PC-Pool der Physik Mittelerde

Klaus Froböse

Universität Konstanz
Fachbereich Physik

27. Oktober 1999

Herzlichen Dank für das Korrekturlesen und Verbesserungsvorschläge an unsere Systemverwalter Peter Wenzel Gerhardt und Guido Günther.

Kapitel 1

Willkommen im PC-Pool der Physik (Mittelerde)

1.1 Beginn einer Sitzung (Einloggen)

Nachdem Sie eine Benutzerkennung (UserId) und ein Paßwort zugeteilt bekommen haben[1], können Sie zu Ihrer ersten Sitzung im PC-Pool starten.

Einen freien Rechner erkennen Sie an dem charakteristischen Bild von Gandalf, dem Zauberer. Neben ihm können Sie Ihre Benutzerkennung, Ihr Paßwort und Ihre Lieblings-Graphik-Oberfläche angeben. Durch Drücken der [↩]-Taste erscheint die von Ihnen gewünschte graphische Oberfläche mit einem oder mehreren geöffneten Fenstern, so daß Sie sofort arbeiten können. Das Erscheinungsbild ist von der Systemverwaltung vorgegeben. Sie können es aber entsprechend Ihren persönlichen Vorstellungen nach Belieben abändern, indem Sie entsprechende Konfiguratiosdateien anlegen.

Sie können sich auch im Textmodus einloggen. Dazu müssen Sie vor dem Einloggen eine der möglichen 6 Textkonsolen mit der Tastenkombination [Strg]+[Alt]+[F<*n*>] aus der Graphik-Umgebung oder mit [Alt]+[F<*n*>] aus einer Text-Umgebung anwählen, wobei <*n*> eine Zahl zwischen 1 und 6 ist für eine andere Text-Konsole oder 7 für die Graphik-Konsole.

Ganz gleich, auf welchem Rechner des PC-Pools Sie sich einloggen, Sie finden immer dasselbe Erscheinungsbild vor. Das liegt daran, daß alle Daten zentral auf dem Server `gandalf` gehalten werden. `gandalf` ist für die normalen Benutzer gesperrt, nur die Systemverwalter können sich einloggen.

Wenn Sie sich zum erstenmal einloggen, ist jetzt vielleicht der geeignete Zeitpunkt, ein eigenes Paßwort zu vereinbaren. Dazu führen Sie den Befehl

```
passwd
```

aus. Wählen Sie bitte ein möglichst sicheres Paßwort. Ein leicht erratbares Paßwort gefährdet nicht nur Ihre eigenen Datenbesände, sondern macht es einem Eindringling auch einfacher, das System zu verändern, wenn es ihm gelingt, sich für Sie auszugeben. Es kommt ziemlich häufig vor, daß versucht wird, von außen in unser System einzudringen. Bis jetzt haben unsere Sicherheitsmechanismen all diesen Versuchen standgehalten. Das sollte für Sie kein Signal sein, es zu versuchen. Solche Versuche werden mit Ausschluß von der Benutzung geahndet.

Ein Paßwort sollte aus mindestens 6 Zeichen bestehen, von denen mindestens eins ein Ziffernzeichen sein sollte. Besser noch sind Sonderzeichen. Allerdings sollte das Paßwort nicht zuviel Fingerakrobatik erfordern, damit man es nicht schon erraten kann, wenn Sie es eintippen. Weiter sollte es keine Begriffe enthalten, die man in einem Lexikon finden kann. Je „zufälliger“ die Buchstabenkombination, desto besser.

[1] Wie man beide erhält, können Sie den Aushängen entnehmen.

1.2 Benutzung

1.2.1 Multi-User-System

Bedenken Sie, daß Sie an einem Multi-User-System teilhaben. Während Sie an der Konsole sitzen, können andere Benutzer sich über das Netz in den Rechner eingeloggt haben, so daß alles, was Sie treiben, andere stören kann. Nach der Benutzungsordnung sind Sie verpflichtet, diese Störungen möglichst gering zu halten.
Alle Aktionen auf den Rechnern werden protokolliert, so daß sich Betriebsstörungen nachträglich rekonstruieren lassen.

1.2.2 Plattenspeicher

Ihnen wird Speicherplatz auf einer Festplatte als Home-Verzeichnis zur Verfügung gestellt. Da der Gesamtspeicherplatz nur endlich ist und sich auf viele Benutzer verteilt, kann für den einzelnen nicht so furchtbar viel übrigbleiben. Haben Sie kurzfristig z. B. zum Entpacken großer Archive einen höheren Speicherbedarf, so können Sie auf die Scratch-Bereiche der Rechner zugreifen. Diese sind erreichbar unter

`/scratch/`<*rechnername*>

wobei für <*rechnername*> der Name eines Rechners des PC-Pools einzusetzen ist. Will man nur den Scratchbereich des Rechners ansprechen, auf dem die aktuelle Shell läuft, so kann man das Verzeichnis auch über den Link (⟹ Unterabschnitt 2.5.9 auf Seite 11)

`/var/scratch`

ansprechen.
Scratchdateien werden automatisch gelöscht, wenn sie ein bestimmtes Alter erreicht haben (⟹ Aushang), bei akutem Bedarf an Scratchspeicherplatz auch nach kürzerer Zeit. Nehmen Sie Rücksicht auf andere und löschen Sie nicht mehr benötigte Daten so schnell wie möglich.

1.2.3 Drucken

Drucken kostet Geld. Deshalb können nur unerhebliche Mengen an bedrucktem Papier kostenfrei sein. Was als unerheblich gilt, und die aktuellen Tagespreise entnehmen Sie bitte dem Aushang.
Die Druckbefehle sind in Unterabschnitt 2.7.7 auf Seite 17 beschrieben.

1.3 Ende einer Sitzung (Ausloggen)

Vergessen Sie nicht, Ihre Sitzung ordnungsgemäß zu schließen, d. h. in einer Graphik-Umgebung die vorgesehenen Schaltfläche zu benutzen, in einer Sitzung an einer Text-Konsole den Befehl `exit` oder die Tastenkombination Strg + D einzugeben.

Die Rechner bitte unter keinen Umständen ausschalten oder zurücksetzen!

Kapitel 2

UNIX

UNIX ist ein sehr mächtiges Betriebssystem, mit dem man sehr leicht große Datenbestände verwalten kann. Um gleich einem Irrtum vorzubeugen: Ein einheitliches UNIX gibt es nicht. Jede Implementation sieht ein wenig anders aus.

2.1 Befehlsinterpreter

Obwohl unter DOS die Verwendung verschiedener Befehlsinterpreter möglich ist, wird praktisch nur der Befehlsinterpreter `COMMAND.COM` verwendet. Unter UNIX ist es dagegen üblich, daß jeder Benutzer einen Befehlsinterpreter seiner Wahl verwendet. Ich halte die BASH (*bourne-again-shell*) für die komfortabelste und flexibelste. Wir werden deshalb hier nur die BASH besprechen.
Wenn der Befehl

```
echo $BASH
```

einen vollständigen Dateinamen (einschließlich Pfad) anzeigt, bedeutet das, daß die BASH aktiv ist. Wird dagegen nur eine leere Zeile ausgegeben, muß die BASH erst mit dem Befehl

```
bash
```

aufgerufen werden.

2.2 Hilfe

Die meisten UNIX-Befehle haben eine eingebaute Hilfsfunktion. Diese kann man i. a. mit

<*befehl*> `--help`

aktivieren. Bei „älteren“ Befehlen muß man statt `--help` oft `-h` oder `-?` eingeben.
Zu vielen UNIX-Befehlen gibt es Manual-Seiten („man pages“), in denen die genaue Funktion des Befehls zusammen mit der Aufrufsyntax ausführlich erklärt sind.

```
man          <befehl>
```

zeigt eine solche Seite an.

```
man -k       <stichwort>
```

zeigt eine Liste von Befehlen, in deren Beschreibung das Wort <*stichwort*> auftaucht (engl.: *keyword*). Das hilft weiter, wenn der Name des Befehls nicht bekannt ist.
Falls die Ausgabe zu lang wird, kann man sie mit dem Filter `less` (⟹ Unterabschnitt 2.7.2 auf Seite 13) seitenweise ausgeben lassen.
Wesentlich umfangreichere Hilfen bekommt man mit dem Befehl

`info` [<*befehl*>]

Nach Eingabe dieses Befehls ohne <*befehl*> wird eine Liste der im Info-System behandelten UNIX-Kommandos angezeigt. In diesem Menü kann mit Hilfe von Tastendrücken (Cursor-Tasten, ↩-Taste, Zeichen-Tasten (`n` = *next*, `p` = *prev*, `u` = *up*)) navigiert werden und die gesuchte Information ausgewählt werden. Querverweisen kann man mit der ↩-Taste direkt folgen.
Wird <*befehl*> angegeben, wird die Hilfe zu <*befehl*> sofort angezeigt.
Das Info-System kann auch direkt aus dem Editor EMACS (⟹ Unterabschnitt 2.7.6 auf Seite 15) aufgerufen werden.

2.3 Notation

Zur Beschreibung der Syntax von Befehlen hat sich folgende Notation eingebürgert:

- **Ausdrücke in eckigen Klammern** ([<*file*>]) dürfen weggelassen werden. Das Programm setzt dann von sich aus einen voreingestellten Wert (engl.: *Default*) ein.
- **Ausdrücke in spitzen Klammern** (<*beispiel*>) müssen vom Benutzer spezifiziert werden.
- Eine **Reihe von Punkten** (. . .) deutet die mögliche Wiederholung des Vorhergehenden an.
- Eine Menge, aus der **genau eine Möglichkeit** ausgewählt werden muß, wird durch `[ a | b | c | . . . ]` bezeichnet.
- **Alles andere muß buchstabengetreu übernommen werden.**

2.4 Befehle

UNIX-Kommandos oder -Befehle gehorchen einer bestimmten Syntax. Am Anfang des Kommandos steht sein Name. Dieser ist meistens eine Abkürzung für das, was der Befehl bewirken soll. Die Wirkung eines Befehls kann durch sogenannte *Optionen* modifiziert werden. Unter UNIX werden viele Optionen durch einen einzelnen Buchstaben beschrieben, dem ein `-`-Zeichen vorangestellt wird. Es gibt aber auch die Form, wo einem solchen Optionsbuchstaben (oder auch Optionswort, das ebenfalls durch ein `-`-Zeichen gekennzeichnet wird) ein Wert folgen muß. In vielen Fällen lassen sich mehrere solcher Optionsbuchstaben zu einer Buchstabenkette (String) zusammenfassen, der ein einziges `-`-Zeichen vorausgeht. Eine solche Kette wird im folgenden durch [<*optionenstring*>] bezeichnet.

