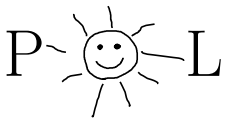
Einführung in das Betriebssystem UNIX



Dr.-Ing. Peter Leibner

Tel.: +49 (0)3834 43 99 579
e-mail: pel@kedaj.org

Inhaltsverzeichnis

1 Einführung

UNIX ist ein interaktives Betriebssystem. Zum Dialog mit dem Rechner dient die SHELL. Sie ist ein Programm, das zeilenweise die eingegebenen Befehle von links nach rechts liest, interpretiert und zur Verarbeitung an den Systemkern weitergibt. Der einzige rechnerabhängige Teil des Betriebssystems ist der Systemkern (→ Abb.1.1).

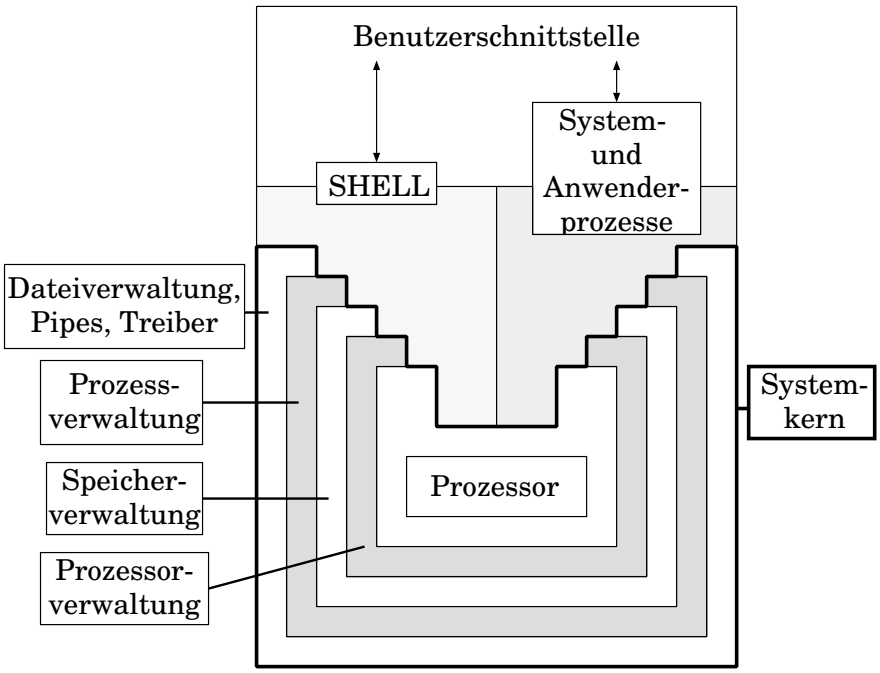


Abbildung 1.1: Architektur des Betriebssystems UNIX

Zur optimalen Ausnutzung des Prozessors im Multiuser- Multitasking-Betrieb, des Speichers, der Prozeß- und Dateiverwaltung, sowie der Rechnerperipherie, dienen die vier Schichten des Systemkerns. Der Systemkern wird beim Systemstart in den Hauptspeicher geladen, wo er während des gesamten Rechnerbetriebs verbleibt. Alle anderen Programme werden nur zu ihrer Ausführung in den Hauptspeicher geladen. Jedem Programm liegt ein Prozeß zugrunde, der dieses Programm ausführt. Findet beim Multitasking-Betrieb im zu kleinen Hauptspeicher kein weiteres Programm genügend Platz, so wird ein dort residierendes zwischenzeitlich auf die Festplatte ausgelagert (**swapping**).

Die Programme des Systemkerns ermöglichen neben der Verwaltung des Hauptspeichers, der Prozesse, des Ausgleichsspeichers (**buffer cache**, zur Beschleunigung der Datenübertragung zwischen Speichermedium und Hauptspeicher), usw., die gleichzeitige Nutzung von Betriebsmitteln durch mehrere Benutzer mittels Systembefehlen (**system**

calls). So ein Systembefehl kann z.B. die Anforderung eines Prozesses zur Datenübertragung aus einem peripherem Speichermedium in den Datenbereich des Prozesses im Hauptspeicher sein. Der Prozeß setzt dazu den Systembefehl **read** an den Systemkern ab. In der Zeit der Ausführung der Anforderung ist der Prozeß im Regime des Systemkerns. In dieser Zeit kann der Systemkern durch keine anderen Prozesse unterbrochen werden.

Diese Systembefehle stehen auch als Funktionen der Programmiersprache C zur Verfügung.

Teil des Systemkerns sind ferner umfangreiche Tabellen zur Verwaltung der geöffneten Dateien, der laufenden Prozesse, der Konfiguration der Peripherie, usw.

UNIX beinhaltet ca. 200-300 fertige Programme (**tools**). Sie werden durch die Übergabe eines „äußeren“ Kommandos an die SHELL gestartet. Neben diesen „äußeren“ Befehlen, zu deren Ausführung immer ein neuer Prozeß generiert wird (z.B. **vi** und **rm**), gibt es auch „innere“, die in der SHELL implementiert sind und daher ohne Prozeßgenerierung laufen (z.B. **echo** und **cd**).

Zusätzlich zu diesen Befehlen kann der Anwender eigene Kommandos erzeugen. Dies erfolgt entweder durch ihre Implementierung in C oder durch SHELL-Skripts (Befehlsfolgen, einschließlich Verzweigungen und Schleifen).

Die Möglichkeit, Prozesse im Hintergrund ablaufen zu lassen oder bei Verwendung von graphischen Oberflächen (X-Window) mehrere Fenster zu generieren und die Programme in eigenen Fenstern zu starten, erlaubt die parallele Ausführung mehrerer Prozesse. Den Prozessen wird dabei die Prozessorzeit gemäß ihrer Priorität zugeteilt.

1.1 Geschichte

Die Entstehung von UNIX wird auf das Jahr 1969 zurückdatiert. Zu dieser Zeit beschäftigten sich in den Bell Laboratories der Firma AT&T unter anderen D. Ritchie, B. Kernighan und K. Thompson mit der Entwicklung des Betriebssystems MULTICS. Da MULTICS sehr groß und unhandlich geraten war, versuchte vor allem Ken Thompson ein einfacheres Betriebssystem zu entwickeln. Dieses wurde für den 8-bit Minirechner PDP-7 in Assembler geschrieben. Bell Labs. lehnte zunächst weitere finanzielle Mittel für das Projekt ab. Erst als ein Dokumentationssystem, das für die Patentabteilung benötigt wurde, implementiert wurde, konnte die Entwicklung auf einer PDP-11 fortgesetzt werden.

Im Laufe mehrerer Jahre hat D. Ritchie die typenfreie Programmiersprache BPCL entwickelt, aus der durch Hinzufügen von Datenstrukturen später C wurde. Die Entwicklung von C wurde ausschließlich von dem Wunsch geleitet, UNIX in eine höhere Programmiersprache überzuführen und damit portierbar zu machen. Heute sind ca. 90% von UNIX in C geschrieben. Der prozessorabhängige Rest ist in Assembler implementiert.

Im Jahre 1979 erfolgte die Weitergabe der Quelldateien der im Jahre 1975 entstandenen Version 7 an die University of California at Berkeley (UCB). Dort entstand eine spezielle

Forschungsgruppe (Computer Science Research Group (CSRG)), die zunächst UNIX auf eine VAX-11/780 portierte und dann mit der Entwicklung eigener Versionen begann. Die weitere Entwicklung von UNIX erfolgte von nun an zweigleisig (→ Abb.1.2).

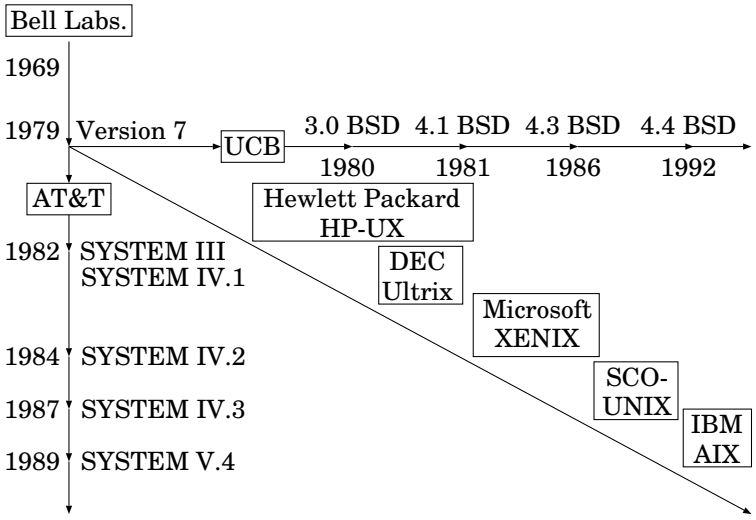


Abbildung 1.2: UNIX-Versionen

Die an der UCB entstandenen Versionen werden mit BSD (Berkeley Software Distribution) bezeichnet. In diesen Versionen wurde UNIX um die virtuelle Adressierung (3.0 BSD), die C-SHELL und den bildschirmorientierten Editor **vi** (4.1 BSD), sowie um Werkzeuge zur Arbeit in Netzen (4.3 BSD), erweitert.

Im Jahre 1982 wurde bei AT&T die Version SYSTEM III fertiggestellt. Diese Version leitete die Vermarktung von UNIX ein. Im gleichen Jahr folgte dann SYSTEM IV mit den Teilversionen (als Release bezeichnet) 1, 2 und 3. Es folgte dann im Jahre 1989 SYSTEM V Release 4 (SVR4), welche die Eigenschaften der AT&T- mit denen der BSD-Version verbindet.

Im Laufe der 80er Jahre entstanden bei unterschiedlichen Herstellern weitere UNIX-Versionen, die sich zwar wie UNIX verhielten, jedoch keine Garantie für die Portierbarkeit der in ihnen geschriebenen Programme gewährleisteten. Aus diesem Grund hat AT&T die SVID-Empfehlung (System V Interface Definition) herausgegeben, die eine generelle Übertragbarkeit von Quelldateien auf andere UNIX-Systeme im Hinblick auf die Systembefehle garantiert. Einen weiteren Versuch der Normierung unternahm 1984 die europäische Anwendervereinigung X/OPEN. Der „X/OPEN Portability Guide“ stimmt in den grundlegenden Gedanken mit SVID überein.

IEEE (Institute of Electrical and Electronics Engineers) hat die Funktionen des Systemkerns in der Norm POSIX 1003.1-1988 definiert. POSIX geht in der Normierung weiter als X/OPEN, da nicht nur die Übertragbarkeit von Quelldateien zwischen UNIX-Systemen, sondern auch zwischen unterschiedlichen Betriebssystemen definiert wird. POSIX deckt auch die Normierung von C, SQL, X-Window, ISO, OSI und NFS ab.

Die Entwicklung von UNIX wird heute wesentlich durch zwei Firmengruppen bestimmt. In UNIX International sind die Firmen Amdahl, CDC, Fujitsu, ICL, Informix, Intel, Motorola, NCR, Olivetti, Sun und Unisys zusammengeschlossen, in der Open Software Foundation sind es die Firmen IBM, HP, DEC, Groupe Bull und Siemens-Nixdorf.

2 Grundsätzliches

2.1 An- und Abmeldung des Benutzers

Zur Anmeldung am System muß der Benutzername und ein Kennwort eingegeben werden. Das System fordert zur Eingabe des Benutzernamens durch

```
login:
```

und zur Eingabe des Kennworts durch

```
password:
```

auf. Jede Eingabe in das System wird durch **<return>** abgeschlossen. Nach der Eingabe des Kennworts, das aus Sicherheitsgründen nicht auf dem Bildschirm wiedergegeben wird, meldet sich das System mit dem „Prompt“

```
$
```

oder, falls der Benutzer Systemverwalter (**root**) ist, mit

```
#
```

Nach dem Erscheinen des Prompts, kann der Benutzer Befehle an die SHELL übergeben. So wird z.B. durch

$ date

```
Tue Jul 25 15:43:51 MET DST 1995

$
```

das aktuelle Datum ausgegeben. Nach Eingabe des Kommandos `passwd`, kann der Benutzer sein Kennwort ändern.

$ passwd

```
Changing password for leibner.
```

```
Old password: altes Kennwort
New password: neues Kennwort
Retype new password: neues Kennwort

$
```

Die Abmeldung vom System erfolgt entweder durch

$ <ctrl d>

(gleichzeitiges Drücken der Tasten **<ctrl>** und **<d>**) oder durch

$ exit

2.2 Befehlsstruktur

Befehle können in unterschiedlichen Formen vorkommen. Der einfachste Befehl hat keine Argumente. Er wird nach der Eingabe von **<return>** ausgeführt. Die Ausgabe des aktuellen Arbeitsverzeichnisses erfolgt z.B. durch

$ pwd

```
/usr/home/leibner/unix

$
```

Sind Argumente (Schalter, Dateinamen) obligatorisch, so werden diese durch ein oder mehrere Leerzeichen (seltener mit dem Tabulator) von einander getrennt. So löscht z.B. das Kommando

$ rm -i file.*

alle Dateien, deren Name mit **file.** anfängt und mit einer beliebigen Folge von Zeichen endet. Der Schalter **-i** (interactive) bewirkt, daß vor dem Löschen der Datei nochmals nachgefragt wird, ob wirklich gelöscht werden soll.

$ rm -i file.*

```
rm: remove file.tex? n
rm: remove file.dvi? y
```

$

Die Syntax der Eingabezeile kann von Befehl zu Befehl unterschiedlich sein, bei den meisten Befehlen erfolgt die Eingabe der Schalter jedoch vor dem Dateinamen. Bei manchen können die Schalter auch zusammengefaßt werden, z.B. bei dem Befehl zur Ausgabe aller Prozesse des Benutzers,

$ ps -elf

bei anderen müssen sie getrennt eingegeben werden, wie z.B. bei dem **sed**-Kommando (**stream editor**), mit dem im folgenden Beispiel alle Vorkommen von **Jan** in der Datei **leute** durch **Peter** ersetzt und die entsprechenden Zeilen in die Datei **Leute** geschrieben werden.

$ sed -n -e /Jan/s//Peter/p leute >Leute

Die Schalter modifizieren die Bedeutung der Kommandos. Sie werden im allgemeinen durch ein - eingeleitet (seltener durch + oder gar nichts). Die Namen der Dateien können Sonderzeichen, sogenannte *Metazeichen* (**wildcards**), enthalten. Ein Beispiel für ein Metazeichen ist der oben verwendete *****. Der ***** wird zur Expansion des Dateinamens verwendet. Tritt er in einem Dateinamen auf, so ersetzt die SHELL den ***** durch eine beliebig lange (auch der Länge Null) Folge von Zeichen.

2.3 Dokumentation

Die heute mitgelieferte Dokumentation entspricht in ihrem Aufbau derjenigen der Version 7. Sie existiert in Form einer Referenz (**manual**) und einer Dokumentation (**documents**). In der Regel liegt die Dokumentation in gedruckter Form vor, die Referenz als Dateien (**on-line manual**). Die Kapiteleinteilung sollte jedoch in beiden Fällen die gleiche sein (→ Tab.2.1).

Kapitel	**Inhalt**
1	Benutzerbefehle (**commands**)
2	Systemkommandos (**system calls**)
3	Unterprogramme (**subroutines**), (Beschreibung von Bibliotheksfunktionen)
4	Dateiformate (**file formats**), (Treiber, spezielle Dateien, Kommunikationsprotokolle, Netzschnittstelle)
5	Verschiedenes (**miscellaneous**), (Dateiformate für unterschiedliche Befehle, z.B. zur Archivierung)
6	Spiele und Demonstrationsprogramme, (häufig nicht mehr vorhanden)
7	Spezielle Dateien (**special files**), (Makroprozessoren zur Textverarbeitung, Tabellen zur Zeichenkodierung, Beschreibung der Normierungen X/Open und POSIX)
8	Befehle des Systemverwalters (**superuser commands**)

Tabelle 2.1: Kapitel des Manuals

Jedem Befehl ist im Manual eine „Manualseite“ (**manual page**) gewidmet. Diese Seite kann sich über mehrere DIN-A4-Seiten erstrecken. Informationen zum Kommando **pwd** erhält man z.B. durch

```
$ man  pwd
```

```
pwd(1)                                          pwd(1)

NAME
  pwd -Displays the pathname of the current (working) directory

SYNOPSIS
  pwd

DESCRIPTION
  The pwd command writes to standard output the full pathname of your
current directory, from the root directory.  All directories are
separated by a / (slash).  The root directory is represented by
the first /, and the last directory named is your current directory.

RELATED INFORMATION
  Commands:  cd(1), csh(1), ksh(1), sh(1).
  Functions:  stat(2).

$
```

Der Aufbau der Seiten ist immer gleich, er enthält jedoch bei komplizierteren Kommandos weitere Abschnitte (z.B. FLAGS bzw. OPTIONS, EXAMPLES, FILES, usw.). Informationen zur Benutzung des Manuals erhält man durch

```
$ man  man
```

Ohne Angabe eines Kapitels werden alle Kapitel nach dem Kommando durchsucht. Es kann vorkommen, daß ein Kommandoname in unterschiedlichen Kapiteln existiert. Mit

```
$ man  1  pwd
```

wird die Suche auf Kapitel 1 beschränkt. Einen Überblick über den Inhalt von Kapitel 2 erhält man z.B. mit

```
$ man  2  intro
```

Eine weitere Hilfe zum Suchen im Manual ist der Schalter **-k**:

```
$ man  -k  assembler
```

```
a.out (4)               - Assembler and link editor output
as (1)                  - assembler
disassembler (3)        - disassemble a machine
                          instruction and print the results

$
```

Mit Hilfe des Schalters **-k** erhält man eine einzeilige Kurzbeschreibung der Befehle, die im Abschnitt NAME das gesuchte Wort (**keyword**) enthalten. Soll nach einer Textpassage gesucht werden, so muß diese in Hochkommas eingeschlossen werden. Durch

$ man -f /etc/passwd

```
passwd (4)              - Password files
passwd, chfn, chsh (1)  - Changes password file information
```

erhält man alle Kommandos, die mit der Datei (file) **/etc/passwd** arbeiten.

Mit Hilfe von Editor-Befehlen (→ **vi**) ist es möglich, sich innerhalb der Manualseite auf und ab zu bewegen und nach Begriffen zu suchen.

