschirmfenster aus.
`ls -l /usr | more`	Gibt seitenweise eine Verzeichnisliste des Verzeichnisses `'/usr'` über den Pipe-Mechanismus aus.

7.4 lpr (Dateien ausdrucken, Druckkommandos)

Das **lpr**-Kommando sendet Dateien an den Drucker zum Drucken. **Standard-Drucker** für alle Benutzer ist der Drucker im Erdgeschoß des Instituts für Numerische und Angewandte Mathematik mit dem Druckernamen **'eg'**.

ASCII–, Postscript–, HP-PCL (Laserjet)– und dvi– Files:

`lpr` *Dateinamenliste* # Kommando für Standarddrucker

pdf–Files:

pdf–Files können nicht direkt mit dem lpr–Kommando gedruckt werden. Diese Files müssen mit dem Programm 'acroread' umgewandelt werden!

`acroread` *filename*`.pdf &` # Aufruf, (über Print-Button ausdrucken)

Sonderformate

ASCII–Files:

`/usr/lbin/ln08rof -K1 <` *filename(s)* `| lpr -Peg` (einseitig, Normalgröße)

Formatierte, verkleinerte Ausgabe mit Rand und Kopfzeile

`a2ps -1` *filename* (doppelseitig, Querformat)

`liste` *filename(s)* (einseitig, default–Einstellung)

`liste -K2` *filename(s)* (doppelseitig)

ps–Files:

`/usr/lbin/ln08rof -K1 <` *filename.ps* `| lpr -Peg` (einseitig, nicht für HP-PCL-Files !)

dvi–Files:

`lprdvi -Peg -K1` *filename*`.dvi` (einseitig)

Weitere Möglichkeiten der K-Option z.B. '–Ktumble' findet man mit: `man lpr`

`lprdvi -Peg -t landscape` *filename*`.dvi` (doppelseitig, Querformat)

Weitere Optionen findet man mit: `man dvips`

`lprdvi -Peg -pp 10:20` *filename*`.dvi` (nur Seiten 10 - 20)

```
lprdvi -Peg -p 17 filename.dvi
```
(ab Seite 17)

```
lprdvi -Peg -l 19 filename.dvi
```
(bis Seite 19)

Druckausgabe umsteuern

`lpr -P`*druckername* *filename(s)* — nur auf den Druckern im Institut für NAM und im MATHE–Institut, über die Option **–P** und *druckername*.
Die Druckernamen sind an den Druckern ausgewiesen.

Drucken aus dem Emacs-Menü: (Tools, Print, Print Buffer)

Die folgende Zeile in das .emacs - Startfile unter Variablen und Funktionen betr.: Files einfügen:

```
(setq lpr-switches '("-Peg"))        ; Print Emacs-Buffer im ASCII-Mode
```

PC–zwischengespeicherte ps–Files:

ps–Files, die z.B. auf einem Windows–PC zum Drucken zwischengespeichert werden, enthalten im Normalfall am Beginn Druck–Code, der eine Erkennung als Postscript–Datei verhindert. Das dadurch falsch interpretierte Postscript–File wird als Source–Code auf dem Drucker ausgegeben (**Druckschrott!!**).

Der PC–spezifische Druck–Code muß mit einem Text-Editor entfernt werden:

⋮ --------------------〉 PC–Code muß für NAM–Drucker entfernt werden!

%!PS-Adobe ... ⎤

⋮ 〉 **korrekter Postscript–Code**

%%EOF ⎦

⋮ --------------------〉 PC–Code muß für NAM–Drucker entfernt werden!

7.4.1 Druckernamen, Druckerstandorte

Folgendende Drucker können zur Zeit benutzt werden:

Drucker–name	Institut	Standort	Typ, Ausgabeform
`keller`	NAM	Keller, Bildschirmraum	Laser, einseitig,
`eg`	NAM	EG, Flur vor Operatorraum	Laser, ein– oder doppelseitig
`buero`	NAM	1. Stock, Flur vor Geschäftszimmmer.	Laser, ein– oder doppelseitig
`bibi`	NAM	2. Stock, Vorraum Bibliothek	Laser, ein– oder doppelseitig
`dach`	NAM	3. Stock, Flur	Laser, einseitig
`umilas`	MATHE	EG, Vorraum zum CIP-Raum	Laser, ein– oder doppelseitig

7.4.2 Druckerwarteschlangen abfragen

Druckaufträge, die sich in einer Druckerwarteschlange befinden, können vom Benutzer abgefragt werden. Die Abfrage ist nur auf dem Rechner möglich, an dem der Druckjob abgesendet wurde.

lpq **–P***Druckername* # Syntax

7.4.3 Drucken von dvi–Dateien, doppelseitig und deckungsgleich

Die Standard–Einstellung für doppelseitiges Drucken auf dem Laserdruckern im Institut für NAM und im MATHE–Institut ist nicht deckungsgleich. Die Vorraussetzungen für ein deckungsgleiches Drucken sind in der TEX–Quell–Datei vorzunehmen. Dazu müssen im Vorspann (*preamble*) die Befehle `twoside`, `\oddsidemargin` und `\evensidemargin` eingesetzt werden:

```
\documentclass[...,twoside,...]{article}
\usepackage{german,...,...]
      ⋮
\oddsidemargin...cm
\evensidemargin...cm
\begin{document}
      ⋮
```

7.5 wc (Zählen von Zeilen, Worten und Zeichen)

Oft ist es hilfreich zu wissen, wie eine Textdatei aufgebaut ist. **wc** (**w**ord **c**ount) liefert zu einer Datei die Anzahl ihrer Zeilen, Wörter und Zeichen. Unter einem *Wort* wird hier jede Zeichenfolge verstanden, die durch Leerzeichen, Tabulatorzeichen, Zeilenanfang oder Zeilenende begrenzt wird.

wc *[-lwc] [Datei]* # Syntax

Als Optionen sind möglich: **l** (Zeilen), **w** (Worte) oder **c** (Zeichen).

wc tiger.dat # Beispiel

liefert z.B. die Ausgabe:

55 461 3179 tiger.dat (55 Zeilen, 461 Worte, 3197 Zeichen und den [Pfad] Dateinamen)

Beispiele:

`wc -w ueb?.c`	Druckt die Anzahl der Worte und Dateinamen aller Dateien aus, die mit den Zeichen `'ueb'` beginnen, und mit der Dateinamenserweiterung `'.c'` enden.
`ls -l | wc -l`	Zählt die Dateinamen des aktuellen Verzeichnisses und druckt **nur** die Zeilenanzahl (in diesem Fall die Dateienanzahl) aus.