`name` [<*optionenstring*>] [<*opt1*> <*wert1*> . . .][<*parameter1*> . . .]

Neuere Befehle lassen auch die Angabe von Optionen im Klartext zu. In diesem Fall stehen vor der Option häufig zwei `-`-Zeichen (`--help`).
Auf die Optionen folgen die Parameter, in der Regel die Namen der Dateien, auf die der Befehl angewendet werden soll.
Jeder Befehl liefert ein Ergebnis über seinen Erfolg oder Mißerfolg an das Betriebssystem, *Fehler-Code* zurück.
Nach Konvention ist der Fehler-Code `0`, wenn der Befehl fehlerlos abgearbeitet werden konnte. In diesem Fall gilt das Ergebnis als `true`.
Ein Ergebnis, das von `0` verschieden ist, deutet auf einen Fehler oder eine Ausnahmesituation hin. In diesem Fall gilt das Ergebnis als `false`.

Ein Ausrufungszeichen vor dem Befehl kehrt `true` und `false` um: Aus `0` wird `1`, aus jedem anderen Wert `0`.
Den Wert des Fehler-Codes des zuletzt ausgeführten Kommandos kann man mit `$?` abfragen und z. B. mit `echo $?` anzeigen lassen.
Meistens gibt es zu dem Befehl (`man`-page) eine Liste mit der Bedeutung der Fehler-Codes.

2.5 Dateisystem

Alle Dateien sind in einem einheitlichen systemglobalen Verzeichnisbaum eingeordnet. Unterschiedliche Speichermedien erscheinen hier lediglich als verschiedene Unterverzeichnisse in dem Verzeichnisbaum.

`mount`	(ohne Parameterangabe)

zeigt alle in das Dateisystem integrierte Speichermedien an, zusammen mit den Namen der Unterverzeichnisse, unter denen sie angesprochen werden. Die Speichermedien werden vom System verwaltet als Dateien des Verzeichnisses `/dev`.

`pwd`

zeigt das aktuelle Verzeichnis an (engl.: *print working directory*).
Der Systemverwalter ordnet jedem Benutzer ein sogenanntes Home-Verzeichnis zu. Mit dem Befehl

```
echo $HOME
```

erfährt man seinen Namen und mit dem Befehl

`cd`

(ohne Parameter) wechselt man hinein.

`cd`	<*pfad*>

ersetzt das aktuelle Verzeichnis durch das durch <*pfad*> bezeichnete Verzeichnis. Es sind absolute (beginnend mit `/`) und relative Pfadangaben erlaubt. Relative Pfadangaben dürfen auch aufsteigen, wenn sie mit `../` (⟹ Unterabschnitt 2.5.1) beginnen.

2.5.1 Datei- und Verzeichnisnamen

Das Trennzeichen für Verzeichnisnamen in Pfadangaben ist `/` (Schrägstrich).
Datei- und Verzeichnisnamen dürfen unter UNIX beliebig lang sein. Groß- und Kleischreibung werden unterschieden. Fast alle Zeichen dürfen für Datei- und Verzeichnisnamen verwendet werden. Es empfiehlt sich aber, von dieser Möglichkeit nur wenig Gebrauch zu machen und nur druckbare Zeichen zu verwenden. Aus der Menge der druckbaren Zeichen sollte man auch solche vermeiden, die für das Betriebssystem eine syntaktische Bedeutung haben.
Enthält der Dateiname Punkte `.`, so nennt man den Teil des Namens nach dem letzten Punkt *Suffix*, *(Datei-)Namenerweiterung* oder *Dateiendung*. Konventionsgemäß weist das Suffix auf den Inhalt der Datei hin (Beispiele in Tabelle 2.1 auf der nächsten Seite).
Der Name `.` (ein Punkt) steht standardmäßig für das aktuelle Verzeichnis. Der Name `..` (zwei Punkte) bezeichnet das dem aktuellen Verzeichnis übergeordnete.
Eine Sonderstellung nehmen die Datei- und Verzeichnisnamen ein, die mit einem Punkt (.) beginnen. Solche Dateinamen werden normalerweise von Dateimasken (⟹ Unterabschnitt 2.5.2 auf der nächsten Seite) nicht abgedeckt und daher nicht bearbeitet. Man findet Dateien mit solchen Namen hauptsächlich in den Home-Verzeichnissen der Benutzer. Sie enthalten meistens persönliche Konfigurationsdaten des

Suffix	Inhalt
`c`	C-Quelle
`h`	C-Header-Datei
`cc`	C++-Quelle (auch Suffix `C` oder `CPP` unter DOS)
`f`	FORTRAN-Quelle (auch Suffix `f90` oder `f77`
`tex`	TEX- oder LATEX-Quelle
`dvi`	von TEX- oder LATEX erzeugte Ausgabedatei
`ps`	Druckdatei im PostScript-Format
`gz`	mit `gzip` komprimierte Datei
`tar`	Archivdatei (kann ganze Verzeichnisbäume enthalten)
`tgz`	mit `gzip -z` komprimierte Datei (auch `tar.gz`)
`Z`	mit `compress` komprimierte Datei

Tabelle 2.1: Beispiele für Dateiendungen

jeweiligen Benutzers zu bestimmten Programmen. In solchen Fällen besteht der Name aus einem Punkt, gefolgt von dem Namen des entsprechenden Programms (z. B. `.emacs`, `.gnuplot`).
Das Home-Verzeichnis des aktuellen Benutzers kann in Befehlen durch `~` abgekürzt werden, das Home-Verzeichnis des Benutzers <*benutzer*> durch `~`<*benutzer*>, vorausgesetzt der Benutzer <*benutzer*> hat externen Zugriff ($\Longrightarrow$ Unterabschnitt 2.5.4 auf Seite 8) auf seine Dateien zugelassen. Mit

```
~froboese
```

findet man z. B. die Dateien in meinem Home-Verzeichnis.

2.5.2 Datei- und Verzeichnisgruppen

Mit speziellen Zeichen in Datei- und Verzeichnisbezeichnern kann man Datei- und Verzeichnisgruppen ansprechen:

`*` steht für eine Folge von beliebigen Zeichen, die Folge darf auch leer sein.

`?` steht für ein einzelnes beliebiges Zeichen.

`[`<*liste*>`]` steht für ein einzelnes Zeichen aus <*liste*>.

`[^`<*liste*>`]` steht für ein einzelnes Zeichen, das nicht aus <*liste*> ist.

<*liste*> steht für eine Folge von Zeichen. In einer solchen Liste sind außerdem Bereichsangeben der Form <*anfang*>`-`<*ende*> wie z. B. `0-9` für eine beliebige Ziffer erlaubt. Die Reihenfolge der Angaben ist beliebig, jedoch muß bei Bereichsangaben das Zeichen <*anfang*> einen kleineren Code haben als das Zeichen <*ende*>.

Das Zeichen `^` hat seine Sonderbedeutung nur, wenn es als erstes Zeichen auf die öffnende Klammer folgt, sonst stellt es sich selbst dar.

Das Zeichen `-` hat seine Sonderbedeutung für die Bereichsangabe nur, wenn es zwischen zwei Zeichen steht, am Anfang oder Ende von <*liste*> stellt es sich selbst dar.

Das Zeichen `]` schließt die Liste nur ab, wenn es nicht unmittelbar auf die öffnende Klammer folgt. Unmittelbar hinter der öffnenden Klammer stellt es sich selbst dar.

Die Angabe `[XA-CY]*-??[^0-9]` bezeichet alle Dateien bzw. Verzeichnisse, deren Namen mit `A, B, C, X` oder `Y` beginnen und nicht auf eine Ziffer enden. Außerdem muß das drittletzte Zeichen im Dateinamen ein `-`-zeichen sein.
Ein Punkt am Anfang eines Dateinamens muß immer explizit angegeben werden.
Die Masken schließen niemals den trennenden Schrägstrich zwischen Verzeichnisnamen ein, so daß in einem längeren Pfad für jede Verzeichnisebene ein eigener Namensteil angegegeben werden muß.

2.5.3 Anzeigen des Inhalts von Verzeichnissen

`ls` [<*optionenstring*>] [<*maske*>]

zeigt die Namen der Dateien und Verzeichnisse des aktuellen Verzeichnisses an, wenn <*maske*> nicht angegeben wurde, sonst die Namen der Dateien und Verzeichnisse, auf die <*maske*> paßt. Paßt <*maske*> auf den Namen eines Verzeichnisses, so wird dessen Inhalt auch ausgegeben. Die Ausgabe erfolgt alphabetisch geordnet in mehreren Spalten. Dateien, deren Name mit . beginnt, werden nur bei Angabe einer der Optionen `-a` oder `-A` angezeigt oder bei expliziter Angabe des Punktes in der Maske.

`-l` gibt vollständige Information über die Dateien (jeweils eine Zeile pro Datei engl.: *long*), nämlich

- Art und Zugriffsrechte (⟹ Unterabschnitt 2.5.4 auf der nächsten Seite)
- Anzahl der Hardlinks (⟹ Unterabschnitt 2.5.9 auf Seite 11)
- Name des Eigentümers (Owner)
- Name der Gruppe, die Zugriffsrechte hat
- Größe der Datei in Bytes
- Datum (und Uhrzeit) der letzten Änderung[1]
- Name der Datei

`-a` Dateien, deren Namen mit . beginnt, werden auch angezeigt (engl.: *all*).

`-A` wie `a`, jedoch werden die Dateien . und .. nicht angezeigt (engl.: *All*).

`-F` bei den Namen von nicht normalen Dateien wird ein zusätzliches Zeichen angehängt:

- `/` bei Verzeichnisnamen,
- `*` bei ausführbaren Dateien,
- `@` bei Softlinks (⟹ Unterabschnitt 2.5.9 auf Seite 11).

`-d` unterdrückt das Auflisten des Inhalts von Verzeichnissen. (engl.: *directory*).

`-R` zeigt rekursiv auch die Dateien aller Unterverzeichnisse an (engl.: *recursive*).

`-1` bewirkt einspaltigen Ausdruck.

`-I` zeigt nur die Dateien an, auf die <*maske*> nicht paßt. <*maske*> muß maskiert werden (⟹ Unterabschnitt 2.10.1.1 auf Seite 21) (engl.: *ignore*).