Mit **<h>** erhält man eine Übersicht über die zugelassenen **vi**-Befehle.

Mit **<return>** wird jeweils die nächste Zeile dargestellt, durch die Eingabe eines Leerzeichens erhält man die nächste Seite.

Die Ausgabe der Seite kann jederzeit mit **<q>** beendet werden.

Die im Abschnitt NAME stehende Information eines Kommandos erhält man durch

$ whatis pwd

```
pwd (1)  - Displays the pathname of the current (working) directory

$
```

Der Ausdruck der Manualseite für **pwd** auf einem PostScript-Drucker erfolgt durch

$ man -troff pwd | lpr

3 Der Editor **vi**

Im folgenden wird der Editor **vi** (**visual interactive**) vorgestellt. Er ist Teil des zeilenorientierten Editors **ex**. Dieser wurde an der UCB von Bill Joy (er schrieb den größten Teil der BSD-Programme, unter anderen auch die C-SHELL; später gründete er die Firma *Sun Microsystems*) geschrieben und zum ersten Mal 1981 mit der 4.1 BSD ausgeliefert.

Teil der offiziellen AT&T-Version wurde er erst im Jahre 1989 im SYSTEM V!

Der Vorgänger des **ex** ist der **ed**. Beide sind zeilenorientiert, beim **ex** werden jedoch alle Zeilen der Datei dargestellt, wogegen der **ed** nur die aktuelle anzeigt. Dies erschwert den Umgang mit dem **ed** wesentlich. Da der **sed** jedoch weitgehend die gleichen Kommandos wie der **ed** verwendet, kann es nicht schaden, wenn man sich eine zeitlang mit dem **ed** beschäftigt.

Da die Bildschirme immer schneller und flexibler wurden, entstand der Wunsch nach einem bildschirmorientierten Editor. Joy schrieb daher eine Schnittstelle zum **ex**, den **vi**. Der **ex** und der **vi** sind folglich ein und dasselbe Programm, der **ex** wurde lediglich um die bildschirmorientierten Befehle erweitert.

Alle **ex**-Kommandos werden durch die Eingabe eines Doppelpunktes (**:**) im Befehlsmodus des **vi** zugänglich (was oft von großem Vorteil ist).

Will man dauerhaft in den **ex** wechseln, so ist im **vi** im Befehlsmodus **Q** einzugeben.

Aus dem **ex** kommt man andererseits durch die Eingabe von **visual** in den **vi**.

Der **vi** wird durch

$ vi *dateiname*

gestartet. Der Cursor befindet sich danach am Anfang des ersten Wortes der ersten Zeile. Möchte man den Cursor zu Beginn z.B. auf das erste Wort in der 250-ten Zeile einstellen, so muß

$ vi +250 *dateiname*

eingegeben werden. Auf die letzte Zeile wird der Cursor durch

$ vi + *dateiname*

positioniert. Mit

$ vi +/*zeichenkette* *dateiname*

wird der Cursor an den Anfang der Zeile gesetzt, in der *zeichenkette* zum ersten Mal vorkommt. Mit **n** kann man dann nach dem nächsten Vorkommen suchen (mit **N** rückwärts).

Der **vi** schreibt den Text zuerst nicht in eine Datei, sondern legt ihn im Ausgleichsspeicher ab. Der Text wird erst beim Beenden des **vi** oder beim expliziten Schreiben in einer Datei auf der Festplatte gespeichert. Alle Befehle, die zu einer Änderung des Textes geführt haben, werden während dem Editieren in einer Hilfsdatei zwischengespeichert. Stürzt der Rechner beim Editieren ab, so kann der aktuelle Stand der Datei durch

$ vi -r *dateiname*

wieder hergestellt werden ;-).

Zum Verlassen des Editors gibt es mehrere Möglichkeiten. Mit dem Kommando

1. **ZZ** wird der geänderte Text auf die Festplatte gespeichert.
2. Bei der Eingabe von **:q!** werden keine Änderungen abgespeichert.

Ratsam ist es, Änderungen des Textes auch zwischendurch auf die Festplatte zu schreiben. Dies geschieht durch **:w**.

Beim Bearbeiten des Textes wird zwischen drei Modi unterschieden:

1. Befehlsmodus
2. Eingabemodus
3. Modus der letzten Zeile

Nach dem Aufruf des **vi** ist man im Befehlsmodus. Im Befehlsmodus wird eine Eingabe als Kommando, im Eingabemodus als Text, verstanden. Zur Eingabe der bildschirmorientierten Kommandos ist kein **<return>** notwendig, die Eingabe der Kommandos der letzten Zeile muß mit **<return>** abgeschlossen werden.

3.1 Cursorbewegungen

Da der **vi** in der Regel mit dem Text (Buchstabe, Wort, Zeile, Satz, Absatz) arbeitet, auf dem sich gerade der Cursor befindet, sollen zunächst die Befehle besprochen werden, die für die Cursorbewegung verantwortlich sind.

Mit **<return>** wird der Cursor auf das erste Wort der nachfolgenden Zeile verschoben. Wird z.B. **60+** (oder **60<return>**) eingegeben, so verschiebt er sich um 60 Zeilen nach vorn.

Durch die Eingabe von **-** kann die Richtung umgekehrt werden. Mit **4-** erfolgt z.B. eine Verschiebung des Cursors um 4 Zeilen zurück auf das erste Wort der Zeile.

In Tabelle 3.1 ist n ein optionaler Wiederholungsfaktor (ohne Angabe von n wird in der Regel *1* angenommen). So bedeutet z.B. **3$** die Verschiebung des Cursors an das Ende der dritten Zeile nach der aktuellen, **0** an den Anfang der aktuellen Zeile und ^ an den Anfang des ersten Wortes der aktuellen Zeile.

Tastatur	Verschiebung des Cursors
n**h**	um n Zeichen nach links
n**l**	um n Zeichen nach rechts
n**j** (n+)	um n Zeilen nach unten
n**k** (n-)	um n Zeilen nach oben
n**$**	an das Zeilenende, n Zeilen weiter
0	auf das erste Zeichen der Zeile
^	auf das erste Zeichen der Zeile das nicht Leerzeichen oder Tabulatorzeichen ist
n**H**	auf die n-te Zeile des Bildschirms von oben
M	auf die Mitte des Bildschirms
n**L**	auf die n-te Zeile des Bildschirms von unten
n**G**	auf die n-te Zeile (erste Zeile: **1G**)
G	auf die letzte Zeile

Tabelle 3.1: Horizontale und vertikale Cursorverschiebungen

Cursorbewegungen können auch über Wörter, Sätze und Absätze hinweg erfolgen. Bei der Verschiebung um Wörter wird bei den Kommandos zwischen Klein- und Großbuchstaben unterschieden.

Die Befehle **w**, **e** und **b** verstehen unter *Wort* eine Folge von Buchstaben, Zahlen und dem _ (**underline**), begrenzt durch ein beliebiges Zeichen.

[P]eter's Kaffee. $\overset{\mathbf{w}}{\rightarrow}$ Peter[']s Kaffee. $\overset{\mathbf{w}}{\rightarrow}$ Peter'[s] Kaffee. $\overset{\mathbf{w}}{\rightarrow}$ Peter's [K]affee.

[P]eter's Kaffee. $\overset{\mathbf{e}}{\rightarrow}$ Pete[r]'s Kaffee. $\overset{\mathbf{e}}{\rightarrow}$ Peter[']s Kaffee. $\overset{\mathbf{e}}{\rightarrow}$ Peter'[s] Kaffee.

Peter's Kaffe[e]. $\overset{\mathbf{b}}{\rightarrow}$ Peter's [K]affee. $\overset{\mathbf{b}}{\rightarrow}$ Peter'[s] Kaffee. $\overset{\mathbf{b}}{\rightarrow}$ Peter[']s Kaffee.

Die Befehle **W**, **E** und **B** arbeiten genauso wie **w**, **e** und **b**, mit dem Unterschied, daß sie als *Wort* eine durch Leerzeichen begrenzte beliebige Folge von Zeichen auffassen.

[P]eter's Kaffee. $\overset{\mathbf{W}}{\rightarrow}$ Peter's [K]affee.

[P]eter's Kaffee. $\overset{\mathbf{E}}{\rightarrow}$ Peter'[s] Kaffee. $\overset{\mathbf{E}}{\rightarrow}$ Peter's Kaffee[.]

Peter's Kaffe[e]. $\overset{\mathbf{B}}{\rightarrow}$ Peter's [K]affee. $\overset{\mathbf{B}}{\rightarrow}$ [P]eter's Kaffee.

Tastatur	Verschiebung des Cursors
n**w**	um n Worte auf das erste Zeichen nach rechts
n**e**	um n Worte auf das letzte Zeichen nach rechts
n**b**	um n Worte auf das erste Zeichen nach links
n**(**	um n Sätze auf den Satzanfang zurück
n**)**	um n Sätze auf den Satzanfang vorwärts
n**{**	um n Absätze auf den Absatzanfang zurück
n**}**	um n Absätze auf den Absatzanfang vorwärts

Tabelle 3.2: Cursorverschiebungen über Textteile

Zur Verschiebung um Sätze dienen runde Klammern. Als *Satz* wird eine Folge von Zeichen, begrenzt durch einen Punkt `.`, ein Fragezeichen `?` oder Ausrufezeichen `!`, verstanden.

Die Verschiebung um Absätze erfolgt durch geschweifte Klammern. *Absätze* sind Textteile, getrennt durch eine Leerzeile. Wie bei den Befehlen in Tab.3.1, kann zur Wiederholung n vor den jeweiligen Kommandos stehen ($\rightarrow$ Tab.3.2). Eine Verschiebung des Cursors um 3 Wörter nach rechts erfolgt dann z.B. durch **3w**.

Zur Verschiebung des Cursors innerhalb einer Zeile dienen die Befehle **f**, **F**, **t** und **T** ($\rightarrow$ Tab.3.3).

P eter's Kaffee. $\overset{\mathbf{ft}}{\rightarrow}$ Pe t er's Kaffee. $\overset{\mathbf{3fe}}{\rightarrow}$ Peter's Kaffe e. $\overset{\mathbf{Fr}}{\rightarrow}$ Pete r 's Kaffee.

P eter's Kaffee. $\overset{\mathbf{tt}}{\rightarrow}$ P e ter's Kaffee. $\overset{\mathbf{3te}}{\rightarrow}$ Peter's Kaff e e. $\overset{\mathbf{Tr}}{\rightarrow}$ Peter ' s Kaffee.

Tastatur	**Verschiebung des Cursors**
n**f**c	auf das n-te Zeichen c nach rechts
n**F**c	auf das n-te Zeichen c nach links
n**t**c	ein Zeichen vor das n-te Zeichen c nach rechts
n**T**c	ein Zeichen vor das n-te Zeichen c nach links
;	Wiederholung des zuletzt eingegebenen Kommandos in gleicher Richtung
,	Wiederholung des zuletzt eingegebenen Kommandos in entgegengesetzter Richtung

Tabelle 3.3: Cursorverschiebungen innerhalb einer Zeile

Ein für die C-Programmierung nützlicher Befehl ist %. Befindet sich der Cursor auf einer Klammer so wird er durch % auf die korrespondierende Klammer verschoben.

```
main()                                  main()
{                          %            {
      printf("Hallo Welt\n")   ->              printf("Hallo Welt\n")
}                                       }
```

Der Ausschnitt des Textes aus dem Ausgleichsspeicher wird durch die Größe des Bildschirms begrenzt. Man braucht daher Befehle, die diesen Ausschnitt verändern. Die Verschiebung des Ausschnitts kann durch *Rollen* (**scrolling**) oder *seitenweise* (**paging**) erfolgen.

Zum Rollen dienen die Kommandos **<ctrl d>** und **<ctrl u>**. Sie verschieben die Seite um eine halbe nach unten respektive nach oben. Der Cursor bleibt dabei in der aktuellen Zeile, die Seite darunter wird verschoben. Wird vor dem Kommando ein Wiederholungsfaktor n eingegeben, so wird die Seite um n Zeilen verschoben.

Die seitenweise Verschiebung erfolgt durch die Befehle **<ctrl f>** und **<ctrl b>**. Sie verschieben die Seite um eine ganze nach unten respektive nach oben. Der Cursor geht dabei an den Anfang der Seite bzw. an ihr Ende. Wird vor dem Kommando ein Wiederholungsfaktor n eingegeben, so erfolgt eine Verschiebung um n Seiten.

Mit **<ctrl l>** wird die dargestellte Seite neu geschrieben (**refresh**).

Oft möchte man die aktuelle Zeile an eine andere Stelle im Fenster bringen. Dazu dient das Kommando **z** (→ Tab.3.4).

In der nachfolgenden Tabelle sind die besprochenen Befehle zusammengefaßt. Da der Wiederholungsfaktor n dabei kaum verwendet wird, wurde er weggelassen.

Tastatur	**Tätigkeit**
<ctrl f>	Auffüllen des Bildschirms mit weiteren Zeichen aus dem Ausgleichsspeicher so, daß die letzten zwei Zeilen des ursprünglichen Bildschirms zu den ersten zwei des neuen werden
<ctrl b>	Auffüllen des Bildschirms mit weiteren Zeichen aus dem Ausgleichsspeicher so, daß die ersten zwei Zeilen des ursprünglichen Bildschirms zu den letzten zwei des neuen werden
<ctrl d>	Auffüllen des Bildschirms mit weiteren Zeichen aus dem Ausgleichsspeicher so, daß die untere Hälfte zur oberen wird und die untere mit neuen Zeilen aufgefüllt wird
<ctrl u>	Auffüllen des Bildschirms mit weiteren Zeichen aus dem Ausgleichsspeicher so, daß die obere Hälfte zur unteren wird und die obere mit neuen Zeilen aufgefüllt wird
z<return>	Die aktuelle Zeile wird zur ersten Zeile des Bildschirms
z.	Die aktuelle Zeile wird zur mittleren Zeile des Bildschirms
z-	Die aktuelle Zeile wird zur letzten Zeile des Bildschirms
<ctrl l>	Erneuerung der dargestellten Seite

Tabelle 3.4: Beeinflussung der Seitendarstellung

Um bei der Bearbeitung an eine bestimmte Stelle im Text jederzeit zurückkehren zu können, kann diese Stelle markiert werden. Dazu dient der Befehl **m**. Zur Markierung stehen die Buchstaben a-z zur Verfügung. So wird durch **ma** die aktuelle Stelle des Cursors als **a** markiert.

Die Rückkehr an diese Stelle ist auf zweierlei Art und Weise möglich. Nach **'a** (**Hochkomma**) kommt man an den Anfang der Zeile, in der die Markierung erfolgte, durch **‘a** (**umgekehrtes Hochkomma**) an die Stelle, auf der sich der Cursor bei der Markierung befand.

Alle bisher besprochenen Kommandos werden im Befehlsmodus des **vi** eingegeben.

Durch die Eingabe von **:**, **/**, **?** bzw. **!** kommt man in den Modus der letzten Zeile.

So wird z.B. der Cursor nach Eingabe von **/wort** auf das nächste Auftreten von **wort** verschoben, durch **?wort** auf das vorhergehende. Mit **n** wird die Suche in gleicher, mit **N** in entgegengesetzter Richtung wiederholt (→ S.12).

3.2 Texteingabe

Zur Texteingabe dienen die Befehle des Eingabemodus (→ Tab.3.5). Der Übergang vom Eingabe- in den Befehlsmodus erfolgt durch ESCAPE (**<esc>**). Bei textverändernden Kommandos wird die Stelle, bis zu welcher die Änderung vorgenommen wird, durch $ markiert.

Kommando	**Bedeutung**
a	Texteingabe nach dem Cursor
A	Texteingabe nach dem Ende der aktuellen Zeile
i	Texteingabe vor dem Cursor
I	Texteingabe vor dem Anfang der aktuellen Zeile
o	fügt eine Leerzeile (NEWLINE) an die aktuelle Zeile an und geht in den Eingabemodus
O	fügt eine Leerzeile (NEWLINE) vor die aktuelle Zeile an und geht in den Eingabemodus
rc	ersetzt Zeichen unter dem Cursor durch c
R	überschreibt alle Zeichen von der Position des Cursors beginnend
*n***s***zeiket*	ersetzt *n* Zeichen an der aktuellen Cursorposition beginnend durch die Zeichenkette *zeiket*
*n***S**	löscht alle Zeichen in *n* Zeilen, beginnend in der aktuellen und geht in den Eingabemodus
*n***C**	Änderung des Textes von der Cursorposition aus bis an das Zeilenende der *n*-ten Zeile von der aktuellen aus gerechnet
c0	Änderung des Textes vor der Cursorposition bis zum Zeilenanfang
*n***c***x*	Ersetzen von *n* Textobjekten *x* durch neuen Text, wobei für *x* **w** – Wort (Bedeutung entspricht **w**) **W** – Wort (Bedeutung entspricht **W**) **c** – Zeile, entspricht **S** **$** – Zeilenende, entspricht **C** **)** – Satz (Bedeutung entspricht **)**) **}** – Absatz (Bedeutung entspricht **}**) stehen kann

Tabelle 3.5: Kommandos zur Texteingabe

Neben den in Tab.3.5 angeführten Kommandos mit Wiederholungsfaktor n, können auch **a**, **A**, **i**, **I**, **r** und **R** mit einem Wiederholungsfaktor versehen werden. Dies führt zu einer n-fachen Wiederholung des eingegebenen Textes, was wohl relativ selten benötigt wird.

Ein Beispiel für die Verwendung des Wiederholungsfaktors in diesem Zusammenhang ist **2rb**, wodurch das aktuelle Zeichen unter dem Cursor durch bb ersetzt wird, ein weiteres **60a***, was in einer Leerzeile nach der Eingabe von **<esc>** 60 Sterne erzeugt.

3.3 Textbearbeitung

Der Text kann sowohl im Eingabemodus als auch im Befehlsmodus bearbeitet werden.

Im Eingabemodus kann entweder durch BACKSPACE oder **<ctrl h>** ein Zeichen und durch **<ctrl w>** ein Wort gelöscht werden. Die im Ausgleichsspeicher gelöschten Zeichen bzw. Wörter bleiben so lange sichtbar, bis sie entweder überschrieben oder durch **<esc>** die Texteingabe beendet wird.

Die wichtigsten Kommandos und einige Beispiele zum Löschen bzw. Ersetzen von Textteilen im Befehlsmodus sind in der nachfolgenden Tabelle 3.6 zusammengefaßt.