7.6 diff (Vergleichen von Dateien oder Verzeichnissen)

Das **diff**–Programm zeigt an, welche Zeilen sich bei einem Vergleich von zwei Textdateien unterscheiden. Bei der Anwendung auf Directories werden nicht vorhandene Dateien angezeigt und, falls Textdateien gleichen Namens vorhanden sind, die unterschiedlichen Zeilen dieser namensgleichen Textdateien ausgegeben.

diff *[Optionen] Datei1 Datei2* # Syntax
diff *[Optionen] Directory1 Directory2* # Syntax

Wenn kein Unterschied festgestellt wird, erscheint ohne Meldung wieder der Prompt. Beim Vergleichen von zwei Textdateien kann z.B. folgender Text ausgegeben werden:

`3c3`	1), 4), 2)
`< Duell der Tiger (Auszug)`	6)
`---`	7)
`> Duel der Tiger (Auszug)`	8)

Erklärung der Symbole:

1)	`n1, n2`	Zeilennummer(n) [von, bis] der Datei1.
2)	`n3, n4`	Zeilennummer(n) [von, bis] der Datei2.
3)	`a`	Fehlende Textzeile(n) in Datei1.
4)	`c`	Fehlende(s) Zeichen in der angegebenen Zeilennummer.
5)	`d`	Fehlende Textzeile(n) in Datei2.
6)	`<` *Text*	Vorgespielter Text der Datei1.
7)	`---`	Trennzeile zwischen dem ausgegebenen Text von Datei1 und Datei2.
8)	`>` *Text*	Vorgespielter Text der Datei2.

7.7 sort (Dateiinhalte sortieren oder vermischen)

Das **sort**–Programm sortiert und/oder vermengt Zeilen der Eingabedatei und schreibt die neu geordneten Zeilenfolgen auf die Standard–Ausgabe oder in eine gewählte Ausgabedatei, wobei die Eingabedatei unverändert bleibt. **Sort** kennt einen zeilenbezogenen Feldbegriff, d.h. jede Zeile wird in *Felder* eingeteilt, deren Numerierung bei Null beginnt. Ein Feld wird durch sogenannte Feldtrenner bestimmt. Voreingestellt sind Leer– oder Tabulatorzeichen, Zeilenanfang oder –ende. Es können auch eigene Feldtrennzeichen vereinbart werden. Wird z.B. der Doppelpunkt als Feldtrenner genommen, so ist das Leerzeichen gültiger Bestandteil von Feldern.

Als Sortierschlüssel kann die ganze Zeile (Standard), oder innerhalb einer Zeile ein oder mehrere Sortierschlüssel angegeben werden. Die Sortierschlüssel *Feld 1 ... Feld n* werden jedoch nur dann betrachtet, wenn die vorhergehenden Felder keine Sortierung ergeben.

sort *[–cmu][–tc][–bdfinr][+pos1 [–pos2]] ... [–o Ausgabedatei] [Eingabedatei ...]* # Syntax

Optionen: (unvollständiger Auszug)

`-m`	Vermenge die Eingabedateien (Die Eingabedateien sollten bereits sortiert sein).
`-tc`	Das Zeichen **'c'** (`für character`) dient zur Feldbegrenzung der Sortierschlüssel.
`-f`	Wandele Großbuchstaben bei Feldvergleichen in Kleinbuchstaben um (Die Ausgabe zeigt den uspründlichen Buchstaben).
`-u`	Unterdrücke alle gleichen Zeilen bis auf eine. Als Gleichheit der Zeilen gilt, daß sie an allen Sortierschlüsselpositionen gleich sind (Felder außerhalb der Sortierschlüssel sowie zu ignorierende Zeichen werden nicht herangezogen).
`-n`	Führe numerische Vergleiche durch.
`-r`	Sortiere in umgekehrter Reihenfolge (Standard ist aufsteigende Reihenfolge).
`+pos1`	Beginn der Sortierposition.
`-pos2`	Ende der Sortierposition. (Ohne diese Angabe reicht die Sortierposition bis zum Ende der Zeile).

Eine hypothetische Datei `'hypo'` hat Zeilen mit folgendem Feldaufbau:

Feld 0	Feld 1	Feld 2	Feld 3	Feld 4	Feld 5	Feld 6
Meier	Petra	13054	Berlin	28	ca50wg	a:40
Hoff	Otto	37077	Göttingen	1	na60mg	a:30
Andes	Hans	81527	München	17	fa70mg	a:20

Beispiele:

`sort -o ausg hypo`	Sortiert zeilenweise, aufsteigend die Eingabedatei `'hypo'` in die über die Option `'-o'` gewählte Ausgabedatei `'ausg'`.
`sort -r hypo > ausg`	Sortiert zeilenweise, in umgekehrter Folge, die Eingabedatei `'hypo'` in die gewählte Ausgabedatei `'ausg'` über Umlenkung.
`sort -n +4 -5 hypo`	Sortiert numerisch, aufsteigend, beginnend mit dem 1. Zeichen von *Feld 4*. ('+4' bedeutet, 4 Felder werden übersprungen; '–5' bedeutet, Sortierposition endet mit dem letzten Zeichen von *Feld 4*). Eingabedatei = `'hypo'`. Ausgabe = Standard–Ausgabe.
`sort -n +5.2 -5.4 hypo`	Sortiert numerisch, aufsteigend, beginnend mit der Positionsangabe 3. Zeichen von *Feld 5* und endet mit der Positionsangabe 4. Zeichen von *Feld 5*. Ein- und Ausgabe wie im Beispiel vorher.
`sort -t: +5n hypo`	Sortiert numerisch, aufsteigend, beginnend nach dem Doppelpunkt in *Feld 6*. Ein- und Ausgabe wie im Beispiel vorher.

7.8 grep (Textmuster in Dateien suchen)

Viele UNIX–Werkzeuge arbeiten mit Text–Suchmustern. Sie werden oft kurz *Suchmuster* oder *Text–Suchmuster* genannt und heißen auch *reguläre Ausdrücke*. Dieser Begriff hat dem **grep** seinen Namen gegeben: *get regular expression*! Für Text–Suchmuster gelten folgende Regeln:

Regel 1: Kein Suchmuster wirkt über ein Zeilenende hinaus.