Es gibt noch eine ganze Reihe von Optionen, z. B. um die Ausgabe nach anderen Kriterien zu sortieren. Die Bedeutung der Optionsbuchstaben kann von System zu System variieren.

Falls die Ausgabe zu lang wird, kann man sie mit dem Filter `less` (⟹ Unterabschnitt 2.7.2 auf Seite 13) seitenweise ausgeben lassen.

[1] Zu jeder Datei gibt es drei verschiedene Zeitpunkte: Zeitpunkt der letzten Änderung des Inhalt, Zeitpunkt der letzten Änderung des Status, Zeitpunkt des letzten Zugriffs. Alle drei können durch entsprechende Optionen sichtbar gemacht werden.

2.5.4 Zugriffsrechte auf Dateien und Verzeichnisse

Die lange Ausgabe des `ls`-Kommandos (`ls -l`) erzeugt in der ersten Spalte eine Kette von Zeichen der Gestalt

<*c*>`rwxrwxrwx`.

Der erste Buchstabe <*c*>[2] ist

- `-` für normale Dateien,
- `d` für Verzeichnisse,
- `l` für Softlinks (⟹ Unterabschnitt 2.5.9 auf Seite 11).

Die nächsten Zeichen bilden Dreiergruppen, die sich nacheinander auf den Eigentümer (user), die Gruppe (group) und den Rest der Welt (other) beziehen. Ein `r` räumt Leserechte (engl.: *read*), ein `w` räumt Schreibrechte (engl.: *write*) zum Verändern des Inhalts der Datei und ein `x` räumt Ausführungsrechte (engl.: *execute*) für die betreffende Datei ein. Besteht ein Recht nicht, so steht anstelle des Buchstabens ein `-`[3].
Ausführungsrechte für Dateien bedeuten, daß die Datei wie ein normales UNIX-Programm ausgeführt werden darf.
Die Zugriffsrechte für Verzeichnisse beziehen sich auf die (i. a. vom Benutzer nicht auswertbare) Datei, in der die Information über das Verzeichnis gespeichert ist. Leserecht bedeutet, daß das Inhaltsverzeichnis ausgegeben werden kann (z. B. mit `ls`). Schreibrechte erlauben das Anlegen und Löschen(!)[4] von Dateien in dem Verzeichnis. Ausführungsrechte für Verzeichnisse bedeutet, daß mit `cd` in das Verzeichnis gewechselt werden darf.

`chmod`	`[-R]`<*wer*>`[ + | - | =]`<*was*> <*maske*>

`chmod`	`[-R]`<*oktalzahl*> <*maske*>

ändert die Zugriffsrechte von Dateien und Verzeichnissen.
In der ersten Form steht <*wer*> für eine Kombination der Buchstaben `u` (*user*), `g` (*group*), `o` (*other*) oder `a` (*all*). <*was*> bezeichnet die Rechte, die geändert werden sollen, mit den Buchstaben `r`, `w` und `x`. `+` fügt die Rechte zu den bestehenden hinzu, `-` löscht sie und `=` setzt sie genau auf diesen Wert.
In der zweiten Form werden die Rechte durch eine vierstellige Oktalzahl ausgedrückt. Hier betrachten wir nur den Fall, daß die führende Ziffer 0 ist. Die zweite Ziffer enthält den Code für „user“, die dritte den für „group“ und die letzte für „other“. Den Code erhält man, indem man für `r` 4, für `w` 2 und für `x` 1 setzt, und diese Zahlen für die zu setzenden Rechte addiert. `0751` bedeutet: alle Rechte für „user“, Lese- und Ausführungsrecht für „group“ und Ausführungsrechte für „other“.
<*maske*> bezeichnet die Dateien, deren Zugriffsrechte geändert werden dürfen. Ist die Option `-R` angegeben, werden die Rechte auch in den Unterverzeichnissen geändert.
Bei der Erzeugung von Dateien werden automatisch alle Rechte vergeben, wobei in der Regel bei Dateien, die von Editoren erzeugt werden, das Ausführungsrecht nicht gesetzt wird. Anschließend werden die mit dem Befehl `umask` definierten gelöscht.

`umask`	`[`<*oktalzahl*>`]`

ohne Parameter zeigt die aktuelle Löschmaske an, mit <*oktalzahl*> wird die Löschmaske auf <*oktalzahl*> gesetzt[5].

```
umask 027
```

legt z. B. fest, daß alle Rechte für den Benutzer erhalten bleiben, für die Gruppe das Schreibrecht gelöscht wird und für den Rest der Welt kein Recht erhalten bleibt.

[2] Andere Buchstaben können dort auch stehen, wie man im Verzeichnis `/dev` sehen kann.
[3] Auch hier können manchmal andere Buchstaben stehen.
[4] **Eine Datei kann nicht dadurch vor dem Löschen bewahrt werden, daß man das Schreibrecht für die Datei sperrt, ausschlaggebend für das Löschen sind die Schreibrechte in dem Verzeichnis, zu dem die Datei gehört.**
[5] Damit <*oktalzahl*> als Oktalzahl erkannt wird, muß <*oktalzahl*> mit 0 beginnen.

2.5.5 Kopieren von Dateien

`cp`	[<*optionenstring*>] <*maske1*> [<*maske2*>...] <*zielverzeichnis*>

kopiert die durch <*maske1*> <*maske2*>... bezeichneten Dateien in das durch <*zielverzeichnis*> bezeichnete existierende Verzeichnis (engl.: *copy*).

`cp`	[<*optionenstring*>] <*quelle*> <*zieldatei*>

kopiert die durch <*quelle*> bezeichnete Datei in die Datei <*zieldatei*>.
In beiden Fällen bleiben Zugriffsrechte und Eigentümer/Gruppe erhalten.
Existieren die Zieldateien, so werden sie überschrieben ($\Longrightarrow$ Option `-i`).
Verzeichnisse werden nicht ohne weiteres kopiert ($\Longrightarrow$ Option `-r`).
Folgende Optionen sind möglich:

- `-p` Zeitpunkte der letzten Änderung und des letzten Zugriffs werden von der Originaldatei übernommen, sonst Zeitpunkt des Kopierens (engl.: *preserve*).
- `-r` kopiert Verzeichnisse (!) rekursiv. Alle Unterverzeichnisse und deren Dateien werden übertragen (engl.: *recursive*).
- `-R` wie `-r`, jedoch werden Links aufgelöst ($\Longrightarrow$ Unterabschnitt 2.5.9 auf Seite 11).
- `-i` vor dem Überschreiben einer existierenden Datei wird der Benutzer zu einer Bestätigung aufgefordert (engl.: *interactive*).
- `-f` erzwingt Überschreiben ohne Rückfrage(engl.: *forced*).
- `-v` zeigt alle Kopiervorgänge an (engl.: *verbose*).
- `-u` kopiert nur solche Dateien, die im Ziel nicht vorhanden oder älter sind als die Quelldatei (engl.: *update*).

2.5.6 Löschen von Dateien

`rm`	[<*optionenstring*>] <*maske*>

löscht die durch <*maske*> bezeichneten Dateien (engl.: *remove*). Es gibt keine Möglichkeit, einmal gelöschte Dateien wieder nutzbar zu machen, obwohl nur die Verweise auf die Festplatte gelöscht werden, die Daten im Prinzip also noch verfügbar sind.
Verzeichnisse werden nicht ohne weiteres gelöscht ($\Longrightarrow$ Option `-r`).
Folgende Optionen sind möglich:

- `-r` löscht Verzeichnisse(!) rekursiv. Alle Unterverzeichnisse und deren Dateien werden gelöscht (engl.: *recursive*).
- `-i` vor dem Löschen einer Datei wird der Benutzer zu einer Bestätigung aufgefordert (engl.: *interactive*).
- `-f` erzwingt Löschen ohne Rückfrage(engl.: *forced*).

Dieser Befehl löscht auch Hard- und Softlinks ($\Longrightarrow$ Unterabschnitt 2.5.9 auf Seite 11).

2.5.7 Umbenennen (Verschieben) von Dateien und Verzeichnissen

`mv`	[<*optionenstring*>] <*quelldatei*> <*zieldatei*>

benennt die Datei <*quelldatei*> in <*zieldatei*> um, oder wenn man so will: verschiebt die durch <*quelldatei*> bezeichnete Datei in die Datei <*zieldatei*> (engl.: *move*).

`mv`	[<*optionenstring*>] <*maske1*> [<*maske2*>...] <*zielverzeichnis*>

verschiebt die durch <*maske1*> <*maske2*>... bezeichneten Dateien in das durch <*zielverzeichnis*> bezeichnete existierende Verzeichnis.

In beiden Fällen bleiben Zugriffsrechte und Eigentümer/Gruppe und der Zeitpunkt der letzten Änderung des Dateiinhalts (`mtime`) erhalten.

Das Verschieben ist logisch äquivalent zu einem Kopieren, bei dem nach erfolgter Kopie die Quelldateien gelöscht werden. Die Parameter haben daher dieselbe Bedeutung wie beim Kopieren (⟹ Unterabschnitt 2.5.5 auf der vorherigen Seite, die Option `-r` gibt es jedoch nicht). Liegen Quelle und Ziel auf demselben Datenträger, werden in Wirklichkeit nur die Verweise auf dem Datenträger geändert. In diesem Fall bleiben auch die `atime` und `ctime` unverändert.

Es gibt noch eine dritte Form dieses Befehls:

`mv`	[<*optionenstring*>] <*quellverzeichnis*> <*zielverzeichnis*>

benennt das Verzeichnis <*quellverzeichnis*>, in <*zielverzeichnis*> um. Das ist aber nur möglich, wenn <*zielverzeichnis*> noch nicht existiert.

2.5.8 Erzeugen und Löschen von Verzeichnissen.

`mkdir`	`[-p][-m` <*mode*>] <*pfad*>

erzeugt das Verzeichnis <*pfad*> (engl.: *make directory*). Das Verzeichnis <*pfad*> darf noch nicht existieren. Das in der Hierarchie direkt übergeordnete Verzeichnis muß existieren, falls nicht die Option `-p` angegeben wird.

Folgende Optionen sind möglich:

`-p`	erzeugt alle noch in <*pfad*> fehlenden Zwischenunterverzeichnisse (engl.: *parents*).
`-m` <*mode*>	<*mode*> legt die Rechte für das neue Verzeichnis fest. Hier ist eine Oktalzahl, wie in Unterabschnitt 2.5.4 auf Seite 8 beschrieben, einzusetzen.