Kommando	**Bedeutung**
n**x**	löscht n Zeichen von der Cursorposition nach rechts
n**X**	löscht n Zeichen vor der Cursorposition
d0	löscht den Text vor der Cursorposition bis zum Zeilenanfang
D	löscht den Text von der Cursorposition bis zum Zeilenende
d/*zeiket*	löscht den Text bis zum ersten Auftreten von *zeiket*
d/*zeiket*/**+0**	löscht den Text einschließlich der Zeile, in der *zeiket* zuerst auftritt
d/*zeiket*/±**n**	löscht den Text bis (−**n**) Zeilen vor dem ersten Auftreten von *zeiket* bzw. bis (+**n**) Zeilen danach
dG	löscht den Text von der aktuellen bis zur letzten Zeile
dn**G**	löscht den Text von der aktuellen bis zur n-ten Zeile
dn**H**	löscht den Text von der aktuellen bis zur n-ten Bildschirmzeile von oben
dn**L**	löscht den Text von der aktuellen bis zur n-ten Bildschirmzeile von unten
n**d**x	Löschen von n Textobjekten x, wobei für x **w** – Wort (Bedeutung entspricht **w**) **W** – Wort (Bedeutung entspricht **W**) **d** – Zeile **$** – Zeilenende **)** – Satz (Bedeutung entspricht **)**) **}** – Absatz (Bedeutung entspricht **}**) stehen kann

Tabelle 3.6: Kommandos zum Löschen von Text

Zum Kopieren von Text dient das Kommando **y** (**yank**) (→ Tab.3.7). Der Text wird dabei in einen namenlosen Speicher abgelegt, der auch den zuletzt gelöschten Textteil zwischenspeichert. Die Syntax und die Textobjekte *x* entsprechen denjenigen der **c**- und **d**-Kommandos.

Der Nachteil des namenlosen Speichers ist, daß er immer wieder durch gelöschte Textteile überschrieben wird und der Speicherinhalt beim Dateiwechsel (→ S.24) verloren geht.

Diese Nachteile werden durch die Verwendung von Speichern mit den Namen a-z vermieden. Der Zugriff auf diese Speicher erfolgt mit Hilfe von **"** (**Anführungszeichen**). Um in den Speicher zu schreiben, wird zuerst der Speichername und dann die Operation (**y**, **d**) angegeben. So werden z.B. durch **"x5yy** fünf Zeilen im Speicher x und durch **"ay0** der Text vor der aktuellen Cursorposition bis zum Anfang der Zeile im Speicher a, gespeichert.

Kommando	**Bedeutung**
"by0	speichert den Text vor der Cursorposition bis zum Zeilenanfang im Speicher **b**
n"**by***x*	speichert im Speicher **b** *n* Textobjekte *x*, wobei für *x* **w**, **W**, **y**, **$**, **)** und **}** (siehe **c** und **d**) stehen kann
"bp	schreibt den gespeicherten Text aus dem Speicher **b** hinter die aktuelle Zeile
"bP	schreibt den gespeicherten Text aus dem Speicher **b** vor die aktuelle Zeile
y0	speichert den Text vor der Cursorposition bis zum Zeilenanfang im namenlosen Speicher
*n***y***x*	speichert im namenlosen Speicher *n* Textobjekte *x*, wobei für *x* **w**, **W**, **y**, **$**, **)** und **}** (siehe **c** und **d**) stehen kann
p	schreibt den gespeicherten Text aus dem namenlosen Speicher hinter die aktuelle Zeile
P	schreibt den gespeicherten Text aus dem namenlosen Speicher vor die aktuelle Zeile

Tabelle 3.7: Speicherbefehle

Zum Auslesen des Speicherinhalts dienen die Kommandos **p** und **P**. Um den im Speicher x gespeicherten Text hinter die aktuelle Zeile zu schreiben, ist **"xp**, um den im Speicher a gespeicherten vor die aktuelle Zeile auszugeben, **"aP**, einzugeben.

Kommando	Bedeutung
u	macht letzte Änderung rückgängig
U	macht alle Änderungen in einer Zeile rückgängig
*n***J**	verbindet *n* Zeilen zu einer
.	wiederholt letzten Befehl
~	macht aus Klein- Großbuchstaben und umgekehrt
<ctrl g>	gibt Informationen über die gerade bearbeitete Datei aus
n>>	verschiebt *n* Zeilen nach rechts
n<<	verschiebt *n* Zeilen nach links
>**G**	verschiebt alle Zeilen von der aktuellen bis zum Dateiende nach rechts
<**H**	verschiebt alle Zeilen von der aktuellen bis zum Bildschirmanfang nach links
>**L**	verschiebt alle Zeilen von der aktuellen bis zum Bildschirmende nach rechts
> /*zeiket*	verschiebt alle Zeilen, von der aktuellen bis zu derjenigen, in der zuerst *zeiket* vorkommt, nach rechts

Tabelle 3.8: Weitere **vi**-Befehle

Der Befehl **U** gilt nur, solange der Cursor sich in der veränderten Zeile befindet.

Bei der Eingabe von **J** ist die Cursorposition unwichtig. Das Auftrennen einer Zeile in zwei geschieht an einfachsten durch **r<return>**. Dadurch wird das Zeichen unter dem Cursor durch RETURN ersetzt.

Zur Wiederholung von Kommandos dient der Punkt. Wird z.B. nach **cw** an einer anderen Stelle **.** eingegeben, so wird an dieser Stelle die vorhergehende Änderung wiederholt.

Die Wirkung von >> und << hängt von dem Attribut **shiftwidth** ab. Es kann durch

:set shiftwidth

abgefragt und durch

:set shiftwidth=4

z.B. auf 4 gesetzt werden. Mit **8** >> werden dann z.B. 8 Zeilen einschließlich der aktuellen um 4 Stellen nach rechts verschoben.

3.3.1 Global wirkende Kommandos

Mit Hilfe der **ex**-Kommandos können in einer Datei globale Veränderungen vorgenommen werden. So können z.B. alle Vorkommen von „Unix“ gegen „UNIX“ ausgetauscht, oder in allen Zeilen ein %-Zeichen am Zeilenanfang eingefügt werden.

Zur Adressierung der Zeilen können unterschiedliche Muster einschließlich Metazeichen verwendet werden. Generell wird der Bereich zwischen der M-ten und N-ten Zeile durch

M,N

angegeben. Die Adressierung kann auch durch die Eingabe von Zeichenmustern erfolgen, z.B. wird durch

/DOS/,/UNIX/

der Bereich zwischen der ersten Zeile die DOS enthält bis zur ersten die UNIX enthält, adressiert. Zur Identifikation der Zeilen können außerden zwei spezielle Zeichen verwendet werden.

1. **.** steht für die aktuelle Zeile,
2. **$** für die letzte Zeile der Datei.

So wird z.B. durch

.+3,$-1

der Bereich 3 Zeilen von der aktuellen bis eine Zeile vor der letzten adressiert. Zur Adressierung können auch Positionen, die durch **m** festgelegt wurden, verwendet werden.

'a,'b

gibt z.B. den Bereich zwischen den Zeilen an, die mit a und b markiert wurden.

Adressierung	**Bereich**
5,20	von der 5-ten bis zur 20-ten Zeile
.,/wort/	von der aktuellen bis zur ersten Zeile die **wort** enthält
10,/wort/-1	von der zehnten bis eine Zeile vor der Zeile, die **wort** enthält
?wort?,.+1	von der letzten Zeile die **wort** enthält, bis eine Zeile nach der aktuellen
.,.+5	die aktuelle Zeile und fünf weitere
.-1,.+1	eine Zeile vor bis eine Zeile nach der aktuellen

Tabelle 3.9: Beispiele zur Adressierung

Mit Hilfe der Bereichsangabe kann in einem bestimmten Teil der Datei eine Substitution vorgenommen werden. Die Substitution einer Zeichenkette durch eine andere sieht allgemein wie folgt aus:

:*M,N***s/***gesuchte_zeichenkette*/*neue_zeichenkette*/**g**

Fehlt die Angabe von **g** (global), so wird in den Zeilen *M-N* nur das erste Auftreten von *gesuchte_zeichenkette* durch *neue_zeichenkette* ersetzt. Um unerwünschte Ersetzungen bei zusammengesetzten Worten zu vermeiden, kann durch

:*M,N***s/\<***gesuchte_zeichenkette***\>/***neue_zeichenkette***/g**

der Wortanfang und das Wortende der *gesuchte_zeichenkette* festgelegt werden. Kommen z.B. im Text die Wörter „Cursor“ und „Cursorbewegungen“ vor, so würden durch

:%s/Cursor//g

alle Vorkommen von „Cursor“ gelöscht, wodurch bei den zusammengesetzten Worten jeweils „bewegungen“ stehen bleiben würde. Bei

:%s/\<Cursor\>//g

werden nur die Worte „Cursor“ gelöscht, „Cursorbewegungen“ bleiben erhalten.

Zur Wiederholung einer Substitution dient **&**, nicht **.** !

Wird das & in *neue_zeichenkette* verwendet, so steht es für *gesuchte_zeichenkette*.

:s/Betriebssystem/& UNIX

ersetzt z.B. in der aktuellen Zeile „Betriebssystem“ durch „Betriebssystem UNIX“.

So wie **g** für einen globalen Ersatz in einer Zeile dient, so dient **%** einem Ersatz in allen Zeilen der Datei.

:%s/Betriebssystem/& UNIX/g

ersetzt in allen Zeilen der Datei jedes Auftreten von „Betriebssystem“ durch „Betriebssystem UNIX“.

Zur globalen Substitution einer Zeichenkette dient das Kommando

:g/*gesuchte_zeichenkette***/s//***neue_zeichenkette***/g**

Soll nicht jedes Vorkommen von *gesuchte_zeichenkette* ersetzt werden, so kann durch das Anfügen von **c** eine Abfrage initialisiert werden.

:g/*gesuchte_zeichenkette***/s//***neue_zeichenkette***/gc**

Bei der Ausführung des Kommandos wird jeweils die zu ersetzende Stelle durch ^ markiert. Soll *gesuchte_zeichenkette* ersetzt werden, so ist dies durch die Eingabe von

y<return> zu bestätigen, falls nicht, so zeigt **<return>** das nächste Vorkommen von *gesuchte_zeichenkette* an.

Eine besondere Rolle beim Suchen haben die Zeichen ^ und **$**.

Beim Suchen bedeutet ^ Zeilenanfang. Da das erste Zeichen in der Zeile in der Spalte 1 steht, verweist ^ auf die Position 0. Der Ersatz von ^ entspricht folglich dem Einfügen von *neue_zeichenkette* am Zeilenanfang.

:%s/^/%

fügt in allen Zeilen am Zeilenanfang ein % ein (das abschließende / hinter **%** kann dabei entfallen).

Die gleiche Bedeutung für das Zeilenende hat **$**. Es handelt sich dabei nicht um das letzte Zeichen in der Zeile, sondern um ein imaginäres weiteres. Der Ersatz von **$** führt daher zu einem Anfügen von *neue_zeichenkette* an das Zeilenende.

:%s/$/.

fügt an jede Zeile einen Punkt an.

Sonderzeichen	**Bedeutung beim Suchen**
a	hebt spezielle Bedeutung des Zeichens *a* auf
^	Zeilenanfang
$	Zeilenende
\<	Wortanfang
\>	Wortende
*	beliebige Anzahl (0,1,...) von Wiederholungen des vorhergehenden regulären Ausdrucks
.	ein beliebiges Zeichen außer NEWLINE
[*menge*]	ein beliebiges Zeichen aus *menge*
[^*menge*]	ein beliebiges Zeichen, außer NEWLINE, das nicht in *menge* vorkommt
\(*menge*\)	*menge* wird in *neue_zeichenkette* verwendet
&	*gesuchte_zeichenkette* zur Substitution in *neue_zeichenkette*
n	*n*-ter Ausdruck, der durch \(und \) eingeschlossen wurde

Tabelle 3.10: Sonderzeichen (reguläre Ausdrücke) des **ex** (**ed**)

Zum Einfügen eines %-Zeichens an den Zeilenanfang kann auch die Kombination

:%s/\(.*\)/%\1

verwendet werden, da im ersten **\(-\)**-Klammerausdruck die gesamte Zeile (**.*** bedeutet ja eine beliebige Anzahl beliebiger Zeichen) abgelegt und durch **\1** (die **1** bezieht sich auf den ersten (und hier einzigen) Klammerausdruck, der in *gesuchte_zeichenkette* vorkommt) hinter **%** angefügt wird.

Für *menge* in [,], kann eine Zeichenfolge, z.B. [**abc**], oder ein Bereich, z.B. [**A-Z**], angegeben werden.

:%s/[1-9][0-9]*/(&)/g

setzt z.B. alle Zahlen im Text in runde Klammern.

In der nachfolgenden Tabelle 3.11 sind weitere nützliche **ex**-Kommandos zusammengestellt.

Kommando	**Bedeutung**
:e *dateiname*	Übergang in die Datei *dateiname*
:e! *dateiname*	Übergang in die Datei *dateiname*, ohne Berücksichtigung von Änderungen in der aktuellen Datei
:e!	neues Einlesen der aktuellen Datei von der Festplatte
:e#	Wechsel in die vorhergehende Datei
:n	Wechsel in die nächste Datei beim gleichzeitigen Editieren mehrerer Dateien
:rew	Zurücksetzen der Bearbeitungsfolge auf die erste Datei, falls gleichzeitig mehrere Dateien editiert werden
:r *dateiname*	Einlesen der Datei *dateiname* hinter die aktuelle Zeile
:-r *dateiname*	Einlesen der Datei *dateiname* vor die aktuelle Zeile
:r! *befehl*	Einlesen der Ausgabe des UNIX-Kommandos *befehl* unter die aktuelle Zeile
:! *befehl*	Ausführung des UNIX-Kommandos *befehl* in der SHELL
:sh	Start einer neuen SHELL, die Rückkehr in den **vi** erfolgt durch **<ctrl d>**
:w *dateiname*	Abspeichern des Inhalts des Ausgleichsspeichers in der Datei *dateiname*
:w! *dateiname*	Überschreiben der Datei *dateiname* mit dem Inhalt des Ausgleichsspeichers
:12,25w *dateiname*	Abspeichern der Zeilen **12-25** in der Datei *dateiname*
:w >>*dateiname*	Anfügen des Inhalts des Ausgleichsspeichers an die Datei *dateiname*
:q	Beenden des **vi**
:q!	Beenden des **vi** ohne Berücksichtigung von Änderungen

Tabelle 3.11: Weitere **ex**-Kommandos

3.3.1.1 Makros und Abkürzungen

Zur bequemeren Arbeit mit immer wieder auftretenden Befehlsfolgen können diese in Makros zusammengefaßt werden. Die allgemeine Definition lautet:

:map *zeichen kommandofolge*

Für *zeichen* sollten Zeichen verwendet werden, die im Befehlsmodus nicht vorkommen. Diese sind:

**g, q, v, K, V, <ctrl a>, <ctrl k>, <ctrl o>, <ctrl t>, <ctrl w>, <ctrl x>, <ctrl z>, *, _, **

Möchte man zwei aufeinanderfolgende Worte in der Reihenfolge vertauschen, so kann dies durch **dW**, **e**, **l** und **p** geschehen. Diese Befehlsfolge kann durch

:map v dWelp

unter **v** abgespeichert werden.

P eter's Kaffee ist heiß. $\xrightarrow{\mathbf{v}}$ Kaffee Peter's□ist heiß.

Soll das Speichern in einem der CTRL-Zeichen erfolgen, so muß diesem in der Regel **<ctrl v>** vorangestellt werden.

:map <ctrl v><ctrl t> dd

speichert unter **<ctrl t>** das Kommando **dd** zum Löschen der aktuellen Zeile.

Möchte man in der Befehlsfolge RETURN, ESC, TAB, BACKSPACE oder DELETE verwenden, so muß diesen ebenfalls **<ctrl v>** vorangestellt werden.

Beim Aufruf externer Kommandos mittels **!** steht der Name der gerade editierten Datei unter % zur Verfügung. Mit **‘basename % .c‘** wird von *dateiname***.c** der Suffic **.c** entfernt (→ on-line-manual).

Für **^M** (Wiedergabe auf dem Bildschirm) ist jeweils **<ctrl v><return>** einzugeben.

:map v :w^M:!cc % -o ‘basename % .c‘^M:!‘basename % .c‘^M

Nach Eingabe von **v** wird durch die vorhergehende Definition zuerst die aktuelle Datei gespeichert, dann der C-Compiler aufgerufen, das Ergebnis nach *dateiname* geschrieben und diese Datei dann ausgeführt. Während der Ausführung bleibt der **vi** aufgerufen, so daß eventuelle Korrekturen sofort durchgeführt werden können.

Die Freigabe der Makros erfolgt durch

:unmap *zeichen*

Zur Abkürzung von Befehlsfolgen kann auch anders vorgegangen werden. Man schreibt zuerst die gewünschte Befehlsfolge in eine leere Zeile und löscht diese anschließend durch

"jdd

z.B. in den Speicher j. Der Befehlsaufruf erfolgt dann durch

@j

Zur Abkürzung von Textpassagen dient

:ab *abkürzung lange_zeichenfolge*

Zur Abkürzung von „Betriebssystem UNIX“ kann z.B. die Abkürzung „uu“ verwendet werden.

:ab uu Betriebssystem UNIX

Sobald vor **uu** ein Leerzeichen ist oder in den Eingabemodus geschaltet wurde und anschließend ein Leerzeichen, **<esc>** oder **<return>** folgt, wird **uu** durch „Betriebssystem UNIX“ ersetzt.

Die Abkürzung kann durch

:unab *abkürzung*

wieder freigegeben werden.

3.3.1.2 Anpassung der Arbeitsumgebung

Der **vi** wird durch sogenannte Attribute eingestellt. Ein Beispiel für solch ein Attribut wurde auf Seite 20 (**shiftwidth**) angeführt.

Die Attribute können ein- oder ausgeschaltet sein. Zum Einschalten dient

:set *attribut_name*

zum Ausschalten

:set no*attribut_name*

Manchen Attributen kann auch ein Wert (Zahl oder Zeichenkette) zugewiesen werden.

:set *attribut_name=wert*

Mit

:set all

werden alle Attribute mit ihren Werten ausgegeben. Mit

:set

werden diejenigen ausgegeben, deren Voreinstellungen geändert wurden.