Regel 2: Paßt ein Suchmuster zu mehreren Zeichenfolgen einer Zeile, so wird stets die längste (und bei mehreren die erste) als *Treffer* angesehen.

Reguläre Ausdrücke unterliegen einer strengen Syntax, die ausführlich in den Manual Pages nachzulesen ist. Hier sei nur folgendes erklärt:

Jedes ASCII–Zeichen "*c*" ist regulär. Ein ASCII–Zeichen "*c*" in einem Suchmuster paßt zu sich selbst in einer Textzeile, außer es ist eines der folgenden Sonderzeichen:

\   [   ]   .   *   _ $ ^ | (   )

Beispiel Das Suchmuster *a* paßt zu *a* in einer Textzeile.
Das Suchmuster *Meier* paßt zu *Meier* in einer Textzeile.

Das Sonderzeichen "\" (Backslash) entwertet eine eventuelle Sonderbedeutung eines ASCII–Zeichens.

Beispiel: Das Suchmuster *3\.14* paßt zu *3.14* in einer Textzeile (Die Sonderbedeutung des Punktes wird hier duch den Backslash 'entwertet').

Soll nach Sonderzeichen gesucht werden, ist es ratsam, diese Sonderzeichen durch Hochkommas ('...') zu entwerten (siehe Beispiele unten).

Kommandos der **grep**–Familie durchsuchen Dateien nach Zeichenfolgen. Wenn das Text–Suchmuster in einer Zeile gefunden wurde, wird der Name der Datei und die gesamte Zeile normalerweise in die Standard–Ausgabe geschrieben. Die **grep**–Familie besteht aus folgenden Werkzeugen:

fgrep *Fast Grep* kann nur reguläre Ausdrücke ohne Sonderzeichen verarbeiten.

grep Verarbeitet reguläre Ausdrücke, die auch Sonderzeichen im Sinne von Wildcards enthalten dürfen.

egrep *Extented Grep* verarbeitet erweiterte reguläre Ausdrücke mit zusätzlichen Sonderzeichen wie z.B. **+** und **?**.

grep *[Optionen] [Suchmuster] [Dateiname(n)]* # Syntax

Optionen: (unvollständiger Auszug)

- `-n` Gibt die Zeilennummer der Datei aus, in der ein Ausdruck gefunden wurde.
- `-i` Ignoriert den Unterschied zwischen Groß- und Kleinschreibung (nur bei **grep** und **fgrep**).
- `-v` Gibt alle Zeilen aus, die **nicht** zu dem Muster passen.

Beispiele:

`grep -n dei kap*`	Durchsucht alle Dateien mit den Anfangszeichen `'kap'` des aktuellen Verzeichnisses nach dem Textmuster "dei" (als 'Dreher' von "die") und listet die Dateinamen und dazugehörigen Textzeilen in Standard–Ausgabe auf.
`grep 'M[ea][iy]er' kap*`	Durchsucht alle Dateien mit den Anfangszeichen `'kap'` des aktuellen Verzeichnisses nach dem Textmuster "Meier", "Meyer", "Maier", "Mayer" und gibt die Dateinamen und die dazugehörige Textzeile in Standard–Ausgabe aus.
`ls -l | grep '^d'`	Die Ausgabe wird über eine Pipe von **ls** nach **grep** geleitet. Durchsucht wird das aktuelle Verzeichnis über das Textmuster `'d'` nach Unterverzeichnissen. Das vorangestellte Sonderzeichen, `'^'` (Hütchen), bedeutet bei **grep** den Beginn einer Zeile. Ausgabe = Standard–Ausgabe.
`ls -l | grep 'r$'`	Dieses Kommando leitet die Ausgabe über eine Pipe von **ls** nach **grep**. Aufgelistet werden alle Zeilen des aktuellen Verzeichnisses, die mit dem Textmuster `'r'` enden. Das hintenangestellte Sonderzeichen `'$'` (Dollar) bedeutet bei **grep** das Ende einer Zeile. Ausgabe = Standard–Ausgabe.

7.9 cut und paste (Zeilenteile ausschneiden und zusammenfügen)

Das **cut**–Programm schneidet Textstücke aus jeder Zeile einer Datei aus und das **paste**–Programm fügt oder mischt in Dateien abgelegte Teile zu einer Datei zusammen. Das **cut**–Programm ist deshalb äußerst nützlich für Dateien mit tabellarischem Aufbau.

cut –**c***Liste [Datei1 Datei2]* # Syntax

paste *[Datei1 Datei2]* # Syntax

Die Option –**f** statt –**c** bewirkt, daß Felder statt Zeichen bei der Auswahl gezählt werden. Die Feldgrenze ist standardmäßig das **<Tab>**–Zeichen. Es können auch andere Zeichen als Feldbegrenzung benutzt werden.

Die Datei `'adresse'` hat folgenden Aufbau:

otto	x123	0551-121314
nina	y456	0551-232425
mark	z789	0551-343536

Beispiele:

`cut -f3 adresse > telnum`	Kopiert das 3. Feld in die Datei `'telnum'`.
`cut -c7-9 adresse > schluessel`	Kopiert das 7. bis 9. Zeichen in die Datei `'schluessel'` (Das Feldberenzungszeichen wird als ein Zeichen dabei gewertet).
`ls -l | cut -c-10,45-`	Listet nur die Spalten 1 – 10 (Dateitypen, Zugriffsrechte) und ab Spalte 45 – Ende (Dateinamen) des aktuellen Verzeichnisses auf.
`ls | paste - - - -`	Listet das aktuelle Verzeichnis in vier Spalten auf.
`paste datei1 datei2 > neu`	Fügt die Dateien `'datei1'` und `'datei3'` zu der neuen Datei `'neu'` zusammen.

7.10 tee (Ausgabe verdoppeln)

Das **tee**–Programm liest seine Eingabedaten von der Standard–Eingabe und leitet die gelesenen Daten in die Standard–Ausgabe und entweder in eine oder mehrere Dateien weiter. Die Bedeutung des Programms ist analog zu einem T–Stück in einer Rohrleitung, das den Rohrinhalt auf mehrere Rohre verteilt. Tee wird meist benutzt, um die Ausgabe eines Programms in einer Datei abzuspeichern und gleichzeitig auf dem Bildschirm darzustellen.