`rmdir`	[-p] <*pfad*>

löscht das durch <*pfad*> bezeichnete *leere* Verzeichnis (engl.: *remove directory*).

`-p`	löscht alle in <*pfad*> enthaltenen Zwischenunterverzeichnisse, wenn sie durch das Löschen von <*pfad*> leer werden (engl.: *parents*).

Ein noch gefülltes Verzeichnis löscht man besser mit der rekursiven Form des Befehls `rm` (⟹ Unterabschnitt 2.5.6 auf der vorherigen Seite).

Auf vielen Systemen sind die beiden Befehle unter den Namen `md` bzw. `rd` verfügbar.

2.5.9 Hard- und Softlinks

Manchmal benötigt man Dateien in verschiedenen Verzeichnissen. Man könnte, wie es in einfachen Betriebssystemen üblich ist, einfach eine Kopie der Datei in den entsprechenden Verzeichnissen anlegen. Das hat den Nachteil, daß man bei Änderungen an der Datei alle Kopien finden und ändern muß, wenn es darauf ankommt, daß die Datei in allen Verzeichnissen den gleichen Inhalt haben muß.
Dieses Problem wird gelöst durch sogenannte *Links*. Dabei werden in die Verzeichnisse nur Verweise auf die Originaldatei eingetragen. Wird auf einen solchen Link zugegriffen, ist das Betriebssystem in der Lage, den Verweis aufzulösen und die Daten aus der Originaldatei zur Verfügung zu stellen.

2.5.9.1 Hardlinks

Bei normalen Dateien ist zu jedem Dateinamen irgendwo (genauer: in der Datei, die dem jeweiligen Verzeichnis entspricht,) abgespeichert, wo die Daten auf der Festplatte abgelegt sind. Hardlinks verhalten im Grunde sich nicht anders. Nur gibt es mehrere Verweise, die auf den gleichen Datenbereich der Festplatte zeigen. Die Daten stehen also genau einmal auf der Festplatte. Es wird also kein zusätzlicher Speicherplatz belegt. Hardlinks sind nur möglich innerhalb desselben Speichermediums. Die Anzahl der Verweise, die auf den gleichen Datenbereich zeigen, werden bei der langen Form des `ls`-Befehls mit ausgegeben.

2.5.9.2 Softlinks (auch: symbolische Links)

Bei Softlinks wird wirklich in dem betreffenden Verzeichnis ein Verweis auf die Originaldatei angelegt, und zwar wird der vollständige Pfad zu der Originaldatei gespeichert. Zusätzlicher Speicherbedarf entsteht nur aus der Speicherung des Pfades und ist meistens vernachlässigbar.
Softlinks sind systemweit möglich, soweit die Zugriffsrechte das gestatten.

`ln`	[<*optionenstring*>] <*maske1*> [<*maske2*>...] <*zielverzeichnis*>

linkt die durch <*maske1*> <*maske2*>... bezeichneten Dateien in das durch <*zielverzeichnis*> bezeichnete existierende Verzeichnis.

`ln`	[<*optionenstring*>] <*quelle*> <*zieldatei*>

linkt die durch <*quelle*> bezeichnete Datei auf die Datei <*ziel*> (engl.: *link*). Wie beim `mv`-Befehl (⟹ Unterabschnitt 2.5.7 auf der vorherigen Seite) gibt es eine dritte Form:

`ln`	[<*optionenstring*>] <*quellverzeichnis*> <*zielverzeichnis*>

Hierbei wird ein Link für Verzeichnisse erstellt.
Die Parameter haben dieselbe Bedeutung wie beim `cp`-Befehl (⟹ Unterabschnitt 2.5.5 auf Seite 9).
Folgende Optionen sind möglich

- `-s` legt statt eines Hardlinks einen Softlink an (engl.: *soft*).
- `-f` der Link wird auf jeden Fall angelegt (engl.: *forced*).

2.5.10 Suchen nach Dateien

`find`	<*verz*> [`-name` <*maske*>] [`-size` <*groesse*>[[c | k]]] [`-exec` <*befehl*> \;]

sucht im Verzeichnis <*verz*> und dessen Unterverzeichnissen nach Dateien, deren Eigenschaften durch die Optionen `-name` und/oder `-size` (und/oder weitere hier nicht behandelten Optionen) beschrieben werden. Wird eine solche Datei gefunden, so wird ihr Name ausgegeben oder der unter `-exec` spezifizierte Befehl <*befehl*> ausgeführt.
<*groesse*> gibt die Größe in 512-Byte-Blöcken ohne Benennung, in Bytes mit `c` oder in Kilobytes mit `k` an. Ein `+`-Zeichen vor <*groesse*> bedeutet, daß auch größere Dateien gemeint sind, ein `-`-Zeichen, daß nur kleinere Dateien gesucht sind.

2.6 Diskettenlaufwerke

Alle Rechner des PC-Pools sind mit $3\frac{1}{2}$"-Laufwerken ausgerüstet. DOS-Disketten können mit dem Programmpaket `mtools` verwaltet werden. Dateimasken sind in Befehlen, für die sie sinnvoll sind, zugelassen. Beziehen diese sich auf die Diskette, müssen sie jedoch durch Anführungszeichen maskiert (⟹ Unterabschnitt 2.10.1.1 auf Seite 21) werden. Die Masken beziehen sich, wie unter UNIX üblich, auf den ganzen Namen (`*` und nicht `*.*` steht z. B. für „alle Dateien".) und \ wird durch `/` ersetzt.
In diesem Paket stehen (neben weiteren ⟹ `man mtools` oder `info mtools`) folgende Befehle zur Verfügung

`mcd` <*verz*>	ändert das aktuelle Verzeichnis auf der Diskette zu <*verz*>.
`mcopy` <*quelle*> <*ziel*>	kopiert Dateien von der Diskette ins UNIX-System und umgekehrt. Bei Verwendung der Option `-m` bleibt das Datum der letzten Änderung erhalten, mit der Option `-t` werden während des Kopierens die Zeilenumschaltungen dem jeweiligen Dateisystem angepaßt (⟹ Abschnitt 2.7 auf der nächsten Seite).
`mdel` <*maske*>	löscht Dateien auf Diskette.
`mdeltree` <*verz*>	löscht rekursiv das Verzeichnis <*verz*> auf der Diskette.
`mdir` [<*maske*>]	zeigt den Inhalt eines Verzeichnisses an.
`mformat a:`	legt eine DOS-Dateistruktur auf der Diskette an.
`mlabel a:`	gibt der Diskette einen Namen.
`mmd` <*uverz*>	legt das Unterverzeichnis <*uverz*> auf der Diskette an.
`mrd` <*uverz*>	löscht das Unterverzeichnis <*uverz*> auf der Diskette.
`mmove` <*quelle*> <*ziel*>	verschiebt Dateien und Unterverzeichnisse.
`mren` <*alt*> <*neu*>	benennt eine existierende Datei auf der Diskette um
`mtype` <*datei*>	zeigt den Inhalt der Datei <*datei*> von der Diskette an.

2.6.1 ZIP-Laufwerke

Der Rechner `boromir` verfügt über ein 100 MB-ZIP-Laufwerk. Um dieses zu benutzen, muß man sich zuerst mit

`ssh`	boromir

auf `boromir` einloggen, eine ZIP-Diskette in das Laufwerk legen und dann mit

`mount`	/zipfat

die Diskette in das Dateisystem einfügen. Der Inhalt der ZIP-Diskette erscheint dann unter dem Verzeichnis `/zipfat`.
Vor dem Entnehmen der Diskette muß sie aus dem Dateisystem mit

`umount`	/zipfat

abgemeldet werden.

2.7 Textdateien

Dem Inhalt nach unterscheidet man Binär- und Textdateien. Bei den Binärdateien handelt es sich im wesentlichen um ausführbare Programme, wie sie beim Compilieren und Linken von Programmquellen entstehen, oder auch um spezielle Informationen, die zweckmäßigerweise (z. B. Platzbedarf) binär abgespeichert werden. Binär gespeicherte Information ist in der Regel nur mit bestimmten Programmen zugänglich.
Textdateien dagegen enthalten normalen Text, der Zeichen für Zeichen (nach dem ASCII-Code) codiert ist. Solche Textdateien sind bis auf kleine Komplikationen z. B. bei den deutschen Umlauten auf (fast) allen Systemen lesbar und eignen sich daher hervorragend zum Datenaustausch. Der Inhalt solcher Textdateien läßt sich leicht anzeigen und mit einem Editor[6] bearbeiten.
Textdateien sind in der Regel nach Zeilen orientiert, d. h. sie enthalten zwischen den Codes, welche die Zeichen des Textes repräsentieren, noch zusätzliche Codes für die Zeilenumbrüche. Unter UNIX werden Zeilenumbrüche durch den ASCII-Code 10 (LFD = Linefeed) dargestellt, unter DOS durch die Sequenz der beiden ASCII-Codes 13 (CR = carriage return) und 10 (LFD). Durch diesen Unterschied kann es manchmal zu kleineren Problemen beim Drucken kommen, die sich aber leicht beheben lassen[7].

2.7.1 Verkettung von Dateien

`cat`	<*maske1*> [<*maske2*> ...]

gibt zunächst die durch <*maske1*>, dann die durch <*maske2*> usw. beschriebenen Dateien der Reihe nach in der Regel auf dem Bildschirm aus. Durch Umleiten ($\Longrightarrow$ Unterabschnitt 2.9.1 auf Seite 20) kann man die gesamte Ausgabe in eine einzige Datei leiten bzw. einem Programm als Eingabe übergeben (engl.: *concatenate*).

2.7.2 „Blättern“ in Dateien

`less [-i]`	<*maske1*> [<*maske2*> ...]

zeigt die durch <*maske1*>, dann die durch <*maske2*> usw. beschriebenen Dateien der Reihe nach seitenweise auf dem Bildschirm an.
Mit der Option `-i` wird erreicht, daß in Suchmustern Groß- und Kleinschreibung nicht unterschieden wird. Allerdings passen Großbuchstaben in Suchmustern nur auf Großbuchstaben, unabhängig von dieser Option. Es gibt noch eine Reihe von weiteren Optionen, deren Erklärung aber den Rahmen dieser Einführung sprengen würde.
`less` ist ein interaktives Programm, das durch Tastendrücke gesteuert wird. Nur die wichtigsten sind hier aufgeführt.