Einige nützliche Attribute sind in Tabelle 3.12 zusammengestellt.

Sollen bei jedem Aufruf des **vi** von der Voreinstellung abweichende Attribute verwendet werden, so müssen die Änderungen in eine Datei namens **.exrc** eingetragen werden.

Ein Eintrag zur Einstellung des automatischen Einrückens, der Zeilennumerierung und des Zeilenumschaltens bei 8 Leerzeichen vor Zeilenende, kann wie folgt aussehen:

set autoindent number wrapmargin=8

Die Datei **.exrc** kann neben **set**-Anweisungen auch **map**- und **ab**-Definitionen enthalten. Sie muß zum Lesen im **$HOME**-Verzeichnis des Benutzers abgelegt sein.

Attribut	**Abkürzung**	**Bedeutung**
autoindent	**ai**	automatisches Absetzen der Texteingabe (nützlich beim Programmieren); Rückkehr um **shiftwidth** nach links durch **<ctrl d>**, an den Zeilenanfang durch **0<ctrl d>**
number	**nu**	Zeilennummerierung
wrapmargin	**wm**	ist **wrapmargin=**n, so erfolgt der Übergang in eine neue Zeile, sobald zum rechten Bildschirmrand der Abstand n Leerzeichen beträgt
showmode	**shm**	zeigt am unteren Bildschirmrand den Editormodus an: **Befehl**: **Information** **i,I** : INSERT MODE **a,A** : APPEND MODE **o,O** : OPEN MODE **C,s** : CHANGE MODE **r** : REPLACE 1 CHAR **R** : REPLACE MODE **S** : INPUT MODE
showmatch	**sm**	zeigt eine korrespondierende Klammer im Eingabemodus an

Tabelle 3.12: Nützliche Attribute des **vi**

3.4 History-Mechanismus der Korn-SHELL

Die Korn-SHELL (**ksh**) ist die Standard-SHELL heutiger UNIX-Systeme. Sie verbindet alle Eigenschaften der C- und TC-SHELL, ihre Programmiersprache ist zu derjenigen der Bourne-SHELL kompatibel. Um herauszufinden, welche SHELL benutzt wird, ist

$ echo $SHELL

einzugeben, durch

$ ksh

wird die Korn-SHELL gestartet.

Die **ksh** hat einen leicht handhabbaren History-Mechanismus, der es erlaubt, frühere SHELL-Kommandos wiederzuverwenden und gegebenenfalls mit Hilfe des **vi** zu modifizieren.

Durch den Eintrag

set -o vi

in die Datei **.profile**, wird der **vi** als Editor zur Modifikation der Eingabezeile gewählt. Falls die **ksh** nur vorübergehend verwendet wird, kann das **set**-Kommando auch in der SHELL eingegeben werden.

$ set -o vi

Durch die Eingabe

$ history

wird die Liste früherer Befehle angezeigt, der voreingestellte Wert ist dabei 128. Die Liste kann durch den Eintrag

HISTSIZE=50; export HISTSIZE

im **.profile** auf 50 Einträge beschränkt werden. Beim Verlassen des Systems wird die Liste in der Datei **.sh_history** gespeichert, die Kommandos stehen folglich beim nächsten Anmelden (login) wieder zur Verfügung.

Zur Wiederholung eines Kommandos aus der History-Liste dient der Befehl **r** (→ Tab.-3.13).

Kommando	**Bedeutung**
r	Wiederholung des letzten Befehls
r *n*	Wiederholung des Befehls mit der Nummer *n*
r *zeiket*	Wiederholung des ersten Befehls aus der History-Liste, der mit der Zeichenkette *zeiket* beginnt

Tabelle 3.13: Kommandos zur Wiederholung von Befehlen aus der History-Liste

Mit **r** wird der geholte Befehl sofort ausgeführt. Dies ist bei fehlerhaft eingegebenen Befehlen sicher nicht erwünscht.

Möchte man den Befehl vor der Ausführung nochmals bearbeiten, so muß die Eingabezeile aus dem Eingabemodus durch **<esc>** in den Befehlsmodus umgeschaltet werden.

Zum Holen eines Kommandos aus der History-Liste in die Eingabezeile dienen die in der Tabelle 3.14 zusammengefaßten **vi**-Befehle. Zur Bearbeitung der Eingabezeile stehen die meisten in diesem Abschnitt besprochenen **vi**-Befehle zur Verfügung.

Kommando	**Bedeutung**
*n***k**	bewegt den Cursor um *n* Stellen in der History-Liste zurück und holt das dort eingetragene Kommando in die Eingabezeile
*n***G**	holt das *n*-te Kommando aus der History-Liste
/*zeiket*	sucht rückwärts nach dem ersten Auftreten der Zeichenkette *zeiket* und holt das Kommando in die Eingabezeile, reguläre Ausdrücke sind dabei für *zeiket* nicht erlaubt, / und **?** arbeiten umgekehrt, als beim **vi**
n	Wiederholt die Suche in gleicher Richtung
N	Wiederholt die Suche in entgegengesetzter Richtung

Tabelle 3.14: Kommandos zum Suchen von Befehlen in der History-Liste

3.5 Kommandoübersicht

Verschiebung des Cursors[1]		**Texteingabe**[1]	
h	um eine Stelle nach links	**a**	rechts vom Cursor
l	um eine Stelle nach rechts	**A**	am Zeilenende
k	um eine Zeile nach oben	**i**	links vom Cursor
j	um eine Zeile nach unten	**I**	am Zeilenanfang
^	zum Zeilenanfang[2]	**o**	nach aktueller Zeile
n-	**n** Zeilen zurück	**O**	vor aktueller Zeile
n	**n** Zeilen vorwärts	**r**	ein Zeichen ersetzen
b	um ein Wort nach links[5,3]	**R**	alle Zeichen überschreiben
B	um ein Wort nach links[5,4]	**c***x*	Textänderung x[7]
w	um ein Wort nach rechts[5]	**C**	Textänderung bis Zeilenende
W	um ein Wort nach rechts[5]	**s***zeiket*	Ersatz eines Zeichens durch *zeiket*
e	um ein Wort nach rechts[6]	**S**	Überschreiben der aktuellen Zeile
E	um ein Wort nach rechts[6]	**y***x*	Textobjekte x[7] speichern
)	um einen Satz vorwärts	**Wiederholungen und spez. Kommandos**	
(	um einen Satz rückwärts	**;**	wiederholt **f**,**F**,**t**,**T**
}	um einen Absatz vorwärts	**,**	wiederholt **f**,**F**,**t**,**T** in umgek. R.
{	um einen Absatz rückwärts	**n**	wiederholt /, **?**
]]	um ein Kapitel vorwärts	**N**	wiederholt /, **?** in umgekehrter R.
[[	um ein Kapitel rückwärts	**.**	wiederholt letztes Kommando
<ctrl u>	um halbe Seite zurück	**u**	macht letzte Änderung rückgängig
<ctrl d>	um halbe Seite vorwärts	**U**	macht alle Ändn. i. d. Z. rückgng.
<ctrl b>	um eine Seite zurück	**%**	verschiebt Cursor auf kor. Klammer
<ctrl f>	um eine Seite vorwärts	**~**	macht aus Gr.-Klbstn. u. umgek.
?*zeiket*	rückwärts auf *zeiket*	**Spezielle Zeichen im :s-Modus**	
/*zeiket*	vorwärts auf *zeiket*	**^**	Zeilenanfang
F*c*	auf Zeichen c nach links	**$**	Zeilenende
f*c*	auf Zeichen c nach rechts	**\<**	Wortanfang
T*c*	nach links vor Zeichen c	**\>**	Wortende
t*c*	nach rechts vor Zeichen c	**&**	gesuchte Zeichenkette[8]
0	zum Zeilenanfang	[*menge*]	ein Zeichen aus *menge*
$	an das Zeilenende	[^*menge*]	ein Zeichen nicht aus *menge*
H	zum Bildschirmanfang	**\(..\)**	zum Ersetzen in neuer Zeichenkette
L	zum Bildschirmende	**\n**	**n**-ter Ausdr. in **\(..\)**-Kl.
1G	auf erste Zeile	**.**	beliebiges Zeichen
G	auf letzte Zeile	*****	bel. Wiederhlng. (0,1,...) des
Löschen[1]			vorhergehenden Ausdrucks
x	aktuelles Zeichen	**Speicher**[9]	
X	Zeichen links vom aktuellen	**"a**	26 Speicher **"a-"z**
d0	zum Zeilenanfang	**Rückkehr zu markierter Stelle** [10]	
d*x*	Textobjekte x[7]	**'x**	Zeilenanfang
D	zum Zeilenende	**`x**	markierte Stelle

Abbildung 3.1: Kommandos des Befehls- und Eingabemodus

[1] Außer bei den Befehlen **0**, **D**, **o** und **O** kann immer ein Wiederholungsfaktor n angegeben werden. **3h** verschiebt z.B. den Cursor um drei Zeichen nach links und **4dw** löscht 4 Wörter.

[2] Auf das erste Zeichen der Zeile das nicht Leer- oder Tabulatorzeichen ist.

[3] Für Kleinbuchstaben ist ein Wort eine Folge von Buchstaben, Zahlen und _.

[4] Für Großbuchstaben ist ein Wort eine beliebige Zeichenfolge, begrenzt durch Leerzeichen.

[5] auf den Wortanfang

[6] auf das Wortende

[7] x kann **w**, **W**, [**c**, **d**, **y**], **$**, **)**, **}**, sein

[8] Z.B. setzt **:%s/**[**1-9**][**0-9**]***/(&)** alle Zahlen im Text in runde Klammern. Im Befehlsmodus wiederholt **&** die letzte Substitution, die durch **:s** erfolgte.

Start des vi	
vi *dateiname*	Editieren oder Erzeugen der Datei *dateiname*, der Cursor ist auf der ersten Zeile
vi +n *dateiname*	der Cursor ist auf der **n**-ten Zeile
vi + *dateiname*	der Cursor ist auf der letzten Zeile
vi +/*zeiket dateiname*	der Cursor ist auf der Zeile, die *zeiket* enthält
vi -r *dateiname*	Erneuerung der Datei bei unkorrektem Beenden des **vi**
Speichern in Datei	
:w	Speichern des aktuellen Ausgleichsspeicherinhalts
:w *dateiname*	Speichern in *dateiname*
:w! *dateiname*	Überschreiben von *dateiname*
:w >> *dateiname*	Anhängen des Ausgleichsspeicherinhalts an *dateiname*
Beenden des vi	
:q	Beenden
:q!	Beenden, ohne Berücksichtigung von Änderungen
:wq	Speichern und Beenden
:x oder **ZZ**	Beenden mit Speichern
:! *befehl*	Ausführen des UNIX-Kommandos *befehl*
Wechsel zwischen Dateien	
:e *dateiname*	Wechsel nach *dateiname* ohne den **vi** zu verlassen
:e! [*dateiname*]	[Wechsel] oder neues Laden der aktuellen Datei, ohne Berücksichtigung von Änderungen
:e#	Rückkehr in vorherige Datei
:rew	Zurücksetzen auf erste Datei
Speichern von Dateibereichen[11]	
:*M,N***w** *dateiname*	speichert die Zeilen *M* bis *N* nach *dateiname*
:*M,N***w** >> *dateiname*	hängt die Zeilen *M* bis *N* an *dateiname* an
Einlesen von Dateien	
:-r *dateiname*	liest Datei *dateiname* vor die aktuelle Zeile
:r *dateiname*	liest Datei *dateiname* hinter die aktuelle Zeile
:-r! *befehl*	die Ausgabe von *befehl* wird vor die aktuelle Zeile eingelesen
:r! *befehl*	die Ausgabe von *befehl* wird hinter die aktuelle Zeile eingelesen
Makros und Abkürzungen	
:map[12] *zeichen kommandofolge*	*zeichen* kann **<ctrl [a,k,o,t,w,x]>**, **g**, **q**, **v**, **K**, **V** sein
:ab *abkürzung zeichenkette*	*abkürzung* wird nach Leerzeichen, ESCAPE oder RETURN durch *zeichenkette* ersetzt

Abbildung 3.2: Starten, Beenden und Anpassen des **vi**

[9] In den Speicher **a** werden z.B. drei Zeilen mittels **"a3yy** geschrieben. Durch **"ap** werden sie hinter, durch **"aP** vor die aktuelle Zeile zurückgeschrieben.

[10] Die aktuelle Cursorposition wird durch **m** (**mark**), z.B. **mx**, markiert. Zur Rückkehr an die markierte Stelle dient **'x**, zur Rückkehr auf die Zeile, in der die Markierung **x** erfolgte, **'x**.

[11] *M* und *N* können Zahlen, Zeichenketten */zeiket/* bzw. **?***zeiket***?**, der Punkt **.** (aktuelle Zeile) oder **$** (letzte Zeile) sein. Die Eingabe von **+n** oder **-n** (**n** ganze Zahl) hinter der Adresse führt zu einer Verschiebung des Adreßbereichs um **n** Zeilen.

[12] Vor <**ctrl**>, ESCAPE, TAB, BACKSPACE, DELETE oder RETURN muß <**ctrl v**> geschrieben werden. Eine andere Möglichkeit zur Bildung und Verwendung von Abkürzungen für Kommandofolgen ist **"jdd** (Befehlsfolge in den Speicher **j** löschen) und **@j** (Befehlsfolge aus dem Speicher **j** aufrufen).

3.6 Übungen

Übung 1 *Geben Sie das Kommando zum Auffinden der Zeichenkette .* an.*

Übung 2 *Die 20-te Zeile soll zur ersten Bildschirmzeile werden. Was müssen Sie dazu eingeben?*

Übung 3 *Sie möchten das Wort* **Bilder** *in* **Bildschirm** *verändern. Geben Sie den einfachsten Weg dazu an.*

Übung 4 *Mit welchem Befehl können Sie die ersten drei Wörter dieses Satzes nach x speichern um sie anschließend in die Datei* **uebungen** *an den Anfang der 20-te Zeile zu schreiben.*

Übung 5 *Wie kann das heutige Datum hinter die aktuelle Zeile eingelesen werden?*

Übung 6 *Wie können Sie alle Vorkommen von* **dies** *in* **das** *ändern, ohne das z.B.* **diesmal** *mit geändert wird?*

Übung 7 *Sie möchten alle Zeilen, die in Spalte 1 nicht mit einer Zahl beginnen, in geschweifte Klammern setzten. Wie lautet das dazu notwendige Kommando?*

4 Dateien

Beim Entwurf von UNIX sind die Autoren hinsichtlich der Organisation des Dateisystems von drei Forderungen ausgegangen:

- Jede Datei muß eindeutig und schnell identifizierbar sein,
- sie erhält auf der Festplatte gerade so viel Platz, wie sie benötigt,
- die Arbeit mit den physikalischen Einrichtungen (**devices**) erfolgt mit den gleichen Befehlen, wie sie für Dateien verwendet werden.

Daraus folgt die Einteilung der Dateien in

1. gewöhnliche Dateien (**files**),
2. Verzeichnisse (**directories**) und
3. spezielle Dateien (**device drivers**).

Eine Datei ist eine strukturlose Folge von Zeichen. Mehrere Dateien bilden zusammen ein Dateisystem (**filesystem**). Dieses kann z.B. auf einer magnetischen Festplatte liegen. Der Zugriff auf dieses Dateisystem erfolgt mittels Systembefehlen.

Jede Datei im Dateisystem wird durch ihren i-node (**index node**) repräsentiert (→ Abb.4.1). Der i-node enthält alle Attribute sowie Verweise auf den Datenbereich der Datei.

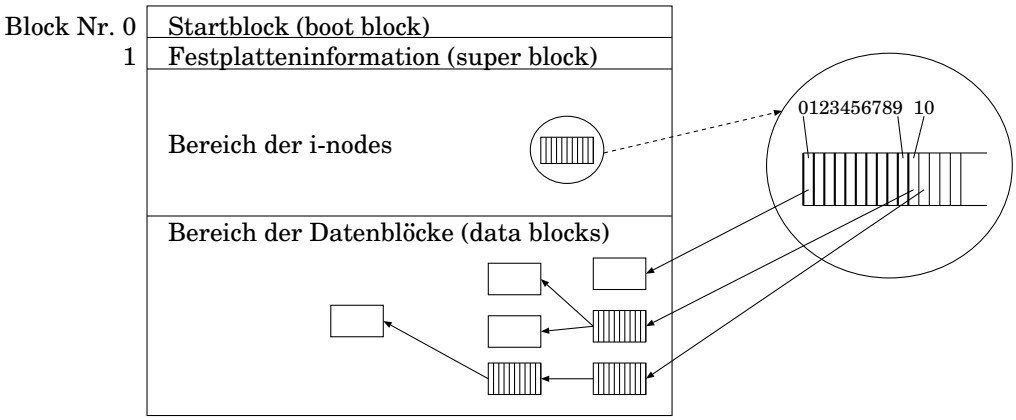


Abbildung 4.1: Struktur des Dateisystems – i-nodes

Die Nummer des i-nodes entspricht seiner Anordnung im Bereich der i-nodes. Jeder Datei ist eindeutig eine Nummer zugeordnet. Die Ausgabe der i-node-Nummern mit den zugehörigen Dateinamen erfolgt mittels

$ ls -i

```
162839 alpha 185856 beta
```

Die wichtigsten Attribute der im aktuellen Verzeichnis stehenden Dateien erhält man durch

$ ls -l # leere Dateien werden nicht mitgezählt (total 1)

```
total 1
-rw-rw-r--   1 leibner  system         0 Jul 10 09:46 alpha
drwxrwxr-x   2 leibner  system       512 Jul 10 10:07 beta

$
```

Jeder i-node enthält unter anderen folgende Attribute:

- Den Dateityp (1-te Spalte):
  - `-` gewöhnliche Datei,
  - `d` Verzeichnis,
  - `l` Verweis (**link**),
  - `c` zeichenorientierte spezielle Datei (**character device**),
  - `b` blockorientierte spezielle Datei (**block device**),
- Zugriffsrechte (2-te bis 10-te Spalte),
- Anzahl der Verweise (**alpha**: 1, **beta**: 2),
- den Eigentümer (leibner) der Datei,
- die Gruppenzugehörigkeit (system),
- Größe der Datei in Byte (**alpha**: 0, **beta**: 512 Byte),
- Datum und Zeitpunkt der letzten Veränderung (**alpha**: 10. Juli, 09:46h),
- Adreßtabelle der Datenblöcke.