Das Kommando

grep -n *heute kap** | **tee** *templiste* # Beispiel

durchsucht alle `'kap'`–Dateien des aktuellen Verzeichnisses nach dem Textmuster "`heute`" und gibt die Dateinamen, die entsprechenden Zeilennummern und die dazugehörige Textzeile sowohl in der Standard–Ausgabe als auch in der Datei `'templiste'` aus.

ls | **tee** *lsdat* | **pr** *–5* | **tee** *spalten5* | **lpr** **–Peg** # Beispiel

Die Ausgabe des **ls**–Programms wird in der ursprünglichen Form in der Datei `'lsdat'` gerettet. Dann wird die **ls**–Ausgabe mit dem **pr**–Kommando in eine fünfspaltige Ausgabe aufbereitet, in die Datei `'spalten5'` übertragen und außerdem auf dem Drucker **eg** ausgegeben.

7.11 tail (Das Ende einer Datei ausdrucken)

Das tail–Programm wird zum Ausdrucken des letzten Teils der Standard–Eingabe oder einer angegebenen Datei benutzt. Es erlaubt, die letzten Zeilen einer Datei anzusehen, ohne den gesamten Text ausdrucken zu müssen.

tail *[±[startpos][lbcr]] [Datei]* # Syntax

Das Argument *startpos* ändert die voreingestellte Anzahl der auszugebenden Zeilen (Standard = 10 Zeilen). Bei einem Minuszeichen errechnet sich die Zeilenzahl vom Ende und bei einem Pluszeichen vom Anfang der Datei. Die Position des Ausdruckbeginns kann in Einheiten von Zeilen **l** (Standard), Blöcken **b** (512 Bytes) und Zeichen **c** angegeben werden.

Beispiele:

`tail -40 buch.neu`	Gibt die letzten 40 Zeilen der Datei `'buch.neu'` aus.
`tail +50 kap1.dat`	Überspringt die ersten 50 Zeilen und gibt den Rest der Datei `'kap1.dat'` aus.
`tail -20r kap2.neu`	Gibt die letzten 20 Zeilen der Datei `'kap2.neu'` in **umgekehrter** Reihenfolge (*reverse order*) aus.
`ls /usr/bin | tail -20`	Gibt die letzten 20 Dateinamen des Verzeichnisses `'/usr/bin'` aus.

8 Compiler

8.1 Allgemeines

Eine Datei enthält z.B. den Quellcode eines Programms. Um sie mit dem entsprechenden Compiler zu übersetzen, verlangt UNIX im Dateinamen eine Kennzeichnung für den Sprachtyp. Bereits der Emacs–Editor berücksichtigt bei der Bearbeitung einer solchen Datei den Sprachtyp, um beispielsweise die Tabulatoren passend einzustellen.

Compileraufruf	Compilername	Dateiendung	
gcc	C–Compiler (GNU)	**.c**	= C–Quellcode–Datei
C++	C++ Compiler (DEC)	**.cc**	= C++ Quellcode–Datei
g++	C++ Compiler (GNU)	**.cc**	= C++ Quellcode–Datei
f77	FORTRAN–Compiler	**.f**	= FORTRAN–Quellcode–Datei
pc	Pascal–Compiler	**.p**	= Pascal–Quellcode–Datei
as	Assembler–Compiler	**.s**	= Assembler–Quellcode–Datei

8.2 Optionen

Compileraufrufe können auch mit Optionen versehen werden. Sie müssen durch ein Leerzeichen (Blank) getrennt und **jede Option einzeln** mit einem Bindestrich davor eingegeben werden.

Nützliche Optionen des g++:

`-c`	Nur übersetzen, nicht binden.
`-Wall`	'Dubiosen' Code melden. Die Erfahrung zeigt, daß weit 50% der Meldungen, die hierdurch erzeugt werden, Fehler anzeigen, die notwendig korrigiert werden müssen.
`-Wwrite-strings`	Stringkonstanten sind als Konstanten zu behandeln. Bei 'sauber' formulierten Prototypen zeigt jede Meldung, die durch diese Option erzeugt wird, einen schwerwiegenden Fehler an, während Meldungen bei schlampig formulierten Prototypen in der Regel bedeutungslos sind.
`-ansi`	Einhaltung von Standard–C prüfen. Einige Erweiterungen des **g++** werden trotzdem akzeptiert.
`-pedantic`	Strikte Einhaltung von Standard–C prüfen.
`-O2`	Maximal optimieren. Diese Option empfiehlt sich immer, da manche Fehler erst im Zuge der Optimierung entdeckt werden.
`-lm`	Einbinden des mathematischen Teils der Sandardbibliohek. Diese Option muß im Gegensatz zu den übrigen hinter den Dateinamen stehen.

g++ –Wall –o aufg01 aufg01.cc –lm # Beispiel: Optionen und Argumente

Hier wird der GNU C++–Compiler aufgerufen, der über die Option **–Wall** alle entstehenden Warnungen ausgeben soll. Die ausführbare Datei bekommt über die Option **–o** den Namen `'aufg01'`. Übersetzt wird der Inhalt der Datei `'aufg01.cc'` und letztlich wird über die Option **–lm** die Bibliothek `'math.h'` eingebunden.

Bei den einzelnen Compilern gibt es unterschiedliche Optionen. Leider sind es nur ein paar wenige, die übergreifend und gleichbedeutend für alle Compileraufrufe gelten. Deshalb muß auf die Manual Pages verwiesen werden.

8.3 make (Compileraufrufe automatisieren)

Compilierungsprozeße lassen sich mit dem **make**–Kommando bequemer gestalten. Es ermöglicht, daß automatisch immer nur die Dateien neu compiliert werden, die sich seit der letzten Compilation geändert haben, sowie die Dateien, die von geänderten Dateien abhängen. **Make** verarbeitet die Datei `'makefile'` (das ist der vorgeschriebene Name), in der nach einem genauen Syntax die Anweisungen stehen.

Aufbau eines Makefiles:

```
# Makefile für Wochentagsberechnung: wievielter Tag        (Kommentarzeile)
aufg14:   aufg14.o   wochentag.o
      gcc   –Wall   –o   aufg14   aufg14.o   wochentag.o
wochentag.o:   wochentag.c   wochentag.h
      gcc   –c   wochentag.c
aufg14.o:   aufg14.c   wochentag.h
      gcc   –c   aufg14.c
```

Syntaxerklärungen zum **make**–Beispiel:

Eine Kommentarzeile beginnt mit einem "#" (*Number Sign*).