`h`	Hilfe (engl.: *help*)
`f`	nächste Bildschirmseite (engl.: *forward*)
`b`	vorige Bildschirmseite (engl.: *backward*)
`d`	nächste halbe Bildschirmseite (engl.: *down*)
`g`	an den Anfang der Datei
`G`	an das Ende der Datei
`/`<*muster*>	vorwärts zum ersten Auftreten von <*muster*>
`?`<*muster*>	rückwärts zum ersten Auftreten von <*muster*>

[6] Das ist ein einfaches Textverarbeitungsprogramm.
[7] Unter der BASH mit den im PCPool verwendeten Einstellungen gibt es die Befehle `unix2dos` und `dos2unix`, welche die Umwandlung (einschließlich deutscher Umlaute) korrekt vornehmen.

`n` vorwärts zum nächsten Auftreten des zuletzt eingegebenen Musters (engl.: *next*)

`p` rückwärts zum nächsten Auftreten des zuletzt eingegebenen Musters (engl.: *previous*)

`:n` nächste Datei (engl.: *next*)

`:p` vorige Datei (engl.: *previous*)

`:x` erste Datei

<*muster*> ist ein *regulärer Ausdruck*. Reguläre Ausdrücke werden in Unterabschnitt 2.7.5 besprochen.

2.7.3 Durchsuchen von Dateien

`grep` [<*optionenstring*>] <*muster*> <*maske1*> [<*maske2*> ...]

durchsucht nacheinander die durch <*maske1*>, dann die durch <*maske2*> usw. beschriebenen Dateien nach dem regulären Ausdruck (⟹ Unterabschnitt 2.7.5) <*muster*> und gibt die Zeilen mit den Fundstellen auf dem Bildschirm aus. Wenn <*muster*> Zeichen enthält, die von der BASH falsch „verstanden“ werden könnten, muß der Ausdruck maskiert (⟹ Unterabschnitt 2.10.1.1 auf Seite 21) werden.

`-i` keine Unterscheidung von Groß- und Kleinschreibung.

`-v` Ausgabe der Zeilen, auf die <*muster*> nicht paßt.

Durch weitere Optionen kann das Verhalten von `grep` modifiziert werden. Außerdem ist es möglich, gleichzeitig nach mehreren Mustern suchen zu lassen (⟹ `man grep`).

2.7.4 Abzählen in einer Datei

`wc` [<*optionenstring*>] <*maske1*> [<*maske2*> ...]

zählt in den durch <*maske1*>, dann in den durch <*maske2*> usw. beschriebenen Dateien der Reihe nach die Anzahl der Zeilen, Wörter und Zeichen und gibt für jede Datei die drei Zahlen und bei mehreren Dateien zum Schluß die Geamtzahl der Zeilen, Wörter und Zeichen auf dem Bildschirm aus (engl.: *word count*).

`-l` Ausgabe der Anzahl der Zeilen

`-w` Ausgabe der Anzahl der Wörter

`-c` Ausgabe der Anzahl der Zeichen

Standardvorgabe ist `-lwc`.

2.7.5 Reguläre Ausdrücke

Reguläre Ausdrücke sind eine Verallgemeinerung von festen Zeichenketten, mit denen sehr komplexe Suchmuster in Textdateien formuliert werden können. Wir beschränken uns hier auf die einfachsten Fälle.
Einige Zeichen haben in regulären Ausdrücken eine Sonderbedeutung, wenn sie nicht maskiert sind.

`.` steht für ein beliebiges Zeichen.

`^` steht für den Zeilenanfang.

`$` steht für das Zeilenende.

`[`<*liste*>`]` steht für ein einzelnes Zeichen aus <*liste*> wie in Unterabschnitt 2.5.2 auf Seite 6.

`[^<liste>]` steht für ein einzelnes Zeichen, das nicht aus <*liste*> ist, wie in Unterabschnitt 2.5.2 auf Seite 6.

`*` ist ein *Repetitor* und bedeutet, daß das vorhergehende Zeichen beliebig oft (auch 0-mal) vorkommen soll.

Soll die Sonderbedeutung dieser Zeichen aufgehoben werden, so ist ihnen ein \-Zeichen unmittelbar voranzustellen. Das \-Zeichen verliert ebenfalls seine Wirkung, wenn ihm ein \-Zeichen vorangeht.

`a.*b` beschreibt Zeichenketten, die mit `a` beginnen und auf `b` enden.

`a.*[1-3].*b` beschreibt Zeichenketten, die mit `a` beginnen und mit `b` aufhören und im Innern eine der Ziffern `1`, `2` oder `3` enthalten.

`[0-9][0-9]*` beschreibt Zeichenketten, die aus mindestens einer Ziffer bestehen.

`[^a-z][0-9]*` beschreibt Zeichenketten, deren 1. Zeichen kein Kleinbuchstabe ist und nachfolgend höchstens Ziffern enthalten können.

`A.*\\.*\..*\*` beschreibt Zeichenketten, die mit `A` beginnen, irgendwo im Innern einen Backslash und einen Punkt (in dieser Reihenfolge) enthalten, und auf `*` enden.

`[^$]` beschreibt eine Leerzeile, d. h. eine solche, die keine Zeichen enthält.

`[^ *$]` beschreibt eine leere Zeile, d. h. eine solche, die höchstens Leerzeichen enthalten darf.

2.7.6 Editoren (EMACS)

Zum Bearbeiten einer Textdatei benutzt man spezielle Programme, sogenannte *Editoren*. Hier wird nur der unter UNIX gebräuchliche Editor EMACS besprochen.

```
emacs [&]
```

ruft die Terminalversion des EMACS auf.

```
xemacs [&]
```

ruft die komfortabelere X-Windows-Version auf.
EMACS besitzt verschiedene Arbeitsmodi, in denen er sich etwas verschieden verhält. Wird z. B. ein Programm erstellt, so erkennt er die Schlüsselwörter der benutzten Programmiersprache und färbt sie ein. Drückt man die Tabualatortaste, werden automatisch die „richtigen“ Einrückungen vorgenommen. Die X-Windows-Version kann zum Teil mit der Maus bedient werden. Das Menu in der Kopfleiste ist für Mausbedienung vorgesehen. Aber es können auch mit der linken Maustaste Textteile markiert werden und an andere Stellen durch Drücken der rechten Maustaste übertragen werden. Markierte Bereiche können mit der Tastenkombination `C-w` gelöscht werden und mit `C-y` an anderer Stelle wieder eingefügt werden.
Die meisten Funktionen werden jedoch über die Tastatur aufgerufen. Da die einfachen Tasten Texte eintragen, können die Funktionen des EMACS nur in Verbindung mit Spezialtasten aufgerufen werden. In der Regel sind das die Strg-Taste (Control, im folgenden `C-`) und die Alt-Taste (Meta, im folgenden `M-`). Die Bedeutung der Tasten ist frei konfigurierbar und daher von System zu System verschieden. Die gängigsten Tastenkombinationen für die wichtigsten EMACS-Befehle sind in der folgenden Liste zusammengestellt.

`C-h` Hilfe mit Untermenü

`C-x C-f` öffnet einen Puffer zum Bearbeiten einer Datei. Der Benutzer wird aufgefordert einen Dateinamen einzugeben. Die Eingabe wird mit der [↩]-Taste abgeschlossen. Existiert eine Datei zu dem angegebenen Namen, wird sie in den Puffer geladen, sonst wird sie angelegt (engl.: *file*).

`C-x C-s` speichert den Inhalt des Puffers in der Datei gleichen Namens (engl.: *save*).

`C-x C-w` speichert den Inhalt des Puffers unter einem neuen Namen, der nach dem Befehl eingegeben werden muß (engl.: *write*).

[→], [←], [↑], [↓]

bewegen den Cursor um eine Bildschirmposition nach rechts, links, oben bzw. unten.

`C-a` bewegt den Cursor an den Anfang einer Zeile.

`C-e` bewegt den Cursor an das Ende einer Zeile (engl.: *end*).

`C-f` bewegt den Cursor ein Zeichen vorwärts (engl.: *forward*).

`C-b` bewegt den Cursor ein Zeichen rückwärts (engl.: *backward*).

`M-f` bewegt den Cursor ein Wort vorwärts (engl.: *forward*).

`M-b` bewegt den Cursor ein Wort rückwärts (engl.: *backward*).

`M-<` bewegt den Cursor an den Anfang der Datei, auch die [Pos1]-Taste.

`M->` bewegt den Cursor an das Ende der Datei, auch die [Ende]-Taste.

`C-SPC` setzt eine Marke. (SPC) steht für die Leertaste. Jede folgende Bewegung definiert einen Bereich (Region) zwischen der Marke und der momentanen Position des Cursors. Dieser Bereich kann farblich unterlegt sein oder auch unsichtbar sein. Zwischen diesen beiden Modi kann mit dem Befehl

`M-x transient-mark-mode`

hin- und hergeschaltet werden. Es gibt eine Reihe von Befehlen, die auf solche Bereiche wirken.

Mit `C-g` kann die farbliche Hinterlegung bis zum nächsten Markierungsbefehl abgeschaltet werden.

`C-w` löscht den markierten Bereich ($\Longrightarrow$ `C-y` weiter unten).

`C-x C-x` vertauscht Position der Marke und des Cursors, aktiviert die farbliche Unterlegung, falls sie eingeschaltet ist.

`C-d` löscht das Zeichen, auf dem der Cursor steht (engl.: *delete*).

`M-d` löscht das Wort (nach rechts), auf dem der Cursor steht (engl.: *delete*).

`C-k` löscht alle Zeichen bis zum Ende der Zeile (engl.: *kill*).

`C-k C-k` löscht alle Zeichen bis zum Ende der Zeile und das (unsichtbare) Zeilenumbruchszeichen (engl.: *kill*).

Durch fortgesetzte Wiederholung von `C-k` lassen sich ganze Zeilenbereiche löschen.

`C-y` fügt das zuletzt mit `C-w`, `M-d` oder einer zusammenhängenden Folge von `C-k` Gelöschte an der Stelle des Cursors ein (engl.: *yank*).

`M-y` ersetzt den mit `C-y` eingefügten Text durch den Inhalt der vorletzten Löschung bzw. bei Wiederholung durch noch frühere Löschungen (engl.: *yank*).