In der Adreßtabelle der Datenblöcke (→ Abb.4.1) zeigen die ersten 10 Einträge auf die Datenblöcke direkt, alle weiteren indirekt mit Hilfe einer oder mehrerer weiterer Tabellen. Die Organisation dieser Tabellen hängt jeweils von der Größe des Speichermediums ab. Sie wird während der Formatierung festgelegt.

Insgesamt enthält ein i-node mehr als 40 unterschiedliche Informationen, von denen die meisten (z.B. die Adressen der Datenblöcke) nur für den Systemkern von Interesse sind. Dem Benutzer reichen in der Regel diejenigen Attribute, die durch das Kommando **ls -la** ausgegeben werden.

In jedem Verzeichnis ist für jede darin enthaltene Datei ein Platz von 259 Zeichen (**bytes**) reserviert. Für den Namen der Datei sind es 255 Zeichen, für die Nummer des i-node 4 (→ Abb.4.2).

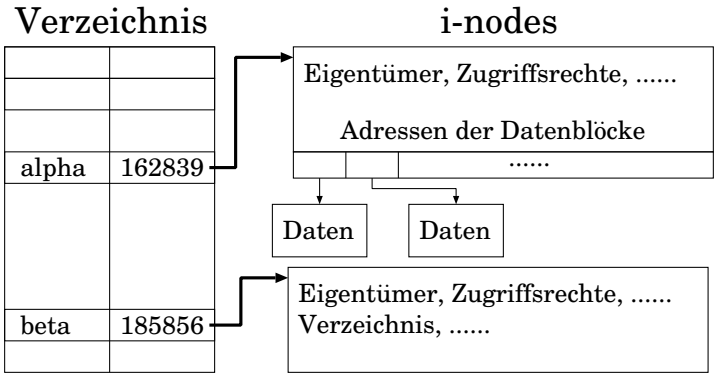


Abbildung 4.2: Verzeichnisbelegung (Dateiname,i-node-Nummer)

Gewöhnliche Datein (z.B. **alpha**) enthalten Benutzerdaten. Ihr Inhalt kann durch

$ cat alpha

auf dem Bildschirm ausgegeben werden.

Das Kommando **cat** liefert nur dann ein sinnvolles Ergebnis, wenn die Datei **alpha** eine Textdatei ist (sie darf nur Zeichen aus dem unteren Teil der ASCII-Tabelle (32-126) enthalten). Binäre Dateien enthalten Zeichen aus der gesamten ASCII-Tabelle, sie sind daher in der Regel nur von speziellen Applikationen oder dem Systemkern lesbar (z.B. ein ausführbares Programm). Der Inhalt einer binären Datei kann mit dem Kommando **od** (**octal dump**) ausgegeben werden.

Ein **Verzeichnis** ist eine binäre Datei mit dem Attribut „Verzeichnis“. Die Ausgabe des Inhalts des aktuellen Verzeichnisses (**Arbeitsverzeichnis**) erfolgt durch

$ od -c .

```
0000000   \0 326 002  \0  \f  \0 001  \0   .  \0  \0  \0 026   | 002  \0
0000020   \f  \0 002  \0   .   .  \0  \0 031 312 002  \0 020  \0 004  \0
0000040    b   e   t   a  \0   s   o  \0 001 326 002  \0 330 001 005  \0
0000060    a   l   p   h   a  \0   7  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000100   \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
*
0001000
```

Der Schalter **-c** bewirkt die Ausgabe aller darstellbarer Zeichen als ASCII-Zeichen (character), sowie die Ausgabe nicht darstellbarer Zeichen durch ihr oktales Äquivalent. Ohne Angabe von **-c** erfolgt die Ausgabe in oktaler, durch Angabe von **-x** in hexadezimaler bzw. durch **-d** in dezimaler Darstellung. Die Schalter können auch kombiniert

werden, **od -cx .** gibt z.B. den Inhalt des aktuellen Verzeichnisses in ASCII und hexadezimal aus. Die in der linken Spalte erscheinenden Zahlen sind oktale Zeichenzähler (z.B. $(0000020)_8 = (16)_{10}$ Zeichen).

Jede Ausgabezeile enthält 16 Zeichen. Beim Betrachten der Ausgabe fällt auf, daß neben **alpha** und **beta** zwei weitere Dateien vorhanden sind. Durch

```
$ ls  -la

total 4
drwxrwxr-x   3 leibner  system       512 Aug 28 11:50 .
drwxrwxr-x   3 leibner  system       512 Jul 10 10:07 ..
-rw-rw-r--   1 leibner  system         5 Aug 28 11:50 alpha
drwxrwxr-x   2 leibner  system       512 Aug 28 11:50 beta

$
```

stellt man fest, daß es sich dabei um Verzeichnisse handelt. Das Verzeichnis **..** ist ein übergeordnetes, **.** das aktuelle Verzeichnis (**.** zeigt auf den i-node des aktuellen Verzeichnisses). Der Aufbau des Dateisystems ist folglich hierarchisch.

Verfolgt man die Hierarchie rückwärts über alle **..**-Verzeichnisse, so gerät man in das Verzeichnis **/** (**root directory**). Dieses Verzeichnis ist die Wurzel des Dateibaums (→ Abb.4.3). Es kann theoretisch eine beliebige Anzahl von Unterverzeichnissen enthalten.

In Bezug auf diesen Dateibaum erhält jeder Benutzer ein Heimatverzeichnis (**home directory**), das nach seiner Anmeldung am System für ihn eingestellt wird. Hier beginnt der Arbeitsbereich des Benutzers. Er kann sich z.B. durch **mkdir** weitere Unterverzeichnisse schaffen oder mit dem **vi** Dateien erzeugen. Mit dem Kommando

```
$ cd  beta
$
```

kann er dann z.B. in das Unterverzeichnis **beta** wechseln (**change directory**). Nach dem Wechsel wird **beta** sein neues Arbeitsverzeichnis (**working directory**). Durch

```
$ pwd

/home/leibner/unix/beta

$
```

erhält er den Namen des Arbeitsverzeichnisses (**print working directory**).

Die Ausgabe „`/home/leibner/unix/beta`“ gibt an, daß der Benutzer sich im Verzeichnis **beta** befindet, dessen übergeordnetes Verzeichnis **unix**, dessen übergeordnetes **leibner** und dessen übergeordnetes wiederum **home** ist. Der Dateibaum endet mit dem Wurzelverzeichnis `/`. Die Ausgabe gibt den Weg (**path**) vom Wurzelverzeichnis zum Arbeitsverzeichnis wieder. Die alleinige Eingabe von

```
$ cd
$
```

stellt das Heimatverzeichnis als Arbeitsverzeichnis ein.

```
$ pwd

/home/leibner

$
```

Zum Wechsel des Arbeitsverzeichnisses können auch **.** und **..** benutzt werden, z.B.

```
$ cd  ..
$ pwd

/home

$
```

In Abbildung 4.3 ist bei jedem Verzeichnis sein Inhalt schematisch wiedergegeben. Beim Wurzelverzeichnis beziehen sich **.** und **..** auf denselben i-node.

Verzeichnis	i-node-Nummer
/	2
/bin	12
/home	13
/home/leibner	126
/home/leibner/dti	21632
/home/leibner/vzs	3074

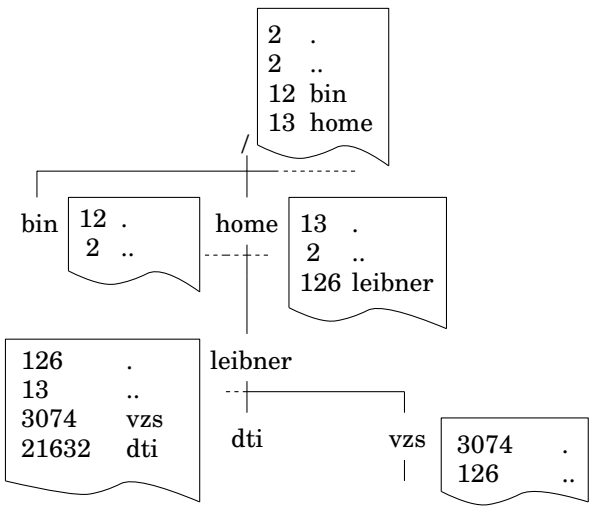


Abbildung 4.3: Fragment des Adreßbaums

Der Zugang zur Rechnerperipherie erfolgt mittels **spezieller Dateien**. Sie sind standardmäßig unter **/dev** abgelegt. Ihre i-node-Belegung entspricht bis auf die Angabe der Größe derjenigen der gewöhnlichen Dateien.

```
$ ls  -l  /dev/lpr

c-w--w--w-   1 root     system     2,0  2 Aug 28 14:30 lp
```

Statt der Größe der Datei werden hier zwei durch ein Komma getrennte Zahlen (2,0) angegeben. Die erste wird als Hauptzahl (**major number**) der speziellen Datei, die zweite als Nebenzahl (**minor number**) bezeichnet.Die Hauptzahl identifiziert den Treibertyp (Drucker, Disk, Terminal, usw.), die Nebenzahl unterscheidet verschiedene konkrete Treiber (z.B. mehrere angeschlossene Drucker). Ist z.B. **/dev/lpr** der Druckertreiber, so kann der Benutzer die Datei **alpha** wie folgt auf dem Drucker ausgeben (durch > wird die Standardausgabe (Bildschirm) auf den Druckertreiber umgelenkt):

```
$ cat  alpha  >/dev/lpr
$
```

Diese Vorgehensweise ist in der Regel nur dem Systemverwalter erlaubt. Die Druckerausgabe erfolgt normalerweise durch einen **spooling**-Mechanismus, mit dessen Hilfe das Betriebssystem nach dem Prinzip des FIFO (**First In First Out**) in einem speziellen Verzeichnis zwischengespeicherte Dateien an den Druckertreiber weiterleitet.

Der Zugang zur Rechnerperipherie ist entweder zeichen- oder blockorientiert. Typische blockorientierte Einrichtungen sind Magnetbänder und Disks, zeichenorientiert ist z.B. das Terminal. Ein Block enthält in der Regel 512 Zeichen, bei neueren UNIX-Versionen kann momentan eine Blockgröße von bis zu 8 kB (kilo Byte) eingestellt werden.

Die Organisation des Dateisystems nach Abbildung 4.1 ist nur für kleine Speichermedien sinnvoll. Aufgrund der Zunahme der Speicherkapazität bei neuen Speichermedien wird in der Regel eine Organisationsform gewählt, wie sie in 4.3 BSD vorgeschlagen wurde. Diese Speicherorganisation ist vornehmlich auf die Reduktion von Zugriffszeiten auf Dateien ausgerichtet. Obwohl dazu die Bereiche der Festplatteninformation und der i-nodes aufgeteilt wurden, bleibt dabei die logische Struktur des Dateisystems erhalten.

Auf dem Dateisystem, welches das Wurzelverzeichnis enthält, kommt dem Startblock eine besondere Bedeutung zu. Dieser enthält das Programm **boot**, das nach dem Einschalten des Rechners mit Hilfe eines Mikroprogramms (in einem ROM abgelegte Firmware, die den Aufbau des Dateisystems nicht kennt und daher den Systemkern nicht selbst laden kann) in den Hauptspeicher geladen und gestartet wird. Das Programm **boot** lädt danach den Systemkern und starten ihn.

Zwischen dem angeschlossenen Dateisystem (Festplatte) und dem Systemkern ist im Hauptspeicher Speicherplatz für einen Ausgleichsspeicher (**buffer cache**) reserviert. In

diesen Bereich werden durch den Systemkern Datenblöcke vor ihrer Verwendung im Datenbereich des Benutzers im Hauptspeicher zwischengespeichert.

Daten im Ausgleichsspeicher, die durch den Benutzer verändert aber noch nicht auf die Festplatte zurückgeschrieben wurden, können bei Systemfehlern (Stromausfall) verloren gehen. Da im Ausgleichsspeicher nicht nur Datenblöcke sondern auch alle vorhandenen Dateisysteme virtualisiert vorliegen, kann es bei Systemfehlern nicht nur zu Datenverlusten sondern auch zu Inkonsistenzen zwischen den virtualisierten und den reellen Dateisystemen kommen. UNIX sucht den Datenverlusten bzw. Inkonsistenzen durch das Kommando

```
$ sync
```

zuvorzukommen. Es wird üblicherweise vom Systemkern alle 30 Sekunden abgesetzt, kann aber auch direkt von jedem Benutzer eingegeben werden. Durch **sync** werden die angeschlossenen Dateisysteme auf den Stand der virtuellen gebracht.

Zu jedem Dateisystem gehört eine Festplatteninformation. Sie enthält Angaben über

- die Größe des Dateisystems,
- die Anzahl freier Datenblöcke,
- ein Verzeichnis nichtbesetzter Datenblöcke,
- die Adresse des ersten nichtbesetzten Datenblocks,
- die Größe des i-node-Bereichs,
- die Anzahl freier i-nodes,
- ein Verzeichnis nichtbesetzter i-nodes,
- die Adresse des ersten nichtbesetzten i-node,
- ein Feld von Sperren für freie Datenblöcke und i-nodes,
- Zeichen der Modifikation der Festplatteninformation.

Es wird dabei nicht vorausgesetzt, daß ein Dateisystem auf einem einzigen Medium implementiert ist. Der Systemverwalter hat vielmehr die Möglichkeit, z.B. ein Dateisystem auf einer magnetischen Festplatte durch ein weiteres auf einer optischen zu erweitern. Dazu wird die magnetische Festplatte als Dateisystem mit einem Wurzelverzeichnis festgelegt und die optische Platte mit dem Befehl **mount** an ein leeres Verzeichnis angehängt:

```
$ mount  /dev/op1c  /optical
$
```

Durch dieses Kommando wird das Dateisystem mit der speziellen Datei „`/dev/op1c`“ unter das leere Verzeichnis „`/optical`“ gehängt. Die optische Platte ist nun wie jedes andere Verzeichnis im Dateisystem ansprechbar und kann so z.B. zum neuen Arbeitsverzeichnis werden:

```
$ cd   /optical
$
```

Der Systemverwalter kann durch

```
$ umount   /dev/op1c
$
```

die optische Platte wieder aus dem Dateisystem der magnetischen Festplatte entfernen.

4.1 Dateinamen

Die Länge eines Dateinamens kann in neueren UNIX-Versionen 255 Zeichen betragen. Bis auf / (bestimmt den Ort der Datei im Verzeichnisbaum) sind alle Zeichen zugelassen.

Nicht zu empfehlen ist die Verwendung von Sonderzeichen der SHELL, wie z.B. `@`, `$`, `|`, `&`, `*`, `!`, `?`, `[`, `]`, `(`, `)`, `;`, ‘, ’ oder `"`.

UNIX unterscheidet zwischen Klein- und Großbuchstaben. So können die Namen **alpha** und **Alpha** für zwei unterschiedliche Dateien verwendet werden. Weitere Beispiele für gültige Dateinamen sind

Alpha127	**Abflug_Freitag**	**Kapitel.127.3**
qwx...5%..mxy	**mw._.j56+:**	**Brief.tex**
end:_.++=6	**INHALT_KAPITEL**	**+**

Nicht empfohlen werden Dateinamen, welche die oben genannten Sonderzeichen der SHELL enthalten:

>5lines	**$inhalt**	**?buch**
alpha|beta	**reihenfolge[23]**	**all***

Es ist offensichtlich, daß UNIX bei der Wahl des Dateinamens dem Benutzer freie Hand läßt. Der Name der Datei muß folglich mit ihrem Inhalt nicht direkt etwas zu tun haben. Es ist dennoch sinnvoll Namen zu verwenden, die auf den Dateiinhalt schließen lassen. Üblicherweise eingehaltene Namenskonventionen sind in der nachfolgenden Tabelle 4.1 angeführt.

Anhang	Inhalt der Datei
.c	Quelltext in C
.h	einzufügende Datei in C (include file)
.f	Quelltext in Fortran
.p	Quelltext in Pascal
.s	Quelltext in Assembler
.o	übersetzte Datei (object code)

Tabelle 4.1: Einige Namenskonventionen für Dateien

4.2 Verzeichnisse

Zum Anlegen eines Verzeichnisses dient der Befehl **mkdir** (**make directory**), zum Wechsel in ein neues Arbeitsverzeichnis **cd**.

```
$ mkdir  verz
$ cd  verz
$
```

Übung 8 *Legen Sie ein Verzeichnis* **verz** *und darin auf folgende Art eine Datei* **alpha** *an:*

```
$ cat  >alpha
alpha ist die erste Datei<return>
<ctrl d>
$
```

Wiederholen Sie diese Vorgehensweise zum Erzeugen weiterer Dateien **beta** *und* **gamma**. *Erzeugen Sie in* **verz** *außerdem ein Unterverzeichnis mit dem Namen* **adr**.

Der Inhalt des aktuellen Verzeichnisses wird durch

```
$ ls

adr    alpha  beta   gamma

$
```

angezeigt. Als Parameter des Kommandos **ls** kann auch der Name eines Verzeichnisses vorkommen, dessen Inhalt ausgegeben werden soll.

```
$ ls   adr
$
```

Bei den Kommandos kann die Angabe eines Datei- oder Verzeichnisnamens absolut oder relativ erfolgen.

Die **absolute** Angabe beginnt immer beim Wurzelverzeichnis, die einzelnen Verzeichnisse werden durch / voneinander getrennt, z.B. sind

/home/leibner/verz / /home/bfs/praktikum/aufgaben

absolute Namen, wogegen

verz praktikum/aufgaben

relative Namen sind. Relative Namen beziehen sich immer auf das aktuelle Arbeitsverzeichnis. Zu ihrer Angabe können auch `.` und `..` verwendet werden:

`..` `../../aufgaben` `../../..`

Beispiel (→ Abb.4.4):

```
$ ls   ../unix/beta
$ pwd

/home/leibner/verz

$ cd   ../unix/beta
$ pwd

/home/leibner/unix/beta

$ cd   ../../../../..   # ein .. überflüssig
$ pwd

/

$
```

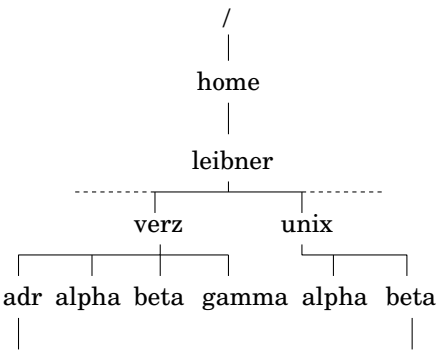


Abbildung 4.4

`#` Die mit einem Kommentar versehene Zeile (der Text hinter `#` wird von der SHELL als Kommentar interpretiert) zeigt, daß das zum Wurzelverzeichnis übergeordnete Verzeichnis wiederum das Wurzelverzeichnis ist (→ Abb.4.3).