Eine Makro–Definition beginnt mit einem Namen, gefolgt von einem Doppelpunkt (`'aufg14:'`, `'wochentag.o:'`, `'aufg14.o:'`), denen Optionen und/oder Argumente durch Leer– oder Tabulatorzeichen getrennt folgen.

Eingerückt durch ein Tabulatorzeichnen folgen auszuführende Kommandos an die Shell (`gcc -Wall -o aufg14 aufg14.o wochentag.o`).

Beginnt eine Zeile nicht mit einem Tabulatorzeichen oder mit "#", beginnt wieder eine neue Makro–Definition.

Benötigen die Argumente mehr als eine Zeile, so wird eine Folgezeile mit einem "\" und der Zeichenfolge LF (neue Zeile = RETURN–Taste) ausgewiesen.

Allgemeine Erklärungen zum **make**–Beispiel:

Neu in diesem Beispiel ist die Option **–c**. Sie sagt aus, eine Datei wird compiliert, aber nicht gebunden. Die resultierende Objektdatei hat die Endung **.o** (`'wochentag.o'` und `'aufg14.o'`).

Die Abhängigkeit ist durch folgende Anordnung gegeben. Links vom Doppelpunkt werden die abhängigen Dateien aufgeführt. Rechts vom Doppelpunkt die unabhängigen. Somit ist `'aufg14.o'` abhängig von `'aufg14.c'` und von `'wochentag.h'`. Sollte sich z.B. `'wochentag.h'` ändern, werden automatisch die Dateien `'aufg14.c'` und `'wochentag.c'` jeweils zu einer neuen Objektdatei übersetzt. Ändert sich beispielsweise `'aufg14.c'` allein, wird eine neue Objektdatei `'aufg14.o'` übersetzt. Wenn durch Veränderungen neue Objektdateien entstanden sind, wird auch ein neues ausführbares Programm `'aufg14'` erstellt. **Make** erkennt eine Veränderung am Erstellungsdatum einer Datei.

9 Arbeiten mit dem Netzwerk

Kommunikationsmöglichkeiten zwischen Benutzern waren von Anfang an in das UNIX–System integriert. Eine einfache direkte Möglichkeit, mit einem anderen Benutzer Nachrichten auf dem Bildschirm auszutauschen, ist das write–Kommando (siehe Abschnitt 9.8.1). **Mail** arbeitet indirekt und realisiert einen Briefkasten–Dienst (siehe Abschnitt 9.8.2).

Heute gibt es weltumspannende und verschiedene Netzwerke (BITNET, HEPNET, SPAN, INTERNET). Die Unterschiede zwischen den einzelnen Netzwerken basieren dabei weniger auf unterschiedlichen physikalischen Kabeln, sondern auf den verschiedenen Protokollen, die in diesen Netzen angesprochen werden. Eines der bekanntesten Netze ist das INTERNET mit der TCP/IP–Protokollsuite.

9.1 TCP/IP

Die INTERNET–Entwicklung hat eine Vielzahl von Protokollen hervorgebracht, deren beiden wichtigsten IP (*Internet Protocol*) und TCP (*Transmission Control Protocol*) heißen. Die unter dem Namen TCP/IP bekannt gewordenen Protokolle (logische Netzwerke) umfassen eine vollständige Familie von Funktionen für die Datenkommunikation zwischen inkompatiblen Rechnersystemen (nicht–UNIX–Maschinen). Als Ziele der TCP/IP–Architektur sind definiert:

- Unabhängigkeit von der verwendeten Netzwerk–Technologie und der Architektur der Hostrechner
- Universelle Verbindungsmöglichkeiten im gesamten Netzwerk
- Standarisierte Anwendungsprotokolle

Die Popularität der TCP/IP–Protokolle ist nicht zuletzt darauf zurückzuführen, daß TCP/IP integraler Bestandteil der meisten UNIX–Implementierungen ist.

9.2 Rechneradressen (*Domaine–Name–Service*)

Netzwerkanwendungen verlangen, daß eine Maschine beim Namen genannt wird. Jede Maschine hat einen eindeutigen Namen, der sogenannte Domaine–Name, der folgende Form hat:

Maschine.Subdomain*1*....Subdomain*n*.Top–Level–Domain # Syntax

u29.num.math.uni-goettingen.de # Beispiel

Jedem Land, das am Domainnamen–System teilnimmt, wird für den Top–Level–Domain ein Ländercode zugewiesen (Deutschland = de).

Zu jedem Namen gibt es auch noch eine IP–Adresse (Internet–Protocol–Address). So ist der Rechner **u29.num.math.uni-goettingen.de** auch unter der IP–Adresse **134.76.80.39** ansprechbar.

9.3 ~~TELNET~~

~~TELNET~~ kann nicht mehr benutzt werden (Unsicherheiten durch Abhörschwächen)! Zum Einloggen auf den Rechnern der NAM von auerhalb sollte nur die Secure Shell (ssh) benutzt werden. ssh ersetzt ~~telnet~~, ~~rsh~~ und ~~rlogin~~. Programme einschließlich einiger Installations- und Benutzungshinweise fr PC's stehen unter: **ftp://ftp.num.math.uni-goettingen.de/pub/misc/**

9.4 ~~FTP~~

~~FTP~~ kann nicht mehr benutzt werden (Unsicherheiten durch Abhörschwächen)! Als Ersatz für ~~ftp~~ zum persönlichen Dateitransfer sollte das entsprechende Secure Copy-Programm (scp bzw. pscp) benutzt werden. Programme einschließlich einiger Installations- und Benutzungshinweise für MS-Windows-Versionen stehen unter: **ftp://ftp.num.math.uni-goettingen.de/pub/misc/**

9.5 PING

PING ist ein Kommando zum Überprüfen, ob ein Rechner noch "lebt". PING sendet einen Echo–Request an einen Rechner, der, sofern die TCP/IP–Software läuft (Verbindung besteht), antwortet.

ping Zielrechnername oder **IP–Adresse** # Syntax

/usr/sbin/ping login.gwdg.de # Beispiel

9.6 ~~R–Kommunikationsprogramme~~

Die R–Kommunikationsprogramme können bis auf das Kommando **rwho** wegen Abhörschwächen nicht mehr benutzt werden. Sie sind durch die S–Kommunikationsprogramme ersetzt worden.
(siehe unten)

9.6.1 rwho

rwho (Remote–Who) Dieser Dienst gibt die 'netzweit' gerade am Rechenbetrieb teilnehmenden Benutzer aus. Wird aber wegen der großen Netzbelastung oft nur auf das lokale Netz beschränkt!

rwho # Syntax

9.7 S–Kommunikationsprogramme

S–Kommunikationsprogramme sind Nachfolgeprogramme für die R–Kommunikationsprogramme. Die Unsicherheiten (Abhörschwächen) wurden hier beseitigt.