`M-c` ersetzt das Zeichen an der Cursorposition durch den entsprechenden Großbuchstaben und bewegt den Cursor ans Wortende (engl.: *capitalize*).

`M-l` ersetzt alle Zeichen ab der Cursorposition bis zum Wortende durch die entsprechenden Kleinbuchstaben und bewegt den Cursor ans Wortende (engl.: *lower case*).

`M-u` ersetzt alle Zeichen ab der Cursorposition bis zum Wortende durch die entsprechenden Großbuchstaben und bewegt den Cursor ans Wortende (engl.: *upper case*).

`M-t` vertauscht die Wörter, zwischen denen der Cursor steht (engl.: *transpose*).

`C-x C-t` vertauscht die Zeile, in welcher der Cursor steht mit der darüberliegenden (engl.: *transpose*).

`C-s` führt eine inkrementelle Suche vorwärts aus (engl.: *search*).

`C-r` führt eine inkrementelle Suche rückwärts aus (engl.: *reverse*).

`M-%` sucht und ersetzt ein Muster durch einen String. Der Benutzer wird aufgefordert, das Suchmuster und den Ersetzungsstring einzugeben. Jeder Ersetzungsvorschlag wird angezeigt und muß durch [y] oder [n] bestätigt oder abgelehnt werden. Gibt man [!] ein, werden alle Suchmuster ohne Rückfrage ersetzt.

`C-g` bricht den laufenden Befehl ab.

`C-x u` macht die letzte Änderung wieder rückgängig. Durch wiederholtes Eingeben können auch weiter zurückliegende Änderungen rückgängig gemacht werden.

`M-x` leitet die direkte Befehlseingabe ein. Hiermit können EMACS-Befehle ausgeführt werden, für die es keine Tastenkombinationen gibt.

`C-q` ermöglicht die Eingabe von Sonderzeichen, für die es keine Tasten gibt.

`C-x 1` stellt die ursprüngliche Größe des Fensters wieder her.

`C-x C-c` beendet die EMACS-Sitzung(engl.: *close*).

2.7.7 Drucken von Dateien

Mit dem Kommando

```
lpr          <dateiname>
```

wird die Datei <*dateiname*> gedruckt. Der Drucker kann nur PostScript-Dateien verarbeiten (Dateiname `*.ps`).
Reine Textdateien (ASCII-Dateien) müssen erst in das PostScript-Format umgesetzt werden. Dazu verwendet man das Kommando

```
a2ps         [[-1 | -2]] [[-l | -p]] [-8] <dateiname>
```

(engl.: *ascii t(w)o postscript*) mit den wichtigsten Optionen

`-1`	1 Seite des Textes auf 1 Blatt Papier
`-2`	2 Seiten des Textes auf 1 Blatt Papier
`-l`	Querformat (engl.: *landscape*)
`-p`	Hochformat (engl.: *portrait*)
`-8`	8-Bit-Code (korrekte Darstellung der deutschen Umlaute)

Ausgabedateien von TEX und LATEX(Suffix `dvi`) müssen erst mit dem Programm `dvips` in PostScript-Dateien umgewandelt werden.

```
dvips        [<optionen>]<datei>[.dvi]
```

erzeugt die Datei <*datei*>`.ps`, die dann ausgedruckt werden kann. Mit

```
dvips        -f [<optionen>]<datei>[.dvi] | lpr
```

wird auf die Standardausgabe ausgegeben und diese auf das Druckprogramm `lpr` umgeleitet (⟹ Unterabschnitt 2.10.2 auf Seite 22). Mit den weiteren Optionen (⟹ `man dvips`) kann ein Teil des Dokuments ausgewählt und das Aussehen des Ausdrucks verändert werden.

`-f`	lenkt die Ausgabe auf die Standardausgabe um.
`-O`<*xoffs*>,<*yoffs*>	verschiebt den Ausdruck auf dem Papier, negative Werte für <*xoffs*> und <*yoffs*> bedeuten Verschiebungen in die entgegengesetzte Richtung.
`-pp` <*liste*>	wählt bestimmte Seiten aus. <*liste*> ist eine durch Kommata getrennte Liste von Seitennummern (<n_1, n_2, ...>) und/oder Seitennummernbereichen (<n_1-m_1, n_2-m_2>, ...)

Aus der TEX- bzw. LATEX-Umgebung des Editors EMACS lassen sich TEX und LATEX-Texte direkt ausdrucken.

2.8 Prozesse

2.8.1 Informationen über Prozesse

Jeder Befehl an das System wird in ein oder mehrere Prozesse zerlegt. Jeder Prozeß bekommt eine Prozeßnummer (PID, engl.: *process identification*), über die er vom System identifiziert wird.
Man unterscheidet Vordergrund- und Hintergrundprozesse (Jobs). Vordergrundprozesse werden gestartet und blockieren, solange sie laufen, das Terminal. Hintergrundprozesse dagegen geben das Terminal nach dem Start sofort wieder frei und geben erst dann eine Meldung aus, wenn sie beendet sind. Da Hintergrundprozesse vom Terminal gelöst sind, dürfen sie keine Ergebnisse auf das Terminal ausgeben.

```
jobs
```

gibt eine numerierte Liste der von dem Terminal gestarteten Hintergrundprozesse aus.
Der zuletzt gestartete Job ist mit `+`, der vorletzte mit `-` gekennzeichnet.
Vordergrundprozesse können jederzeit mit der Tastenkombination Strg + c abgebrochen oder mit der Tastenkombination Strg + z unterbrochen werden.
Ein unterbrochener Prozeß wird sofort in die Jobliste aufgenommen.

```
fg           [%<jobnr>]
```

startet den Prozeß mit der Jobnummer <*jobnr*> oder, falls <*jobnr*> nicht angegeben wurde, den zuletzt unterbrochenen Prozeß im Vordergrund.

```
bg          [%<jobnr>]
```

startet den Prozeß mit der Jobnummer <*jobnr*> oder, falls <*jobnr*> nicht angegeben wurde, den zuletzt unterbrochenen Prozeß im Vordergrund.

```
ps          [<optionenstring>]
```

zeigt alle Prozesse des Benutzers an, die von dem aktuellen Terminal gestartet wurden. Die wichtigsten Optionen sind

`[-]f` zeigt mehr Informationen für den Prozeß an, z. B. die PID des Prozesses, der diesen Prozeß gestartet hat, die PPID (engl.: *parent process identification*), den Speicherbedarf, die Startzeit, die verbrauchte Rechenzeit und den Namen des zugehörigen Programms (engl.: *full*).

`[-]e` zeigt alle auf dem System laufenden Prozesse an (engl.: *every*).

```
top
```

zeigt eine nach CPU-Belastung geordnete Liste aller Prozesse auf dem Rechner an. Die Liste wird laufend aktualisiert. Während das Programm läuft, können andere Prozesse gewaltsam beendet werden (`kill`, $\Longrightarrow$ Unterabschnitt 2.8.2) oder mit anderer (niedrigerer) Prioritätsstufe (`nice`, `renice` $\Longrightarrow$ Unterabschnitt 2.8.3) versehen werden. Beendet wird das Programm durch Drücken der q -Taste.

2.8.2 Vorzeitiges Beenden eines Prozesses

Manchmal ist man gezwungen, einen Prozeß abzubrechen. Dieses erreicht man mit dem Befehl

```
kill        [[-15 | -9]] [<pid> | %<jobnr>]
```

Die Option `-9` ist das schärfste Vorgehen. Man sollte sie in der Regel nur dann verwenden, wenn `-15` oder die Angabe von keiner Option nicht gefruchtet haben.
<*pid*> ist die Nummer, unter der der Prozeß mit dem `ps`-Kommando erscheint, <*jobnr*> ist die Nummer, unter er mit dem `jobs`-Kommando erscheint.

2.8.3 Prioritäten

Jeder Prozeß läuft mit einer gewissen Prorität. Diese entscheidet darüber, wieviel Rechenzeit das Betriebssystem dem Prozeß zuteilt. Die Werte liegen zwischen -20 (höchste) und 19 (niedrigste) Priorität. Damit rechenzeitintensive Prozesse für andere Benutzer das Arbeiten auf dem gleichen Rechners nicht zu sehr behindern, sollte für solche Prozesse die Priorität herabgesetzt werden. Das geschieht beim Starten des Prozesses mit dem Kommando

```
nice        [<wert>] <normale befehlszeile>
```

startet einen Prozeß, als wäre er mit <*normale befehlszeile*> gestartet worden, nur wird seine Priorotätsstufe um <*wert*> gegenüber der laufenden Shell erhöht. Letztere ist meistens 0, so daß der Prozeß mit der Priorotät <*wert*> läuft. Der Default-Wert für <*wert*> ist 10.
Wie der Befehlsname schon andeutet, verhält sich ein solcher Prozeß fair gegenüber anderen Prozessen, vor allen gegenüber interaktiven Prozessen, die wenig Rechenzeit erfordern, bei denen aber eine schnelle Reaktion des Rechners erwartet wird, z. B. in einer Editorsitzung. Ein '"geniceter" Prozeß (Stufe > 10) stört eine Editorsitzung nur unwesentlich, da der „ungenicete" Prozeß praktisch sofort bedient wird. Umgekehrt steht dem „geniceten" Prozeß die gesamte Rechenleistung zur Verfügung mit Ausnahme des kleinen Anteils, den eine Editorsitzung typischerweise verbraucht.
Mit

```
renice      <pid>
```

läßt sich die Prioritätsstufe eines Prozesses nachträglich erhöhen (Erniedrigung der Priorität)

2.9 Ein- und Ausgabekanäle (Streams)

Bevor eine Datei bearbeitet werden kann, muß sie geöffnet werden. Das muß jedes Programm mit Hilfe des Betriebssystems für sich erledigen. Standardmäßig sind jedoch für jedes Programm drei Kanäle geöffnet:

Nummer	Name	Bedeutung
0	stdin	Standardeingabe, normalerweise Tastatur
1	stdout	Standardausgabe, normalerweise Bildschirm
2	stderr	Standardfehlerausgabe, normalerweise Bildschirm

2.9.1 Ein- und Ausgabeumleitung

Die Eingaben, die ein Programm erwartet, müssen nicht unbedingt von der Tastatur kommen. Sie können auch aus einer Datei stammen oder sogar von einem anderem Programm übernommen werden ($\Longrightarrow$ Unterabschnitt 2.10.2 auf Seite 22).
Angenommen, das Programm `prog` erwartet Eingaben von der Tastatur. Durch die Schreibweise

```
prog        [0]< <datei>
```

wird erreicht, daß die Datei <*datei*> zur Standardeingabe wird und die Daten aus der Datei <*datei*> gelesen werden.
Die Ausgabe eines Programms kann ebenfalls umgeleitet werden. Dazu gibt es zwei Möglichkeiten.

```
prog        [1]> <datei>
```

leitet die Ausgabe des Programms `prog` in die Datei <*datei*> um. Existiert die Datei schon zuvor, wird sie gelöscht und neu angelegt.

```
prog        [1]>> <datei>
```

hängt die Ausgabe des Programms `prog` an das Ende der Datei <*datei*> an, falls sie exisitiert. Sonst wird sie neu angelegt.
Die Standardfehlerausgabe kann ebenfalls (in eine andere Datei) umgeleitet werden. Dazu ist vor den Umleitungszeichen `>` bzw. `>>` die Kanalnummer `2` einzufügen.
Mit der Schreibweise

```
prog        &>[>] <datei>
```

wird erreicht, daß beide Ausgabekanäle in dieselbe Datei <*datei*> gelenkt werden.
Für ein Programm dürfen gleichzeitig Ein- und Ausgabe umgeleitet werden. Das Programm `prog` darf mit Optionen und anderen Parametern aufgerufen werden. In solchen Fällen stehen die Umleitungsanweisungen immer am Ende.