Wie bereits erwähnt, entspricht `.` dem Namen des Arbeitsverzeichnisses. Der Punkt kann daher als passende Abkürzung in denjenigen Befehlen verwendet werden, in denen der Name des Arbeitsverzeichnisses als Parameter benötigt wird.

```
$ cp   /home/leibner/verz/alpha   .   # copy   von_datei   nach_datei
```

```
cp: ./alpha: Permission denied

$
```

Hat man im aktuellen Verzeichnis keine Schreibrechte, so wird keine Kopie angelegt.

Außer in der Bourne-SHELL kann in allen anderen für das Heimatverzeichnis ~ verwendet werden. Ein Wechsel nach **verz** im Heimatverzeichnis **/home/leibner** erfolgt dann durch

```
$ cd  ~/verz
$
```

So wie durch **mkdir** ein Verzeichnis angelegt werden kann, so kann es durch **rmdir** (**remove directory**) wieder gelöscht werden.

```
$ mkdir  vzs
$ ls

adr    alpha  beta   gamma  vzs

$ rmdir  vzs
$ ls

adr    alpha  beta   gamma

$
```

Ist das Verzeichnis nicht leer, so wird es durch **rmdir** nicht gelöscht. Sind in einem Verzeichnis weitere Unterverzeichnisse und Dateien vorhanden, so kann der gesamte Dateibaum (**rekursiv**) durch **rm -r** gelöscht werden.

4.2.1 Standardverzeichnisse

Die grundlegende Struktur der Verzeichnisse (→ Abb.4.5), die direkt unter dem Wurzelverzeichnis angeordnet sind, ist untrennbar mit dem Betriebssystem verbunden. Da viele Programme auf diese Verzeichnisse zugreifen, kann es kein Hersteller riskieren, ihre Namen und Anordnung zu verändern. Inkompatibilität zu anderen UNIX-Systemen wäre die unmittelbare Folge solchen Tuns.

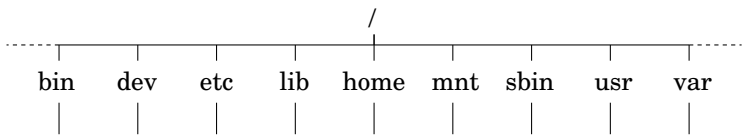


Abbildung 4.5: Standardverzeichnisse unter dem Wurzelverzeichnis

Wie bereits erwähnt, unterscheidet die SHELL zwischen inneren und äußeren Befehlen. Die äußeren Befehle sind Programme, die unter **/bin** abgelegt sind, z.B. **calendar, chmod, chown, cp, find, grep, mailx, more, sh, test** usw. Ihre Namen entsprechen denjenigen der Befehle.

Die Rechnerperipherie ist dem Benutzer nur mittels Systemkern zugänglich. Teil des Systemkerns ist eine Bibliothek von Ein- und Ausgabemodulen, in denen für jedes periphere Gerät eine Menge von Funktionen zur Verfügung steht, die dieses Gerät bedienen können (driver). Der Zugriff auf diese Funktionen und damit auf die Peripherie ist nur über spezielle Dateien (device driver) möglich. Diese Dateien sind unter **/dev** abgelegt.

Mit diesen speziellen Dateien arbeitet in der Regel nur der Systemverwalter oder besondere „daemons“[1] (Prozesse, die im Hintergrund laufen und Funktionen zur Unterstützung des Systemlaufs anbieten, z.B. der Druckerdaemon oder Maildaemon). Für alle zugänglich ist ein „Abfallkorb“, in den nichterwünschte Bildschirmausgaben umgelenkt und mit dessen Hilfe leere Dateien erzeugt werden können.

```
$ xhost +zeta >/dev/null # Umlenkung der Ausgabe von xhost\
in den Abfallkorb
$ cat /dev/null >leer
$ cp /dev/null auchleer
$ ls -l

total 4
drwxrwxr-x   2 leibner  system       512 Aug 30 15:04 adr
-rw-rw-r--   1 leibner  system         6 Aug 30 14:01 alpha
-rw-rw-r--   1 leibner  system         0 Sep  1 11:41 auchleer
-rw-rw-r--   1 leibner  system         7 Aug 30 14:01 beta
-rw-rw-r--   1 leibner  system         6 Aug 30 14:01 gamma
-rw-rw-r--   1 leibner  system         0 Sep  1 11:41 leer

$
```

Für den fehlerfreien Lauf des Betriebssystem ist der Inhalt von **/etc** von Bedeutung, da in diesem Verzeichnis die Dateien zur Benutzerverwaltung (**passwd** und **group**) sowie Dateien zur Systemkonfiguration enthalten sind.

Eine Bibliothek von Funktionen für unterschiedliche Systemanforderungen (z.B. dem Systemstart) ist unter **/lib** untergebracht.

Das Verzeichnis **/mnt** ist ein leeres Hilfsverzeichnis, unter das temporär ein weiteres Dateisystem angebunden werden kann.

[1] Aus dem Keltischen für *vergöttertes Wesen*, zum Unterschied zu „demon“, was *böser Geist* bedeutet.

Unter **/home** fängt der Bereich der Benutzer an.

/tmp ist ein offen zugängliches Verzeichnis zur vorübergehenden Aufbewahrung von Dateien. Es wird vor allem von Anwendungen, wie z.B. dem C-Compiler, zur Ablage von Zwischenergebnissen verwendet. Da **/tmp** regelmäßig geleert wird, ist es zur Archivierung von Dateien nicht geeignet.

Ursprünglich lag der Benutzerbereich unter **/usr**. Mit der Zunahme der Anwendungen begann man dieses Verzeichnis für diese neuen Komponenten des Betriebssystems zu nutzen, der Benutzerbereich wurde gleichzeitig abgetrennt und unter **/home** angesiedelt.

Unter **/usr/man** liegt z.B. das on-line-manual, unter **/usr/ccs** der C-Compiler, unter **/usr/include** Dateien, die in C mittels **#include** eingefügt werden, wie z.B.

#include <stdio.h>

für die Standardeingabe und -ausgabe. Ferner enthält **/usr/lib** Programme, welche die Arbeit im Betriebssystem und mit dem C-Compiler (Preprozessor **cpp**) unterstützen.

Während die Verzeichnisse und Dateien, die unter **/usr** abgelegt sind, von fester Größe sind, können diejenigen unter **/var** in ihrer Größe variieren.

Dies betrifft z.B. **/var/spool/mail**, wo ankommende „e-mails“ zwischengespeichert werden oder **/var/spool/cron**, wo Verzeichnisse über Aufgaben des Programms **cron** verwaltet werden.

4.3 Verwendung von Sonderzeichen bei der Namensangabe

Zur Angabe von Dateinamen können Sonderzeichen der SHELL verwendet werden. Es sind dies die Zeichen `?`, `*`, `[`, `!`, `-` und `]`.

Angenommen, am Rechner werde gerade ein Skriptum erstellt.

Im Arbeitsverzeichnis **/home/leibner/skript** befinden sich dann z.B. die Dateien der Kapitel (Anfangsbuchstabe **k**), Tabellen (Anfangsbuchstabe **t**), Bilder (Anfangsbuchstabe **b**), des Anhangs (Anfangsbuchstabe **a**) und weitere (z.B. **inhalt**).

$ ls

```
a    aC     b2-2   b2-5       inhalt   k22   k4          t2-1
aA   b1-1   b2-3   b4-1       k1       k23   literatur   t2-2
aB   b2-1   b2-4   einleitung k21      k3    t1-1        vorwort
```

```
$
```

Das Sonderzeichen `?` repräsentiert **genau ein** beliebiges Zeichen. Möchte man z.B. die Namen aller Dateien wissen, die mit einem **k** beginnen und von genau einem weiteren Zeichen gefolgt werden, so ist

?

```
$ ls  k?

k1  k3  k4

$
```

einzugeben. Die Namen der Verzeichnisse, die nur aus einem Zeichen bestehen, erhält man durch

```
$ ls  ?

a
```

* Das Sonderzeichen `*` repräsentiert eine **beliebig lange** Folge (auch der Länge Null) von Zeichen.

Der Stern am Beginn eines Namens vertritt keine Dateinamen, die mit **.** beginnen!

Die Namen aller Dateien, die mit **a** oder **b** beginnen, stellt man durch

```
$ ls  a*  b*

a  aA  aB  aC  b1-1  b2-1  b2-2  b2-3  b2-4  b2-5  b4-1

$
```

fest. Zur Angabe eines Dateinamens können mehrere Sonderzeichen gleichzeitig verwendet werden:

```
$ ls  *2*

b2-1  b2-2  b2-3  b2-4  b2-5  k21  k22  k23  t2-1  t2-2

$ ls  ?1*

b1-1  k1  t1-1
```

Bevor die SHELL die Namen dem Kommando übergibt, expandiert sie die verwendeten Sonderzeichen. Dies kann bei unvorsichtiger Verwendung dieser Zeichen in Befehlen, die Dateien löschen, unangenehme Folgen haben. Sollen Sonderzeichen in Dateinamen bei solchen Befehlen verwendet werden, so kann ihre Wirkung zuvor mit Hilfe des Kommandos **echo**, das lediglich seine Argumente (expandiert) darstellt, überprüft werden:

```
$ echo  ?1*

b1-1  k1  t1-1

$
```

Das Paar von Sonderzeichen `[` und `]` erlaubt die Angabe von **genau einem** Zeichen aus der von ihnen eingeschlossenen Zeichenmenge. So erhält man z.B. alle Dateien, die mit dem Buchstaben **t** oder **l** anfangen, durch `[]`

```
$ ls  [tl]*

literatur  t1-1  t2-1  t2-2

$
```

Das Sonderzeichen `-` erlaubt die Angabe eines Intervalls von ASCII-Zeichen. `[-]`

```
$ ls  *[2-4]*

b2-1  b2-2  b2-3  b2-4  b2-5  b4-1  k21  k22  k23  k3  k4  t2-1  t2-2

$
```

Durch `!` wird die Menge der eingeschlossenen Zeichen ausgeschlossen. Mit `[!]`

```
$ ls  [!tabk]*

einleitung  inhalt  literatur  vorwort
```

werden alle Dateien erfaßt, deren Namen **nicht** mit **t**, **a**, **b** oder **k** anfangen.

Selbstverständlich können die angesprochenen Sonderzeichen auch kombiniert

```
$ ls  *[1-3]??

b1-1  b2-1  b2-2  b2-3  b2-4  b2-5  t1-1  t2-1  t2-2
```

und in relativen oder absoluten Pfadnamen, wie z.B.

/home/leibner/skript/b* ../*e*[!r-w]

verwendet werden. Durch

```
$ set  -f
$ ls  *

* not found

$
```

kann die Expansion von Sonderzeichen durch die SHELL aufgehoben und durch

```
$ set  +f
$
```

wieder eingestellt werden.

4.4 Kopieren, Löschen und Umbenennen von Dateien

4.4.1 Kopieren von Dateien

Zum Kopieren von Dateien dient der Befehl **cp** (**copy**). Eine Kopie der **einleitung** mit dem Namen **einleitung.kopie** erhält man z.B. durch

```
$ cp  einleitung  einleitung.kopie
$ ls  ein*

einleitung  einleitung.kopie

$
```

Für das Kommando **cp** gibt es drei unterschiedliche Anwendungsmöglichkeiten. Die erste, das Kopieren einer Datei, wurde soeben gezeigt.

Die Kopie eines Verzeichnisses samt seiner Unterverzeichnisse erhält man durch

```
$ cp  -r  altes_verz  neues_verz    # der Schalter -r steht für recursive
$
```

Das Verzeichnis, welches beim Kopieren anlegt und in welches die Dateien kopiert werden, darf vorher nicht existiert haben!

In ein vorhandenes Verzeichnis können gleichzeitig mehrere Dateien kopiert werden.

```
$ cp  dat1  dat2  dat3  ..
$
```

Die kopierten Dateien erhalten dabei die Namen der Quelldateien.

Ist eine der Quelldateien ein Verzeichnis, so funktioniert das Kopieren so nicht:

```
$ cp  dat1  dat2  dat3  verz  ..

cp: verz is a directory (not copied).
```

Mit dem Schalter **-r** kann man dieses Problem überwinden:

```
$ cp  -r  dat1  dat2  dat3  verz  ..
$
```

Die Dateinamen können auch mit Hilfe von Sonderzeichen angegeben werden. Möchte man z.B. alle Kapitel des Skriptums aus dem vorherigen Abschnitt in ein eigenes Verzeichnis **nur_kapitel** kopieren, so kann dies durch

```
$ mkdir  ../nur_kapitel
$ cp  k*  ../nur_kapitel
$ ls  ../nur_kapitel

k1  k21  k22  k23  k3  k4

$
```

geschehen.

Beim Kopieren auf eine bereits vorhandene Datei wird deren ursprünglicher Inhalt überschrieben. Mit dem Schalter **-i** fragt **cp** vor dem Kopieren nochmals nach, ob man die existierende Datei wirklich überschreiben will:

```
$ cp  -i  a  aA

overwrite aA? n
```

```
$
```

Ist die Antwort **y**, wird kopiert, ist sie **n**, nicht.

Beim Kopieren kann für den Namen des Arbeitsverzeichnisses (gleicher i-node) auch **.** verwendet werden.

```
$ cp   ../../unix/alpha   .
$
```

4.4.2 Löschen von Dateien

Mit dem Kommando **rm** (**remove**) werden Dateien gelöscht.

```
$ ls

dat1  dat2  dat3

$ rm   dat2
$ ls

dat1  dat3
```

Wird beim Löschen für die Namen `*` verwendet, so wird außer den Dateien, die mit einem Punkt beginnen und eventuellen Unterverzeichnissen, alles gelöscht.

```
$ ls  -F

dat1  dat3  verz/

$ rm  *

rm: verz directory

$ ls  -a

.  ..  verz

$
```

Der Schalter **-F** kennzeichnet Verzeichnisse durch ein angehängtes /.

Verzeichnisse samt Unterverzeichnissen werden mit Hilfe des Schalters **-r** gelöscht:

```
$ rm  -r  verz    # der Schalter -r steht für recursive
$ ls  -a
```

```
.  ..

$
```

Wie beim Kommando **cp** gibt es auch hier die Möglichkeit, vor der Ausführung des Befehls, mit dem Schalter **-i** eine Abfrage zu starten:

```
$ ls

dat1  dat2

$ rm  -i  *

rm: remove dat1? n
rm: remove dat2? y

$ ls

dat1

$
```

Ist eine Änderung des Dateiinhalts aufgrund der Zugriffsrechte nicht erlaubt, so fragt **rm** nochmals nach, ob tatsächlich gelöscht werden soll.

```
$ ls  -l

-r--r--r--  1  leibner  system  5 Sep 20 09:02  dat1

$ rm  dat1

rm: override protection 444 for dat1? n

$
```

Der Schalter **-f** (**force**) erzwingt das Löschen einer Datei ohne Rücksicht auf die Zugriffsrechte und ohne nochmaliges Abfragen.

```
$ rm  -f  dat1
$ ls  dat1

dat1 not found

$
```

4.4.3 Umbenennen und Verschieben von Dateien

Zum Umbenennen und Verschieben von Dateien dient das Kommando **mv** (**move**).

Angenommen, im Arbeitsverzeichnis sind drei Verzeichnisse und drei Dateien:

```
$ ls -F

dat1  d2  dat3  verz1/  verz2/  verz3/
```

Für das Kommando **mv** gibt es, wie bei **cp**, drei Anwendungsmöglichkeiten. In allen drei Fällen werden dabei lediglich die Namen in den Verzeichnissen (→ Abb.4.2) entweder geändert oder in anderen Verzeichnissen neu (evtl. verändert) angelegt. Die i-node-Nummern und damit auch die Lage der gespeicherten Daten, bleiben dabei unverändert.

Die erste Anwendung ist das Umbenennen einer Datei:

```
$ mv  d2  dat2
$ ls

dat1  dat2  dat3  verz1/  verz2/  verz3/

$
```

Die Verschiebung und eventuelle Änderung eines Datei- oder Verzeichnisnamens erfolgt durch

```
$ mv  verz2  ../vvv
$ ls

dat1  dat2  dat3  verz1  verz3

$
```

Eine Verschiebung ist nur innerhalb eines Dateisystems möglich (→ Abb.4.1)! Sie kann außerdem nur in Verzeichnisse erfolgen, in denen man entsprechende Schreibrechte hat.

```
$ mv  dat1  /home

mv: rename dat1 to /home/dat1: Permission denied
```

```
$
```

Wie bei **cp** und **rm** gibt es auch bei **mv** die Möglichkeit, mittels **-i** die Ausführung des Kommandos von der Beantwortung einer Frage abhängig zu machen:

```
$ mv  -i  dat1  ../.

overwrite .././dat1? n

$
```

In ein vorhandenes Verzeichnis können gleichzeitig auch mehrere Datei- und Verzeichnisnamen verschoben werden.

```
$ ls

dat1  dat2  dat3  verz1  verz2  verz3

$ mv  *[12]  verz3
$ ls

dat3  verz3

$ ls  verz3

dat1  dat2  verz1  verz2

$
```

Da die Umbenennung mehrerer Dateien mit **mv** nicht möglich ist, muß vor der Verwendung von Sonderzeichen gewarnt werden.

4.5 Zugriffsrechte

Jede Datei hat die auf Seite 36 angegebenen Attribute. Die neun Zugriffsrechte (**access rights**, in den Spalten 2-10) legen die mit der Datei erlaubten Operationen fest. Sie werden getrennt für den Eigentümer **u** (**user**, 2-4-te Spalte), die Gruppe **g** (**group**, 5-7-te Spalte) und alle anderen **o** (**others**, 8-10-te Spalte), angegeben.