9.7.1 ssh

Mit *secure shell* wird auf einem fernen Rechner eine Shell gestartet, der beim Aufruf ein fern auszuführendes Kommando mitgegeben wird.

ssh u29 who # Beispiel

9.7.2 slogin

Slogin ist ein *secure login* auf einem fernen Rechner. *Secure login* ist eine abhörsichere Verbindung zwischen zwei Rechnern.

slogin u33 # Beispiel

9.7.3 scp

Scp ist ein *secure copy* zwischen zwei fernen Rechnern. *Secure copy* ist abhörsicher und benutzt die selbe Authentifizierung und die selben Sicherheitsaspekte wie **ssh**.

1) Kopieren einer Datei vom Windows-Rechner zum entfernten NAM-System:
   scp datei1.typ username@u01.num.math.uni-goettingen.de:datei2.typ # Beispiel

2) Kopieren einer Datei vom NAM-System zum Windows-PC:
   scp username@u01.mum.math.uni-goettingen.de:datei.typ . # Beispiel

9.8 Elektronische Post

Das Netz ermöglicht das Versenden und Empfangen elektronischer Post (E–Mail). Um eine Nachricht zu versenden, braucht man die Adresse der Person, an die die Nachricht verschickt werden soll. Wenn diese Person Zugang zum INTERNET hat, sieht eine Adresse folgendermaßen aus:

name@domainname # Syntax

Benutzername@math.uni-goettingen.de # Beispiel

name ist der Name der Mailbox, das ist so etwas wie ein Briefkasten. Auf UNIX–Systemen wird dafür gewöhnlich der Loginname des Benutzers werwendet.

domainname ist die Gebietsanschrift (Adresse), über den der Briefkasten erreicht werden kann.

Für eine abgeschickte E–Mail gibt es **keine** Benachrichtigung, ob die Nachricht korrekt angekommen ist. Allerdings gibt es eine Information, falls die Nachricht nicht zugestellt werden konnte.
Bei UNIX kann der Briefkasten–Dienst so eingestellt werden, daß der Empfänger auf eine eingetroffene Nachricht aufmerksam gemacht wird.

9.8.1 write

write ist die einfachste Form, Nachrichten zeilenweise an einen **gerade aktiven Benutzer** (auch sich selbst) direkt zu senden.

write *Benutzername* # Syntax: Nachricht senden
Zeilenweise Eingabe der Nachricht <*RETURN*>
...
CTRL–d (*Mailbox–Service beenden*)

Eine mit dem **write**–Kommando übertragene Nachricht gelangt beim Empfänger **sofort und ohne Warnung** auf den Bildschirm. Der Empfang von Nachrichten kann unterdrückt oder zugelassen werden. Das Kommando **mesg** wirkt nur bei **write** und **talk**. (**talk:** siehe Manual Pages)

mesg n # Nachricht unterdrücken

mesg y (Standard) # Nachricht empfangen

Nachrichten an einen durch **mesg n** nicht empfangsbereiten Teilnehmer gehen verloren! Deshalb empfiehlt es sich, eine Nachricht in einer Datei abzulegen, um sie wiederholt abschicken zu können.

write *Empfänger* < **brief.txt** # Beispiel: Nachricht über Datei

9.8.2 mail

Mail ist ein einfacher Briefkasten–Dienst, der nur elementare Möglichkeiten zum Editieren beim Schreiben einer Nachricht zur Verfügung stellt.

mail *Benutzername [Benutzername]* # Syntax: Nachricht senden
Zeilenweise Eingabe der Nachricht <*RETURN*>
...
CTRL–d (*Mailbox–Service beenden*)

mail # Syntax: Nachricht empfangen
? (*Ausgabe von Informationen über* **mail**–*Kommandos*) <*RETURN*>
CTRL–d (*Mailbox–Service beenden*)

9.8.3 dxmail

Ein wesentlich komfortabler Briefpost–Dienst ist **dxmail**, der menügesteuert in einem eigenen Fenster arbeitet. Der Aufruf kann auf zwei Arten erfolgen. Einmal durch Eingabe des Kommandos **dxmail** oder im Session–Manager–Fenster unter **Applications** mit der linken Maustaste **Mail** anwählen. Die Menühandhabung von **dxmail** erklärt sich selbst.

9.8.4 rmail (Briefpost–Dienst im EMACS–Editor)

Der **emacs**–Editor stellt auch einen Briefpost–Dienst zur Verfügung, der im **emacs**–Fenster arbeitet. Die Handhabung erfolgt menügesteuert oder über Kommandoeingabe im Minipuffer. Beim Aufruf von **rmail** werden alle **emacs**–Kommandos abgestellt und nur die **rmail**–Kommandos sind verfügbar. Eine Übersicht der **rmail**–Kommandos können mit dem Kommando `C-h m` vorgespielt werden. Startkommando lautet:

M–x rmail # Aufruf

9.8.5 Netscape–Mailbox

Netscape bietet über die Mailbox einen Briefkastendienst mit vielen Funktionen an: Senden, Empfangen, Weiterleiten, Filtern, Speichern, Drucken, Verwalten verschiedener Mailordner und Adresslisten, Anhängen von Attachments an Mails.

Achtung: Bei Verwendung der Netscape–Mailbox unter UNIX muß man beachten, daß in der Standardeinstellung abgespeicherte Mails (im Mail-Ordner) von anderen Mailprogrammen (z.B. pine) nicht gefunden werden. Wenn man sich also für die Netscape–Mailbox entschieden hat, soll man **immer** bei der Netscape–Mailbox bleiben.