2.10 Befehlsgrammatik

2.10.1 Parameterauflösung

2.10.1.1 Auflösung von Parametern, Maskieren

Wenn der BASH ein Befehl mit Parametern übergeben wird, so untersucht sie alle Zeichen darauf, ob sie aufgelöst werden können.
Wird z. B. in einem Befehl eine Dateimaske mit den in Unterabschnitt 2.5.2 auf Seite 6 beschriebenen Zeichen angegeben, so löst die BASH die Dateimaske auf, indem sie dem Befehl eine alphabetisch geordnete Liste aller Dateien übergibt, auf welche die Dateimaske paßt.
Auch Sequenzen, die mit einem `$`-Zeichen beginnen, werden aufgelöst.
Ferner haben bestimmte Zeichen `(`, `)`, `{`, `}` und `;` eine feste syntaktische Bedeutung. Wenn sie im „falschen“ Zusammenhang auftreten, werden sie als Syntaxfehler bewertet.
Dieses Verhalten kann manchmal lästig sein, z. B. wenn man ein solches kritisches Zeichen einem Programm übergeben muß.
Um die Interpretation solcher Zeichen zu unterdücken, muß man diese Zeichen maskieren (quotieren). Dazu gibt es 3 Möglichkeiten:

- Jedem kritischen Zeichen wird ein \ vorangestellt. Dann verliert es seine spezielle Bedeutung. Das funktioniert auch beim \ selbst. Die Folge \\ repräsentiert ein \-Zeichen ohne Sonderbedeutung.
- Teile der Eingabe werden in Anführungszeichen (") eingeschlossen. Dann verlieren außer dem `$`-Zeichen alle eingeschlossenen Zeichen ihre spezielle Bedeutung.
- Teile der Eingabe werden in Apostrophe (') eingeschlossen. Dann verlieren alle eingeschlossenen Zeichen ihre spezielle Bedeutung.

2.10.1.2 (Shell-)Variable

Die BASH kann Variable verwalten. Die Namen dieser Variablen dürfen nur aus alphanumerischen Zeichen bestehen. Groß- und Kleinbuchstaben werden unterschieden. Traditionsgemäß verwendet man für systemnahe Variablennamen nur Großbuchstaben.
Diese Variablen können Strings, das sind Ketten von beliebigen Zeichen, speichern. Der gespeicherte String darf auch leer sein, d. h. keine Zeichen enthalten.

```
set
```

gibt eine Liste der definierten Shell-Variablen aus.
Man unterscheidet exportierbare und nicht exportierbare Variablen. Wenn aus einer Shell eine neue Shell gestartet wird, werden nur die exportierbaren Variablen in die neue Shell übernommen.

```
export      [<var1> [<var2> ...]]
```

macht die Variablen <*var1*>, <*var2*>, ... exportierbar. Werden keine Variablennamen angegeben, gibt der Befehl eine Liste aller exportierbaren Variablen aus.

```
[export]    <varname>=<string>
```

deklariert die Variable <*varname*> und belegt sie mit dem Wert <*string*>. Mit dem Vorsatz `export` ist die Variable exportierbar. Links und rechts von dem `=`-Zeichen dürfen keine Leerzeichen stehen.
Einzelne Shell-Variable können mit dem `echo`-Befehl angezeigt werden.

```
echo        $<name>
```

Mit `$BASH` oder `$HOME` für <*name*> wird z. B. geprüft, ob eine BASH-Shell (⟹ Seite 3) vorliegt oder wie das Home-Verzeichnis (⟹ Seite 5) heißt.
Eine Reihe von Umgebungsvariablen wird automatisch angelegt, einige davon sind nur lesbar und können vom Benutzer nicht verändert werden (⟹ Tabelle 2.2 auf der nächsten Seite).

SHELL	-	Kommandointerpreter
BASH	-	vollständiger Name der BASH
HOME	-	Home-Verzeichnis des Benutzers
LOGNAME	r	Benutzername
UID	r	Benutzerkennng (als Zahl)
HOSTNAME	r	Name des Rechners
PATH	-	Liste der Suchpfade für Befehle, Trenner: Doppelpunkt (:) (Wenn auch Befehle aus dem aktuellen Verzeichnis ausgeführt werden sollen muß das aktuelle Verzeichnis durch . (⟹ Unterabschnitt 2.5.1 auf Seite 5) explizit aufgeführt sein.)
INFOPATH	-	Liste der Suchpfade für das INFO-System
KDEDIR	-	Verzeichnis mit KDE-Dateien
PWD	r	aktuelles Verzeichnis
OLDPWD	r	letztes aktuelles Verzeichnis
PAGER	-	Programm, mit dem Dateien seitenweise angezeigt werden können.
PS<*n*>	-	Aussehen der verschiedenen „Prompts“, d. h. was angezeigt wird, wenn auf eine Eingabe gewartet wird (verschieden je nach Modus, <*n*> = 1 .. 4)
COLUMNS	-	Zahl der Spalten des Terminals
LINES	-	Zahl der Zeilen des Terminals

Tabelle 2.2: Beispiele für Umgebungsvariable der BASH (r: nur lesbar)

2.10.2 Verkettete Befehle (Pipelines)

Bei einer Pipeline werden zwei oder mehr Befehle so miteinander verkettet, daß der jeweils nächste die Ausgabe des vorherhergehenden übernimmt.

```
[time] [!]   befehl1 | befehl2 ...
```

d. h. die Standardausgabe eines Programms kann zur Standardeingabe eines anderen Programms werden. Die Programme `befehl1` und `befehl2` dürfen mit Optionen und Parametern aufgerufen werden.
Die Technik der Aus- und Eingabeverkettung wird benutzt um große Ausgabemengen nach gewissen Kriterien zu filtern. Die im Abschnitt 2.7 auf Seite 13 ff. besprochenen Programme sind für solche Zwecke geeignet, weil sie ihre Eingabedaten aus der Standardeingabe lesen, wenn keine Eingabedateien angegeben werden.
Das Ergebnis einer solchen Pipeline ist das Ergebnis des letzten Befehls der Pipeline (⟹ Abschnitt 2.4 auf Seite 4). Ist `!` angegeben, so wird das Ergebnis invertiert, d. h. aus `0` wird `1`, aus allen anderen Ergebnissen wird `0`.
Ist `time` angegeben, so werden die bis zum Ende des Befehls vergangene Zeit und die vom Benutzer und die vom System verbrauchte Rechenzeit angezeigt.

```
ls -l | less
```

gibt z. B. die Möglichkeit, in der erzeugten Verzeichnisliste „herumzublättern“, was bei großen Verzeichnissen sehr praktisch ist.

2.10.3 Befehlslisten

Eine Befehlsliste entsteht durch Verketten von Pipelines durch Operatoren:

`pipe1` <*op1*> `pipe2` <*op2*> `pipe3 ...`

Die Operatoren, die hier eingesetzt werden dürfen, sind in Tabelle 2.3 auf der nächsten Seite aufgeführt. Die Priorität innerhalb einer Gruppe ist gleich. Die erste Gruppe (`&&` und `||`) hat die höhere Priorität.

<*op*>	Bedeutung
`&&`	UND-Verknüpfung: Der folgende Befehl wird nur ausgeführt, wenn der vorangehende erfolgreich war.
`||`	ODER-Verknüpfung: Der folgende Befehl wird nur ausgeführt, wenn der vorangehende ein von `0` verschiedenes Ergebnis hatte.
`;`	Alle Kommandos werden der Reihe nach abgearbeitet.
`&`	Der vorhergehende Befehl wird im Hintergrund ausgeführt. Auf seine Beendigung wird nicht gewartet. Als Fehler-Code wird `0` angenommen und das folgende Kommando wird gestartet.
<*neue zeile*>	wie `;`

Tabelle 2.3: Operatoren zur Verkettung von Pipelines

2.10.4 Verbundbefehle

Schließt man eine Befehlsliste in runde oder geschweifte Klammern ein[8]; so entsteht ein *Verbundbefehl*. Dieser wird bei Verwendung der runden Klammern in einer eigenen Shell ausgeführt, was z. B. impliziert, daß dieser nur die exportierbaren Umgebungsvariablen (⟹ Unterabschnitt 2.10.1.2) „sieht“, während er bei Verwendung von geschweiften Klammern in der aktuellen Shell ausgeführt wird.

pwd; { cd /usr; pwd; }; pwd wechselt in das Verzeichnis `/usr`

pwd; (cd /usr; pwd); pwd öffnet eine eigene Subshell, wechselt innerhalb dieser Shell in das Verzeichnis `/usr` und kehrt nach Beendigung der Subshell zum Ausgangsverzeichnis zurück.

2.11 Ablaufsteuerung

Die Befehle zur Ablaufsteuerung werden hauptsächlich in Shell-Scripts verwendet.