Der Eigentümer einer Datei ist derjenige, der die Datei angelegt hat.

```
$ ls  -l

drw-------   2 leibner  system       512 Oct  4 09:16 briefe
-rw-rw-r--   1 leibner  system         9 Oct  4 09:16 geburtstag
drwxr-xr-x   2 leibner  system       512 Oct  4 09:16 skript
```

Bei den Zugriffsrechten wird zwischen lesen **r** (**read**), schreiben **w** (**write**) und ausführen **x** (**execute**), unterschieden. Der Bindestrich **-** schließt die jeweilige Operation aus, **r**, **w** und **x** erlauben sie.

Unter UNIX werden ausführbare Dateien nicht speziell gekennzeichnet, die Ausführbarkeit wird allein durch das Setzen von **x** bestimmt. Ausführbar kann eine binäre Datei oder ein SHELL-Script sein.

Obwohl in UNIX zwischen Dateien und Verzeichnissen kein grundsätzlicher Unterschied besteht, kann man sich dennoch kaum ein „ausführbares Verzeichnis“ vorstellen. Bei Verzeichnissen gibt daher **x** nicht die Ausführbarkeit sondern die Durchgängigkeit des Dateibaums bei einem Verzeichniswechsel an. Ist **x** eingestellt, so kann das Verzeichnis zum Arbeitsverzeichnis gemacht werden (→ Tab.4.2).

Zgrt.	**Datei**	**Verzeichnis**
r	Dateiinhalt lesen (**more**, **cat**)	Verzeichnisinhalt lesen (**ls**)
w	Dateiinhalt modifizieren (z.B. **vi**)	Anlegen und Löschen von Dateien
x	Datei ausführen	Verzeichnis zum Arbeitsverzeichnis machen (**cd**)

Tabelle 4.2: Bedeutung der Zugriffsrechte

Hat ein Verzeichnis zusätzlich zu **x** (arbeiten im Verzeichnis) **w** eingestellt, so kann es beschrieben werden. Dies bedeutet, daß darin Dateien angelegt (in den i-node eingetragen) oder gelöscht (aus dem i-node entfernt) werden können.

Ist neben **x** das Leserecht **r** gesetzt, so kann der Verzeichnisinhalt ausgegeben und vorhandene Dateien, falls sie für den Benutzer **w** gesetzt haben, verändert werden. Neue Dateien können im Verzeichnis nicht angelegt werden.

Da die Kommunikation mit der Rechnerperipherie ebenfalls über (spezielle) Dateien erfolgt, kann durch die Änderung der Zugriffsrechte auf diese Dateien die Menge der Benutzer, die bestimmte periphere Geräte bedienen darf, bestimmt werden.

4.5.1 Änderung der Zugriffsrechte

Zum Ändern der Zugriffsrechte dient das Kommando **chmod**. Die Änderung kann numerisch oder symbolisch erfolgen.

Die numerische Einstellung der Zugriffsrechte erfolgt durch die Angabe des oktalen Wertes der binären Dreiergruppen für **u**, **g** und **o**:

```
$ chmod 755 geburtstag
$ ls -l geburtstag
```

```
-rwxr-xr-x   1 leibner  system         7 Oct  4 11:09 geburtstag
```

Eine 1 entspricht dem Setzen, eine 0 dem Entfernen des Zugriffsrechts.

r w x	r - x	r - x
1 1 1	1 0 1	1 0 1
7	5	5

Die mnemonische Eingabe der Zugriffsrechte erfolgt durch die Zuweisung der Rechte dem Eigentümer, der Gruppe, den anderen oder allen (**a**, **all**).

```
$ ls  -l  geburtstag
----------   1 leibner  system         7 Oct  4 11:09 geburtstag
$ chmod  a=rx,ug+w  geburtstag
$ ls  -l  geburtstag
-rwxrwxr-x   1 leibner  system         7 Oct  4 11:09 geburtstag
$ chmod  g-w  geburtstag
$ ls  -l  geburtstag
-rwxr-xr-x   1 leibner  system         7 Oct  4 11:09 geburtstag
$ chmod  go=  geburtstag
$ ls  -l  geburtstag
-rwx------   1 leibner  system         7 Oct  4 11:09 geburtstag
$
```

Durch = werden Rechte zugewiesen, durch + hinzugefügt, durch **-** weggenommen und durch = allein, der angegebenen Kategorie von Benutzern alle Rechte entfernt.

Übung 9 *Fügen Sie der Datei* **dat1** *die Rechte zur Ausführung für* **g** *und* **o** *hinzu.*

Übung 10 *Mit dem Schalter* **-R** *können Zugriffsrechte rekursiv vergeben werden. Ändern Sie die Rechte für* **verz** *und alle darunterliegenden Dateien in* `rwxr--r--`*.*

Übung 11 *Nehmen Sie den Dateien* **dat1**, **dat2** *und* **dat3** *für* **g** *und* **o** *das Leserecht und fügen Sie das Recht zur Ausführung für alle hinzu.*

4.5.2 Set-uid-Bit, set-gid-Bit und sticky-Bit

Die Benutzer werden in UNIX durch ihre Identifikationsnummern (**user-identifier, uid**) unterschieden. Diese Nummern kennzeichnen z.B. im i-node den Eigentümer einer Datei.

Der „login"-Name wird nur zur Anmeldung am System verwendet. Für ein und dieselbe uid können somit unterschiedliche Benutzernamen und Paßwörter existieren.

Die Identifikationsnummern der Benutzer werden zusammen mit weiteren wichtigen Informationen in der Datei **/etc/passwd** gespeichert. Für jeden Benutzer enthält diese Datei eine Zeile:

```
$ grep leibner /etc/passwd

leibner:GIRYnioXpy3AQ:200:25:Dr.Leibner:/home/leibner:/bin/ksh
```

Die einzelnen Einträge sind durch einen Doppelpunkt getrennt. Ihre Bedeutung ist folgende:

Login-name:Paßwort:uid:gid:Kommentar:Heimatverzeichnis:SHELL

Das Paßwort ist in chiffrierter Form gespeichert. In neueren UNIX-Versionen wird das Paßwort in der Regel in der Datei **/etc/shadow** abgelegt. Neben der Identifikationsnummer des Benutzers ist diejenige der Gruppe aufgeführt. Danach folgt im Kommentarfeld normalerweise der vollständige Name des Benutzers. In einem weiteren Feld steht der absolute Pfad zum Heimatverzeichnis und im letzten der Aufrufpfad zum SHELL-Interpreter. Zur Auswahl stehen dabei in der Regel mehrere unterschiedliche Interpreter, z.B. **ksh**, **csh**, **tcsh**, **bash**, usw.

Der normale Benutzer kann sein Paßwort mit Hilfe des Programms **/bin/passwd** jederzeit ändern, obwohl er für die Datei **/etc/passwd** nur das Leserecht hat:

```
$ ls -l /etc/passwd

-rw-r--r--   1 root     system       761 Dec 20  1994 /etc/passwd

$
```

Die Bearbeitung von **/etc/passwd** durch das Programm **/bin/passwd** ermöglicht das **set-uid**-Bit.

```
$ ls -l /bin/passwd

-rws--x--x   3 root     bin        16384 Aug  9  1994 /bin/passwd
```

```
$
```

Ein Programm, das für den Besitzer statt **x** das set-uid-Bit **s** gesetzt hat, erhält die uid und damit auch die Zugriffsrechte seines Eigentümers, unabhängig davon, von wem es gestartet wird. Da das laufende Programm **/bin/passwd** bei der Arbeit mit der Datei **/etc/passwd** damit die Rechte **rw** von root erhält, kann jeder Benutzer mit **/bin/passwd** den Eintrag in der Datei **/etc/passwd** ändern.

Das set-uid-Bit wird entweder numerisch durch oktal **4000**, oder mnemonisch durch das Hinzufügen mit +, gesetzt.

```
$ ls -l prog1

-rwxr-xr-x   1 leibner  system         6 Oct  4 14:34 prog1

$ chmod 4755 prog1
$ ls -l prog1

-rwsr-xr-x   1 leibner  system         6 Oct  4 14:34 prog1

$ ls -l prog2

-rwxr-xr-x   1 leibner  system         6 Oct  4 14:35 prog2

$ chmod u+s prog2
$ ls -l prog2

-rwsr-xr-x   1 leibner  system         6 Oct  4 14:35 prog2

$
```

Auf die gleiche Art und Weise wird das **set-gid**-Bit gesetzt. Dieses Bit gibt dem zugreifenden Programm die Rechte der Gruppe, in die der Eigentümer der Datei gehört.

Das set-gid-Bit wird oktal durch **2000** gesetzt.

Wird ein Programm häufig benötigt, so kann das wiederholte Ein- und Auslesen von der Festplatte durch das Setzen des **sticky**-Bits vermieden werden. Ist das sticky-Bit gesetzt, so wird das geladene Programm nach seiner Beendigung nicht auf die Festplatte zurückgeschrieben, sondern im Ausgleichsspeicher zwischengespeichert. Dadurch wird der wiederholte Zugriff vereinfacht und beschleunigt.

```
$ ls -l /bin/vi

-rwxr-xr-t   5 bin      bin       294912 Aug  9  1994 /bin/vi
```

```
$
```

Das sticky-Bit **t** kann in der Regel nur der Systemverwalter, numerisch durch die oktale Angabe von **1000**, setzen.

```
$ ls  -l  prog

-rwxr-xr-x   1 leibner  system         6 Oct  4 14:34 prog

$ chmod  1755  prog
$ ls  -l  prog

-rwxr-xr-t   1 leibner  system         6 Oct  4 14:34 prog

$
```

4.5.3 Änderung des Eigentümers und der Gruppe

Die Benutzer des Betriebssystems können in Gruppen eingeteilt werden. Dies erlaubt die Vergabe gleicher Gruppenrechte an Dateien, die von unterschiedlichen Benutzern einer Gruppe bearbeitet werden sollen.

Die Namen aller Gruppen im System erhält man durch

```
$ groups

user1 user2 user3
```

Mit Hilfe des Kommandos **groups** kann man auch seine eigene Gruppenzugehörigkeit feststellen:

```
$ groups  leibner

user2

$
```

Der Eintrag in eine neue Gruppe erfolgt durch

```
$ newgrp  user3
$
```

Falls der Benutzer mehreren Gruppen angehört (die Gruppen stehen in **/etc/groups**), erhalten neu angelegte Dateien immer die Gruppenzugehörigkeit, die beim letzten

newgrp-Kommando angegeben wurde.

Durch

```
$ newgrp -
$
```

wird die Gruppe eingestellt, die beim Einrichten der Benutzerkennung festgelegt wurde.

Mit **chgrp** (**change group**) kann die Gruppenzugehörigkeit einer Datei geändert werden:

```
$ chgrp user1 dat1
$ ls -l dat1
-rwxr-xr-x   1 leibner  user1          6 Oct  4 14:34 dat1
$
```

Falls die Gruppe nicht existiert, meldet das System einen Fehler:

```
$ chgrp xx dat1
chgrp: unknown group: xx
$
```

Mit dem Kommando **chown** kann der Eigentümer einer Datei geändert werden. Dies ist in der Regel nur dem Systemverwalter erlaubt.

```
$ chown otto dat1
chown: dat1: Not owner
$
```

Die Kommandos **chgrp** und **chown** verfügen über den Schalter **-R** (**recursive**), der eine rekursive Änderung der Gruppe und des Eigentümers in allen Dateien und Unterverzeichnissen erlaubt.

4.5.4 Voreingestellte Rechte

Die Zugriffsrechte einer neu angelegten Datei (Verzeichnis) werden nach dem Wert der „file-creation mode mask“ eingestellt. Der Wert dieser Maske kann mit **umask** abgefragt und geändert werden.

```
$ umask

02

$ cat  /dev/null  >dat
$ ls  -l  dat

-rw-rw-r--   1 leibner  system         0 Oct  5 13:56 dat

$
```

Der Wert `02` gibt nicht die eigentlichen Zugriffsrechte an. Die Zugriffsrechte erhält man durch das Subtrahieren des umask-Wertes vom oktalen Wert 666 bei gewöhnlichen Datein, bzw. 777 bei Verzeichnissen.

```
$ umask  22
$ cat  /dev/null  >dat
$ mkdir  verz
$ ls  -l

-rw-r--r--   1 leibner  system         0 Oct  5 14:00 dat
drwxr-xr-x   2 leibner  system       512 Oct  5 14:01 verz

$
```

	Datei	Verzeichnis
	1 1 0 1 1 0 1 1 0	1 1 1 1 1 1 1 1 1
−	0 0 0 0 1 0 0 1 0	− 0 0 0 0 1 0 0 1 0
	1 1 0 1 0 0 1 0 0	1 1 1 1 0 1 1 0 1

4.6 Bildschirmausgabe von Dateien

Die Bildschirmausgabe einer Datei ist nur dann sinnvoll (und ungefährlich), wenn es sich bei der Datei um eine Textdatei handelt.

Die Art der Datei kann anhand einiger Schlüsselwörter mit dem Kommando **file** bestimmt werden.

```
$ file  /  /etc/r*
```

```
/:                      directory
/etc/rc.config:         /bin/sh shell script -- commands text
/etc/remote:            ascii text
/etc/resolv.conf.:      ascii text
/etc/rmt:               symbolic link to ../usr/sbin/rmt
/etc/routes:            ascii text
/etc/rpc:               commands text
```

Die Inhalte von Dateien, deren Klassifikation das Wort `text` enthält, können mit den nachfolgenden Befehlen auf dem Bildschirm dargestellt werden.

Ein Kommando, das nicht nur zur Ausgabe eines Dateiinhalts auf dem Bildschirm, sondern auch zur Verkettung (con**cat**enation) mehrerer Dateien dient, ist **cat**.

Die einfachste Anwendung von **cat**, ist die Bildschirmausgabe des gerade an der Tastatur eingegebenen Textes:

```
$ cat
erste Zeile
erste Zeile
zweite Zeile
zweite Zeile
<ctrl d>
$
```

Der eingegebene Text erscheint zweimal. Die erste Ausgabe führt der Tastaturtreiber durch. Er liest die Zeichen, die auf der Tastatur eingegeben werden und stellt sie sofort auf dem Bildschirm dar. Die eigentliche Ausgabe von **cat** steht in den geradzahlingen Zeilen.

Das Dateiende wird durch **<ctrl d>** bestimmt. Mit diesem Zeichen wird dem Kommando das Ende der Eingabe (**end of file**, **EOF**) mitgeteilt, **<ctrl d>** ist nicht Teil der Datei.

Bei einer gewöhnlichen Datei erhält **cat** das Zeichen **<ctrl d>** durch das Einlesen des letzten Zeichens der Datei (EOF).

Da die SHELL ebenso wie **cat** ein Programm ist, kann durch **<ctrl d>** der SHELL das Ende der Eingabe mitgeteilt und damit die Sitzung am Rechner beendet (logout) werden.

Durch die Umlenkung der Bildschirmausgabe in eine Datei, kann mit **cat** eine neue Datei erzeugt werden:

```
$ cat  >dat
Diese Datei hat nur eine Zeile.
<ctrl d>
$
```

Werden **cat** beim Aufruf ein oder mehrere Dateinamen übergeben, so wird der Inhalt dieser Datei bzw. Dateien auf dem Bildschirm ausgegeben:

```
$ cat  tel-1  tel-2

Ingrid Beck                 22223
Renate Krauss               26844
Angela Sczesny              25820
Gabriele Heiss              27372
Helga Lohmann               26113
Lothar Haschigk             44834
Irmgard Kalasch             44835
Peter Leibner               63436
Friedrich Leicher           47194
Karlheinz Reichenstetter    63325
Klaus Renno                 63315
Winfried Speidel            63502
Marie Winter                44835
Dagmar Zankl                63534
Elmar Ziegler               47608

$
```

Bei der Angabe der Dateinamen können auch Sonderzeichen verwendet werden, z.B. **cat t*[12]**.

Sind die Dateien sehr lang, so kann aufgrund der Schnelligkeit, mit der die Inhalte auf dem Bildschirm erscheinen, die Ausgabe kaum verfolgt werden. Die geeignete Alternative zu **cat** ist in diesen Fällen das Kommando **more**, da **more** die Ausgabe seitenweise durchführt.

```
$ more  tel-1  tel-2

::::::::::::::
tel-1
::::::::::::::
```

```
Ingrid Beck                 22223
Renate Krauss               26844
Angela Sczesny              25820
Gabriele Heiss              27372
Helga Lohmann               26113



tel-1: END (next file: tel-2)<return>



Lothar Haschigk             44834
Irmgard Kalasch             44835
Peter Leibner               63436
Friedrich Leicher           47194
Karlheinz Reichenstetter    63325
Klaus Renno                 63315
Winfried Speidel            63502
Marie Winter                44835
Dagmar Zankl                63534
Elmar Ziegler               47608



tel-2: END<return>

$
```

Eine neue Seite erhält man durch die Eingabe eines Leerzeichens, eine neue Zeile durch RETURN. Vorzeitig beenden kann man **more** durch die Eingabe von **q**.

Beim Betrachten des Dateiinhalts können eine ganze Reihe von Kommandos, z.B. zum Suchen von Textstellen oder zum Verschieben des Ausgabefensters, eingegeben werden (→ Tab.4.3). Einen Überblick über diese Kommandos erhält man durch die Eingabe von **h**.

Den Kommandos mit * kann eine Zahl vorangestellt werden.	
h	Zeige diese Tabelle an.
f, <ctrl f>, SPACE	* N Zeilen vorwärts (ein Bildschirm).
b, <ctrl b>	* N Zeilen rückwärts (ein Bildschirm).
j, RETURN	* N Zeilen vorwärts (eine Zeile).
k	* N Zeilen rückwärts (eine Zeile).
d, <ctrl d>	* N Zeilen vorwärts (halber Bildschirm).
u, <ctrl u>	* N Zeilen rückwärts (halber Bildschirm).
g	* Gehe zur N-ten Zeile, (erste Zeile).
G	* Gehe zur N-ten Zeile (letzte Zeile).
p, %	* Cursor zur Stelle N Prozent der Datei.
r, <ctrl l>	Bildschirm auffrischen.
R	Bildschirm auffrischen, gepufferte Eingabe verwerfen.
m[a-z]	Markiere aktuelle Stelle unter Buchstabe a-z.
'[a-z]	Gehe an zuvor markierte Stelle.
"	Gehe an vorhergehende Position.
/*zeiket*	* Suche vorwärts zur N-ten Zeile, die *zeiket* enthält.
!*zeiket*	* Suche vorwärts zur N-ten Zeile, die NICHT *zeiket* enthält.
?*zeiket*	* Suche rückwärts zur N-ten Zeile, die *zeiket* enthält.
?!*zeiket*	* Suche rückwärts zur N-ten Zeile, die NICHT *zeiket* enthält.
n	* Wiederhole Suche.
:a	Zeige die Liste der Dateien an.
E	Überprüfe eine neue Datei.
:n, N	* Überprüfe die nächste Datei.
:p, P	* Überprüfe die vorhergehende Datei.
:t	Überprüfe einen „Tag“.
v	Editiere die aktuelle Datei.
=, <ctrl g>	Gebe Informationen über die aktuelle Datei aus.
q, :q, or ZZ	Exit.