10 Datensicherung

Versehentliches Löschen oder auch der Ausfall einer Festplatte können zu Datenverlusten führen. Im Institut für Numerische und Angewandte Mathematik werden täglich Sicherungskopien durchgeführt. Für eine langfristige Aufbewahrung, länger als 2 Wochen, ist grundsätzlich der Benutzer selbst verantwortlich. Sicherungskopien (Backups) können auf Disketten oder Bandkassetten erfolgen. Entsprechende Geräte stehen zur Verfügung.

10.1 tar (Erzeugen von Archiv–Dateien)

tar *Schlüssel Optionen Datei–Liste* # Syntax

tar erzeugt eine Archiv–Datei oder restauriert Dateien einer Archiv–Datei. Der **Schlüssel** entscheidet, ob ein Archiv angelegt oder von einem Archiv gelesen werden soll.

Schlüssel: (unvollständiger Auszug)

- **c** Erzeuge eine Archiv–Datei
- **x** Extrahiere Dateien aus einer Achiv–Datei
- **t** Liste den Inhalt der Archiv–Datei

Optionen: (unvollständiger Auszug)

- **v** 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 nach Schlüssel). Wird anstelle eines Dateinamens ein Minuszeichen angegeben, so wird Standard–Ausgabe oder Standard–Eingabe verwendet (Beispiel dafür bei **compress**).

Beispiele:

`tar -cvf buch.tar buch`	Erzeugt ein Archiv des Unterverzeichnisses `'buch'` aus dem Heimatverzeichnis heraus (falls `'buch'` ein Verzeichnis ist, sonst wird die Datei `'buch'` archiviert).
`tar -xvf buch.tar`	Restauriert der Inhalt des Archives `'buch.tar'` wieder in seine ursprüngliche Form.

10.2 Komprimieren und Entkomprimieren von Dateien

Die folgenden Kommandos dienen dazu, Dateien zu komprimieren und wieder aus dem komprimierten Zustand zu restaurieren. Sie werden folgendermaßen aufgerufen:

compress *dateiname* # Syntax (komprimieren)
uncompress *dateiname*.**Z** # Syntax (entkomprimieren)
zcat *dateiname*.**Z** **oder** *dateiname*.**gz** # Syntax (entkomprimieren)
gzip *dateiname* # Syntax (komprimieren)
gunzip *dateiname*.**z** **oder** *dateiname*.**gz** # Syntax (entkomprimieren)

compress reduziert die Größe der Datei `'dateiname'` und speichert das komprimierte Resultat in der Datei `'Dateiname.Z'` ab.

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

zcat stellt auch die komprimierte Datei `'Dateiname.Z'` in ihrer ursprünglichen Form wieder her, aber die komprimierte Datei `'Dateiname.Z'` bleibt erhalten. Die Ausgabe erfolgt bei **zcat** in Standard–Ausgabe.

tar cvf – buch | compress > buch.tar.Z # Beispiel

Es wird eine Datei `'buch.tar.Z'` erstellt. Das Minuszeichen im **tar**–Kommando veranlaßt **tar** dazu, die Ausgabe nach Standard–Ausgabe zu schreiben. Standard–Ausgabe wird mit Hilfe der Pipe als Standard–Eingabe in das Kommando **compress** umgeleitet. **compress** leitet die Ausgabe in die gewünsschte Datei `'buch.tar.Z'` um.

zcat buch.tar.Z | tar xvf – # Beispiel

Diese Kommandozeile stellt den Inhalt der komprimierten Archiv–Datei wieder her. Das Minuszeichen informiert **tar**, daß die Eingabe über Standard–Eingabe erfolgt (über die Pipe).

Wesentlich effizienter als das vorgestellte **compress** und **uncompress** sind die Programme **gzip** und **gunzip**. Die Benutzung dieser Programme wird empfohlen. Eine mit **gzip** komprimierte Datei erhält die Namensendung `'.z' ('z'` als Kleinbuchstabe!).
Hinweis: gunzip kann eine unter **compress** erstellte Datei entkomprimieren. **uncompress** kann eine unter **gzip** erstellte Datei **nicht** entkomprimieren!

10.3 Komprimieren von Dateien im Emacs–Verzeichnis–Editor (*dired*)

Der Verzeichnis–Editor (*dired*) des Emacs kennt verschiedene Kommandos (*Operating on Files*), die beim Arbeiten mit Dateien sehr hilfreich sind. Unter anderem auch ein **compress**–Kommando (*dired–do–compress*). Zum Komprimieren wird das Programm **gzip** und zum Entkomprimieren **gunzip** eingesetzt.

Der Verzeichnis–Editor wird mit `C-x d` oder `M-x dired` aufgerufen. Als Kommando–Eingabe zum Komprimieren oder zum Entkomprimieren wird der Großbuchstabe "Z" eingegeben. Die Komprimierungsprozedur erkennt selbst, ob eine Datei komprimiert oder entkomprimiert werden muß. Eine komprimierte Datei erhält die Namensendung `'.z'`.

Auswählmöglichkeiten mehrerer Dateien zum Komprimieren oder Entkomprimieren:

Über Eingabe im Mini–Buffer:

(ESC-Taste allein drücken
numerisches Argument eingeben) nur bei Erklärungspunkt 1 erforderlich (siehe unten)

Großbuchstabe "Z" eingegeben
bestätigen der Ent– oder Komprimierungsfrage im Minipuffer

Durch Markieren mit dem Zeichen '*':

Cursor an beliebiger Stelle auf entsprechende Zeile setzen
Taste Kleinbuchstabe "`m`" *(dired-mark)* betätigen
Großbuchstabe "Z" eingegeben
bestätigen der Ent– oder Komprimierungsfrage im Minipuffer

Erklärungen:

1) Wenn dem Kompresss–Kommando ein numerisches Prefix–Argument n vorangestellt wird, werden die nächsten n Dateien, beginnend bei der aktuellen Datei (bestimmt durch Cursor–Position), bearbeitet.
2) Alle "*"–markiert Dateien werden bearbeitet.
3) Ohne Prefix–Argument oder "*"–Markierung wird nur die aktuelle Datei (bestimmt durch Cursorposition) bearbeitet.