2.11.1 `if`-Strukturen

Eine `if`-Struktur wird beschrieben durch

```
if <if_liste>; then <ex_if_liste>;
[elif <elif_liste1>; then <ex_elif_liste1>; ...]
[else <ex_else_liste>];
fi
```

Wenn die Bearbeitung von <*if_liste*> erfolgreich abläuft, wird <*ex_if_liste*> ausgeführt und sonst nichts. Ist das nicht der Fall, werden nacheinander die `elif`-Listen bearbeitet, sofern vorhanden, bis eine erfolgreich abgearbeitet werden konnte. Danach wird die entsprechende <*ex_elif_liste*> bearbeitet und hinter `fi` gesprungen.
Wurde weder <*if_liste*> noch eine `elif`-Liste erfolgreich bearbeitet, so wird, sofern vorhanden, <*ex_else_liste*> ausgeführt.

2.11.2 `case`-Strukturen

Durch Verwendung der `case`-Struktur lassen sich komplizierte `if`-Strukturen oft erheblich vereinfachen, nämlich dann, wenn der Wert einer Variablen zu einem bestimmten Muster passen soll. Für den Mustervergleich gelten die gleichen Regeln wie für die Beschreibung von Dateigruppen (⟹ Unterabschnitt 2.5.2).

[8] Bei manchen UNIX-Systemen ist bei geschweiften Klammern zu beachten, daß sie isoliert stehen, d. h. daß links und rechts von ihnen mindestens ein Leerzeichen stehen muß.

```
case <ausdruck> in
    <m1> [|| <m11>] ...)  <liste1>;;
    <m2> [|| <m21>] ...)  <liste2>;;
        ⋮
esac
```

Es wird grundsätzlich nur die Befehlsliste bearbeitet, deren Muster als erstes auf <*ausdruck*> paßt. Paßt keins der angegeben Muster auf <*ausdruck*>, wird kein Befehl ausgeführt.
Das folgende Programmstück wartet auf eine Eingabe, die durch [↩]-Taste abgeschlossen werden muß, und reagiert entsprechend der Eingabe auf verschiedene Weise.

```
while true; do
    echo -n "Eingabe: "
    read
    case $REPLY in
        1)          befehl1;;
        a || A)     befehlA;;
        q || quit)  exit 0;;
        *)          echo -e \\a Eingabe nicht verstanden;;
    esac
done
```

Der Befehl `true` in der `while`-Schleife (⟹ Unterabschnitt 2.11.3) tut nichts, das aber erfolgreich. (`false` tut auch nichts, meldet aber einen Mißerfolg.) Die Option `-n` beim `echo`-Befehl bewirkt, daß der Zeilenvorschub nach der Ausgabe unterbleibt. Der Ausgabestring wurde in Anführungszeichen " eingeschlossen, damit das Leerzeichen mit ausgegeben wird. Die Eingabe, auf die von der `read`-Anweisung gewartet wird, bis sie durch die [↩]-Taste abgeschlossen wird, steht in der Umgebungsvariablen `REPLY`, die mit der `case`-Anweisung ausgewertet wird.
Mit einer Modifikation der `case`-Anweisung, der `select`-Anweisung lassen sich sehr einfach Menüs programmieren (⟹ Handbücher).

2.11.3 Wiederholungen, Schleifen

Die BASH stellt zwei Wiederholungsanweisungen zur Verfügung, deren Abbruch an das Ergebnis von Befehlen gebunden ist.

```
while <test_liste>; do
    <liste>
done
```

und

```
until <test_liste>; do
    <liste>
done
```

Die `while`-Anweisung wird solange wiederholt, wie <*test_liste*> wahr ist, die `until`-Anweisung wird solange wiederholt, wie <*test_liste*> falsch ergibt.
Die `for`-Schleife ordnet nacheinander der Variablen <*name*> die Ausdrücke <*wort*>, <*wort1*>, ... zu und arbeitet für jeden dieser Ausdrücke <*liste*> ab.

```
for <name> [in <wort> <wort1> ...]; do
    <liste>
done
```

Mit `continue` kann die Bearbeitung eines Schleifendurchlaufs beendet werden und mit dem nächsten fortgesetzt werden, mit `break` kann die ganze Schleife beendet werden.

2.12 Tests

In den letzten Abschnitten wurden die Ergebnisse von Befehlslisten benötigt, z. B. <*test_liste*> und in <*if_liste*>. Zum Testen bestimmter Bedingungen an Dateien stellt die BASH eine Reihe von Test-Funktionen zur Verfügung. Tests werden hauptsächlich in Shell-Scripts verwendet.

2.12.1 Datei-Tests

`test` <*op1*> <*datei*>
`test` <*datei1*> <*op2*> <*datei2*>

äquivalent:

`[` <*op1*> <*datei*> `]`
`[` <*datei1*> <*op2*> <*datei2*> `]`

<*op1*>	wahr, wenn <*datei*> existiert
`-e`	
`-f`	und reguläre Datei ist
`-d`	und Verzeichnis ist
`-L`	und symbolischer Link ist
`-r`	und lesbar ist
`-w`	und beschreibbar ist
`-x`	und ausführbar ist

Tabelle 2.4: Einseitige Operatoren zum Testen von Dateieigenschaften

Für <*op2*> gibt es `-nt` (neuer als) und `-ot` (älter als), welche den Zeitpunkt der letzten Änderung der Dateien vergleichen.

2.12.2 String-Tests

`test` <*op1*> <*string*>
`test` <*string1*> <*op2*> <*string2*>

äquivalent:

`[` <*op1*> <*string*> `]`
`[` <*string1*> <*op2*> <*string2*> `]`

<*op1*>	wahr, wenn
`-z`	<*string*> leer engl.: zero
`-n`	<*string*> nicht leer (darf auch weggelassen werden) (engl.: non-zero)

Tabelle 2.5: Einseitige Operatoren zum Testen von Strings

Datei- und String-Tests können durch die Operatoren `!`, `-a`, `-o` verneint oder verbunden werden (not, and, or).

<*op2*>	wahr, wenn <*string1*> und <*string2*>
==	gleich (auch = ist erlaubt)
!=	verschieden
<, >	lexikalisch geordnet
-eq, -ne	numerisch gleich bzw. ungleich
-lt, -le	numerisch kleiner (gleich)
-gt, -ge	numerisch größer (gleich)

Tabelle 2.6: Zweiseitige Operatoren zum Testen von Strings

2.13 Konfigurieren der BASH

Beim Aufruf einer BASH wird entweder die Datei `.bashrc` (normale Shell) oder `.profile` (Login-Shell) des jeweiligen Benutzers ausgewertet. Die in diesen Dateien enthaltenen Befehle und Definitionen werden nach dem der BASH ausgeführt[9]. Man kann also hier ganz persönliche Einstellungen für die eigenen Zwecke vornehmen. Ein Beispiel finden sie in der Datei

```
~froboese/.bashrc
```

Wenn Sie spezielle Wünsche haben, legen Sie sich eine Datei `.bashrc` mit den entsprechenden Befehlen an und zusätzlich die Datei `.profile` mit dem Inhalt

```
test -f .bashrc && .  .bashrc
```

2.14 Shell-Scripts

Immer wiederkehrende Folgen von Befehlen schreibt man am besten untereinander in eine Datei, macht diese ausführbar (`chmod`, ⟹ Unterabschnitt 2.5.4 auf Seite 8) und ruft dann nur noch diese Datei auf[10]. Eine solches Shell-Script wird dann Zeile für Zeile abgearbeitet.
Die oben aufgeführten Sprachelemente geben der BASH die Mächtigkeit einer Programmiersprache wie z. B. BASIC, so daß man mit solchen Scriptdateien recht komplexe Aufgaben lösen kann, ohne daß man gleich zu FORTRAN oder C/C++ greifen muß.
Mit

```
man bash
```

findet man detailierte Hilfe zu allen Eigenschaften der BASH.

Beispiel:

```
#!/bin/bash     # diese Zeile muß immer am Anfang eines Shell-Scripts stehen
echo Wie heißen Sie?
read            # Name wird nach REPLY eingelesen
if [ -z $REPLY ]; then
# alternativ:  if test -z $REPLY; then
    echo Sie haben keinen Namen angegeben!
    exit 1      # Programm wird mit Return-Code 1 beendet
fi
HOUR=$(date +_%H)
case $HOUR in
    [0-9] || 1[0-1])    H=Morgen;;  # vor 12
    1[2-8] )            H=Tag;;     # vor 19
    *)                  H=Abend;;   # sonst
esac
```

[9] ähnlich wie `AUTOEXEC.BAT` bei DOS-Systemen
[10] ähnlich wie BATCH-Dateien bei DOS-Systemen

```
echo "Guten $H, $REPLY!"
echo "Welche Datei soll angezeigt werden (Name und Pfad)?"
if [ -f $REPLY ]; then
    cat $REPLY
else
    echo "Die Datei $REPLY gibt es nicht."
fi
```

Das Zeichen # an einem Wortanfang leitet einen Kommentar ein. Von diesem Zeichen ab wird der Rest der Zeile überlesen. Einzige Ausnahme: Folgt in der 1. Zeile der Datei auf # ein Ausrufungszeichen, so wird der Rest der Zeile als Aufruf des Interpreters aufgefaßt, so daß die 1. Zeile aller Script-Dateien wie die 1. Zeile der Beispiele
Gerade weil die Befehle so mächtig sein können, sind Kommentare unerläßlich.
Shell-Scripts können auch beim Aufruf übergebene Parameter verarbeiten. Diese werden innerhalb des Scripts entsprechend ihrer Reihenfolge im Aufruf als `$1, $2` usw. angesprochen. Dieses wird am Beispiel des Shell-Scripts `calc` gezeigt, mit dem sich rechen läßt. Das Script ruft dazu das Programm `bc` auf, das auf (fast) allen UNIX-MASCHINEN installiert ist.
Die ausführbare Datei `calc` enhält

```
#!/bin/bash         # hier muß der vollständige Dateiname der BASH stehen
if [ -n $2 ]; then
    prec=$2             # vorgegebene Genauigkeit
else
    prec=5              # Default-Genauigkeit 5 Nachkommastellen
fi
echo "scale $prec; $1" | bc -l
```

Eine kürzere Fassung (für fortgeschrittete BASH-Programmierer) mit gleicher Funktion sieht so aus:

```
#!/bin/bash
echo "scale=$2:-5; $1" | bc -l
```

Der Aufruf `calc "2/3"` liefert `0.66666` [11]. Der Aufruf `calc "2/3" 10` liefert `0.6666666666`.

[11] `bc` rundet nicht, sondern schneidet nach jeder Rechnung auf die mit `scale` vorgegebene Zahl von Stellen ab.

Inhaltsverzeichnis