Stichwortverzeichnis

Arbeiten mit der Shell
- Aliase 20
- Allgemeines 18
- Dateien umlenken 18
- Dateinamen ergänzen (*wildcards*) 19
- Filter 19
- Kommandos verknüpfen (*pipe*) 19
- Standard–Ausgabe 18
- Standard–Eingabe 18
- Umlenkzeichen 18

Benutzung der Maus
- Doppelklick (*Double Click*) 11
- Drücken (*Press*) 11
- Klick (*Click*) 11
- Maus–Druckknöpfe (*Mouse–Buttons*) 11
- Positionieren (*Point*) 11
- Shift Klick (*Shift Click*) 11
- Ziehen (*Drag*) 11

Compiler
- Allgemeines 39
- Aufbau eines Makefiles 40
- Aufrufe automatisieren (*make*) 40
- Optionen 39

Dateien siehe UNIX –> Dateien
Dateien drucken siehe Hilfsprogramme für Textdateien
Dateien umlenken siehe Arbeiten mit der Shell
Dateiverwaltung
- Allgemein 26
- Dateien kopieren (*cp*) 27
- Dateien löschen (*rm*) 26
- Dateien umbenennen (*mv*) 27
- Dateimodus ändern (*chmod, chown, chgrp*) 28
- Verweise auf Dateien (*links*) 27
  - Hard–Link 27
  - Symbolischer–Link 28
- Verzeichnisse erzeugen (*mkdir*) 26
- Verzeichnisse löschen (*rmdir*) 26
- Zugriffsrechte unterbinden (*umask*) 30

Datensicherung
- Allgemein 45
- Archiv–Dateien 45
- Ent–/Komprimieren im Emacs 46
- Entkomprimieren 45
- Komprimieren 45

Emacs
- Beenden 13
- Dateien laden 13
- Fehlerbehandlung 13
- Fensterverwaltung 16
- Großschreibung ändern 16
- Hilfe 17
- Kleinschreibung ändern 16
- Kommandos
  - Eingabe 12
  - Übergreifende 12
- Markieren 14
- Minipuffer 16
  - Kommandoeingbe 17
  - Kommandoliste (*history list*) 17
  - Kommandos vervollständigen 17
  - Kommandowiederholungen 17
  - Tastenbelegungen 16
- Modi (*major modes*) 12
- Positionieren 14
  - Mit Cursorfunktionen 14
  - Über Bildschirm blättern 14
- Puffer (*buffer*) 16
- Starten 13
- Verzeichniseditor (*dired*) 13
- Zeichen
  - Einsetzen 14
  - Löschen 14
  - Vertauschen 16
- Zeichenketten
  - Ersetzen 15
  - Inkrementelles Suchen 15
  - Nichtinkrementelles Suchen 15
  - Suchen aller Vorkommenden 15

Hilfsprogramme für Textdateien
- Ausgabe verdoppeln (*tee*) 38
- Bildschirmausgabe seitenweise (*more*) 31
- Dateien betiteln, formatieren (*pr*) 31
- Dateien drucken (*lpr*)
  - Allgemeines, Druckkommandos 32
  - Druckernamen, Druckerstandorte 33
  - Druckerwarteschlangen abfragen 33

Dateien vergleichen (*diff*) 34
Dateien verknüpfen und drucken (*cat*) 31
Dateiende ausdrucken (*tail*) 38
Dateiinhalte sortieren (*sort*) 35
Dateiinhalte vermischen (*sort*) 35
Textmuster in Dateien suchen (*grep*) 36
Verzeichnisse vergleichen (*diff*) 34
Zeilen, Worte, Zeichen zählen (*wc*) 34
Zeilenteile ausschneiden (*cut*) 37
Zeilenteile zusammenfügen (*paste*) 37

Hilfsprogramme unter UNIX
Aktuelles Verzeichnis einsehen (*pwd*) 21
Benutzer auflisten (*who*) 23
Dateiattribute bestimmen (*file*) 22
Dateinamen suchen (*find*) 24
Information suchen (*apropos*) 21
Informationen vorspielen (*man*) 21
Kommandokurzbeschreibung (*whatis*) 21
Passwort ändern (*yppasswd*) 24
Prozesse abbrechen (*kill*) 23
Prozesse anzeigen (*ps*) 23
Verzeichnis auflistenl (*ls*) 22
Verzeichnis wechseln (*cd*) 21

Login siehe UNIX –> Anmelden
Logout siehe UNIX –> Abmelden

Netzwerk
Allgemeines 41
Elektronische Post (*E–Mail*)
Adressenaufbau 43
Dxmail 44
Mail 43
Netscape–Mailbox 44
Rmail (Emacs–Briefdienst) 44
Write 43
~~FTP~~ (*File Transfer Protocol*) 41
PING (Netzverbindung prüfen) 42
R–Kommunikationsprogramme
Allgemein 42
rwho (*Remote–Who*) 42
Rechneradressen
Domaine Name Service 41
Internet Protocol Adress 41
S–Kommunikationsprogramme
scp (*Secure–Copy*) 42
slogin (*Secure–Login*) 42
ssh (*Secure–Shell*) 42
TCP/IP–Protokoll 41
~~TELNET~~ (*Remote Login*) 41

Pause siehe UNIX –> Abmelden

UNIX
Abmelden im UNIX-System 2
Allgemeines 1
Anmelden im UNIX-System 1
Bildschirm einschalten 1
Bildschirm Power Save 1
Bildschirm–Schoner 1
Dateien
Allgemeines 3
Benutzerarten 6
Dateiarten 5
Dateien erzeugen 5
Dateinamen 4
Dateinamenserweiterung 4
Drucken siehe Hilfsprogramme für Textdateien
Schutzcode 6
Versteckte Dateien (Punkt–Dateien) 4
Zugriffsarten 6
Zugriffsrechte 6
Dateisystem
Hierarchisch 7
Pfadnamen 8
Punkt-Punkt-Verzeichnisse 8
Verzeichnisse (Directories) 8
Drucker Ein– Ausschalten 1
Drucker Papierstau 1
Kern (Betriebssystem) 2
Kommandointerpreter (*Shell*) 3
Kommandos
Allgemein 9
Einfache 9
Gruppen 10
Trenner 10
Zusammengesetzte 9
Prozesse
Allgemeines 3
Hintergrundprozesse 3
Prozesse abbrechen siehe Hilfsprogramme unter UNIX
Prozesse anzeigen siehe Hilfsprogramme unter UNIX
Vordergrundprozesse 3
Umgang mit Maschinen 1
UNIX 1