Tabelle 4.3: „On-line-help“ des Kommandos (Voreinstellung in Klammern) **more**

Ähnlich wie **more**, arbeitet **pg** (**pager**). Da auf vielen Systemen das Kommando **pg** jedoch nicht installiert ist, wird es hier auch nicht weiter behandelt.

Sollen nur die ersten Zeilen einer Datei auf dem Bildschirm ausgegeben werden, so kann dazu der Befehl **head** verwendet werden.

```
$ head  -2  tel-1

Ingrid Beck                22223
Renate Krauss              26844

$
```

Wird die Anzahl der Zeilen nicht explizit aufgeführt, so werden die ersten zehn Zeilen ausgegeben.

Die letzten Zeilen einer Datei können mit **tail** betrachtet werden. Als Schalter kann man eine positive oder negative Zahl angeben. Wird ein Plus vorangestellt ($+n$), so werden alle Zeilen, beginnend bei der n-ten, ausgegeben, steht vor der Zahl ein Minus ($-n$), so sind es die letzten n Zeilen. Implizit ist der Schalter auf -10 eingestellt.

```
$ tail  +4  tel-1

Gabriele Heiss              27372
Helga Lohmann               26113

$ tail  -2  tel-1

Gabriele Heiss              27372
Helga Lohmann               26113

$
```

Durch einen weiteren Schalter kann man angeben, ob mit n Zeilen (**l**, impliziter Wert), Blöcke (**b**) oder Zeichen (**c**) gemeint sind. Die letzten 95 Zeichen von **tel-1** erhält man so z.B. durch

```
$ tail  -95c  tel-1

Sczesny             25820
Gabriele Heiss              27372
Helga Lohmann               26113

$
```

Mit Hilfe der zu einer **Pipe** zusammengeschalteten Kommandos **head** und **tail**, kann ein beliebiger Zeilenbereich definiert werden:

```
$ tail  +3  tel-2 | head  -1

Peter Leibner               63436

$
```

Dateien, deren Klassifikation nicht das Wort `text` enthält, können mit **od** auf dem Bildschirm ausgegeben werden.

```
$ od  -cb  tel-1

0000000    I   n   g   r   i   d       B   e   c   k
         111 156 147 162 151 144 040 102 145 143 153 040 040 040 040 040
0000020                                                 2   2   2   2
         040 040 040 040 040 040 040 040 040 040 040 040 062 062 062 062
0000040    3  \n   R   e   n   a   t   e       K   r   a   u   s   s
         063 012 122 145 156 141 164 145 040 113 162 141 165 163 163 040
```

```
0000060                                                       2   6
        040 040 040 040 040 040 040 040 040 040 040 040 040 040 062 066
0000100   8   4   4  \n   A   n   g   e   l   a       S   c   z   e   s
        070 064 064 012 101 156 147 145 154 141 040 123 143 172 145 163
0000120   n   y
        156 171 040 040 040 040 040 040 040 040 040 040 040 040 040 040
0000140   2   5   8   2   0  \n   G   a   b   r   i   e   l   e       H
        062 065 070 062 060 012 107 141 142 162 151 145 154 145 040 110
0000160   e   i   s   s
        145 151 163 163 040 040 040 040 040 040 040 040 040 040 040 040
0000200           2   7   3   7   2  \n   H   e   l   g   a       L   o
        040 040 062 067 063 067 062 012 110 145 154 147 141 040 114 157
0000220   h   m   a   n   n
        150 155 141 156 156 040 040 040 040 040 040 040 040 040 040 040
0000240                   2   6   1   1   3  \n
        040 040 040 040 062 066 061 061 063 012
0000252

$
```

Ohne Angabe eines Schalters erfolgt die Ausgabe oktal.

Der Schalter **-c** bewirkt die Ausgabe als ASCII-Zeichen. Manche nichtdarstellbare Zeichen werden als darstellbare mit einem vorangestellten \ ausgegeben (→ Tab.4.4). Andere werden durch drei oktale Ziffern, entsprechend ihrem Wert in der ASCII-Tabelle, dargestellt.

Durch **-b** erfolgt die Ausgabe jedes Byte als oktale Zahl. Für die Leerzeichen zwischen den Namen und den Telefonnummern im obigen Beispiel steht daher $(040)_8$ (→ ASCII-Tabelle).

Symbol	**Zeichen**
\0	nul (leeres Zeichen)
\b	backspace (ein Zeichen zurück)
\f	form-feed (Seitenvorschub)
\n	newline (neue Zeile)
\r	return (an den Anfang der Zeile)
\t	tab (Tabulatorzeichen)

Tabelle 4.4: Umsetzung nichtdarstellbarer Zeichen laut ASCII-Tabelle

Pro Zeile gibt **od** 16 Zeichen aus. Die Summe aller Zeichen im obigen Beispiel ist somit $(0000252)_8 = (170)_{10}$.

Die Anzahl aller Zeichen in einer Datei erhält man einfacher durch **wc** (**word count**).

$ wc tel-1

```
         5        15       170 tel-1

$
```

Ausgegeben wird die Anzahl der Zeilen (5), Worte (15) und Zeichen (170). Am Ende der Zeile wird der Name der Datei angeführt. Werden mehrere Dateien **wc** übergeben, so berechnet **wc** zusätzlich die Summe (total) aller Zeilen, Worte und Zeichen.

```
$ wc  *

         5        15       170 tel-1
        10        30       340 tel-2
        15        45       510 total

$
```

Mit dem Schalter **-l** wird nur die Anzahl der Zeilen, mit **-w** die der Worte und mit **-c** die der Zeichen, ausgegeben.

```
$ wc  -lw  tel-2

        10        30 tel-2

$
```

Die Anzahl aller angemeldeter Benutzer erhält man z.B. mit Hilfe einer Pipe:

```
$ who | wc  -l

         3

$
```

4.7 Druckerausgabe von Dateien

Das Betriebssystem UNIX ist ein Mehrbenutzersystem, in dem oft mehrere Rechner zu einem Netz zusammengeschaltet sind.

Die Dateien, die ausgedruckt werden sollen, müssen in einem speziellen Verzeichnis zwischengespeichert werden, um dann auf einem oft nur einmal im Netz vorhandenen Drucker nacheinander ausgegeben werden zu können.

Der Vorgang des Zwischenspeicherns und anschließendem Druckens wird **spooling**[2] genannt. Die Ausgabe der in einer Warteschlange stehenden Druckaufträge besorgt

[2]Der Begriff wurde von IBM in den 60er Jahren geprägt, er steht für „simultaneous peripheral operations offline“

der Druckerdaemon (**line printer daemon**, **lpd**). Der Druckerdaemon wird bei der Initialisierung des Systems gestartet und läuft von da an als Hintergrundprozeß. Er schaut periodisch in einem speziellen Verzeichnis nach, ob Dateien zur Ausgabe anstehen und schickt diese gegebenenfalls nacheinander an den Drucker.

Leider gibt es zur Ausgabe, zur Überprüfung der Druckerschlange und zum Löschen von Druckaufträgen, bei den AT&T- und BSD-Systemen unterschiedliche Kommandos (→ Tab.4.5).

Bedeutung	**System V**	**Berkeley Unix**
Datei an Drucker schicken	**lpr**	**lp**
Druckaufträge anzeigen	**lpq**	**lpstat**
Druckauftrag löschen	**lprm**	**cancel**

Tabelle 4.5: Druckerkommandos

Die Kommandos **lpr** und **lp** (**line printer**) erlauben keine wesentliche Änderung der auszudruckenden Dateiinhalte.

Eine Formatierung einer Datei vor dem Druck ist mit Hilfe der Kommandos **pr** (**preview**) und **nl** (**number lines**) möglich. Nach Eingabe von

$ pr tel-1

erhält man für die ersten 10 Zeilen

```
Oct  9 11:20 1995 tel-1 Page 1


Ingrid Beck                22223
Renate Krauss              26844
Angela Sczesny             25820
Gabriele Heiss             27372
Helga Lohmann              26113
```

Ohne Angabe eines Schalters erzeugt **pr** 66 Zeilen lange Seiten, beginnend jeweils mit zwei Leerzeilen, gefolgt von einem Titel, zwei weiteren Leerzeilen, 56 Zeilen Text mit anschließenden weiteren 5 Leerzeilen. Die Zeilen enthalten maximal 72 Zeichen. Die Ausgabe ist dem amerikanischen Seitenformat angepaßt, das ungefähr 1cm länger ist als DIN A4.

Die Seitenlänge kann durch den Schalter **-l** (→ Tab.4.6) der europäischen Papiergröße angepaßt werden.

Schalter	Mnemonik	Bedeutung
-f	formfeed	zum Trennen der Seiten wird FORMFEED anstelle von NEWLINE verwendet
-m	multiple	druckt den Inhalt mehrerer Dateien, jede Datei in einer eigenen Spalte
-t	trailer	unterdrückt die ersten und die letzten 5 Zeilen (Titel); geeignet zum Eintrag der Ausgabe von **pr** in eine Datei
-h *zeiket*	header	im Titel wird anstelle des Dateinames *zeiket* eingetragen
-l*n*	length	legt die Seitenlänge auf n Zeilen fest
-s*c*	separator	bei mehrspaltiger Ausgabe erfolgt die Trennung der Spalten durch das Zeichen c
-w*n*	width	legt bei mehrspaltiger Ausgabe die Zeilenlänge fest
-*n*	–	Ausgabe einer Datei in n Spalten
+*n*	–	die Ausgabe beginnt bei Seite n

Tabelle 4.6: Schalter des Kommandos **pr**

Mit dem Kommando **nl** kann man die Zeilen einer Datei numerieren.

```
$ nl  tel-1

     1  Ingrid Beck              22223
     2  Renate Krauss            26844
     3  Angela Sczesny           25820
     4  Gabriele Heiss           27372
     5  Helga Lohmann            26113

$
```

Zur Druckerausgabe dient das Kommando **lpr**. Einige UNIX-Versionen bieten sowohl **lpr** als auch **lp** an, andere nur eins von beiden. Ausgeben kann man mehrere Dateien gleichzeitig:

```
$ lpr  dat1  dat2  dat3
$
```

Stehen unterschiedliche Drucker zur Verfügung, so kann durch **-P** der Drucker ausgewählt werden, an dem die Datei ausgegeben werden soll.

```
$ lpr  -Plaser2  dat
$
```

Gibt es nur **lp**, so lautet das entsprechende Kommando

```
$ lp  -d  laser2  dat
$
```

Sollen drei Kopien der Dateien **dat1** und **dat2** ausgegeben werden, so ist

```
$ lpr  -#3  dat1  dat2  # oder  lp  -n  3  dat1  dat2
$
```

einzugeben. Ohne die Angabe eines Druckernamens wird der in der Systemvariablen **PRINTER** (oder **LPDEST**) gespeicherte Name genommen. Ist die Systemvariable nicht gesetzt, so wird der in **/etc/printcap** (**printer capabilities**) unter **lp** stehende Name verwendet.

Sobald **lpr** den Namen des Druckers kennt, stellt er in **/etc/printcap** den Verzeichnisnamen für die Druckerschlange des ausgewählten Druckers fest. In diesem Verzeichnis erzeugt er für jede Datei zwei Einträge: **cf***N* und **df***N*, wobei *N* die Identifikationsnummer des Druckauftrags (**job identification**), ist. Der erste Eintrag enthält Informationen über den Druckauftrag (Eigentümer, Größe), der zweite den eigentlichen Dateiinhalt.

Anschließend wird der Druckerdaemon **lpd** informiert, der dann darüber entscheidet, ob der Druckauftrag an einem lokalen Drucker oder an einem Drucker im Netz ausgeführt werden soll. Im zweiten Fall werden die Dateien in die Druckerschlange des entfernten Rechners eingereiht und die lokalen Dateien gelöscht.

Zur Wartung der Drucker (Stoppen, Starten, usw.) dient das Kommando **lpc** (**line printer control**, nur dem Systemverwalter zugänglich), zur Ausgabe der Druckerschlange **lpq** (**line printer queue**) und zum Löschen von Druckaufträgen **lprm** (**line printer remove**).

Die Ausgabe der Druckerschlange für den voreingestellten Drucker erfolgt durch

```
$ lpq  # oder  lpstat

Mon Oct 16 15:15:15 1995: laser2 is ready and printing

Rank    Pri   Owner      Job    Files              Total Size
active   0    leibner    888    skript.dvi.ps      1024 bytes
1st      0    leibner    110    zpravy             2048 bytes

$
```

Es werden dabei die Anordnung in der Druckerschlange (Rank), die Priorität (Pri), die Eigentümer der Dateien (Owner), die Identifikationsnummern der Druckaufträge (Job), die Dateinamen (Files) und ihre Größe (Total Size), angegeben.

Die Priorität der Druckaufträge kann von System zu System unterschiedlich sein. Manche Systeme arbeiten nach dem FIFO-Prinzip (→ S.40), andere bevorzugen bei der Ausgabe kleine Dateien gegenüber großen.

Um einen Druckauftrag aus der Druckerschlange zu entfernen, ist z.B.

```
$ lprm 110   # oder cancel 110
$
```

einzugeben, wenn 110 die Identifikationsnummer des zu löschenden Druckauftrages ist (s.o., bei Job). Ohne die Angabe einer Nummer wird die aktuelle Ausgabe unterbrochen. Durch **lprm -** werden alle Druckaufträge des Benutzers aus der Druckerschlange gelöscht.

Mit Hilfe einer Pipe können unter anderem z.B. Manualseiten des „on-line-manuals“

```
$ man who | pr -f | lpr
$
```

oder große plakative Texte

```
$ banner 'This is' silly | lpr
$
```

auf dem Drucker ausgegeben werden.

Übung 12 *Geben Sie das Kommando zur zweispaltigen Ausgabe der Dateien* **tel-1** *und* **tel-2** *mit dem Titel „Telefonnummern der Lehrer“, an.*

Übung 13 *Geben Sie den Inhalt Ihres Arbeitsverzeichnisses, sowie den von* **/bin** *und* **/usr/bin**, *mit dem Titel „Arbeitsverzeichnis, /bin und /usr/bin“, auf den Bildschirm aus.*

Übung 14 *Geben Sie das Kommando zum Löschen aller ihrer Druckaufträge in der Druckerschlange für den Drucker* **laser2** *an.*

4.8 Archivierung von Dateien

Zur Archivierung von Dateien auf einer Diskette oder einem Magnetband dient das Kommando **tar** (**tape archiver**).

```
$ pwd
```

```
/usr/home/leibner/verz

$ ls

tel-1  tel-2  verz

$ ls  verz/verz

alpha  beta

$ tar  c  verz
$
```

Die Aktion, die **tar** durchführen soll, wird durch eine **Funktion** (ein Buchstabe) spezifiziert (→ Tab.4.7). Im obigen Beispiel wird durch die Funktion **c** (**create**) eine neues **tar**-Archiv auf einem Magnetband erzeugt und **verz** darauf archiviert. Das Kommando schreibt dabei auf ein Magnetband, daß in dem voreingestellten Laufwerk (**default tape drive**), mit dem Treiber **/dev/rmt0h**, eingelegt ist.

Funktion	Mnemonik	Bedeutung
c	create	Anlegen eines neuen Archivs; Kopieren von Dateien in das Archiv
r	replace	Anfügen von Dateien an das Ende des Archivs
t	table	Durchsuchung des Archivinhalts nach Dateien; Überprüfung der Lesbarkeit des Datenträgers
u	update	Anfügen von Dateien an das Ende des Archivs, falls die Dateien nicht bereits archiviert sind oder ihr Inhalt verändert wurde
x	extract	Übertragung von Dateien aus dem Archiv auf die Festplatte; wird als Name ein Verzeichnisname angegeben, so werden auch alle Unterverzeichnisse kopiert, wird kein Name angegeben, so werden alle Dateien aus dem Archiv kopiert

Tabelle 4.7: Einige Funktionen von **tar**

Durch

```
$ tar  t

verz
verz/verz
verz/verz/alpha
verz/verz/beta
```

```
$
```

kann der Inhalt des Archivs überprüft und durch

```
$ tar  x  verz/verz/alpha
$
```

die Datei **alpha** unter dem relativen Pfad **verz/verz/alpha** zurückgeschrieben werden.

Zur Angabe des Namens bei einem zu archivierenden Verzeichnis sollte niemals ein absoluter Pfadname angegeben werden, da **tar** beim Zurückschreiben versucht, die archivierten Dateien unter dem gleichen Pfadnamen zu erzeugen (der dann auf einem anderen Rechner evtl. nicht vorhanden ist).

Bei relativen Pfadnamen erzeugt **tar** beim Zurückschreiben auf die Festplatte den archivierten relativen Dateibaum im Arbeitsverzeichnis neu, falls dieser nicht bereits vorhanden war. Eine bereits vorhandene Datei wird gegebenenfalls überschrieben.

```
$ ls

tel-1  tel-2

$ tar  xv   # v, siehe unten

x verz
x verz/verz
x verz/verz/alpha, 4 bytes, 1 tape blocks
x verz/verz/beta, 4 bytes, 1 tape blocks

$ ls

tel-1  tel-2  verz

$
```

Neben den Funktionen kennt **tar** noch sogenannte **Modifikatoren** (→Tab.4.8), welche die Funktionen genauer spezifizieren.

Modifikator	Mnemonik	Bedeutung
f	file	der nach den Modifikatoren folgende Parameter wird als spezielle Datei verstanden, in die geschrieben oder aus der gelesen wird (je nach Funktion); falls **-** angegeben wird, wird darunter die Standardeingabe bzw. -ausgabe verstanden
v	verbose	bei Benutzung von **v** wird der Name der gerade bearbeiteten Datei ausgegeben; in Verbindung mit **t** werden weitere Informationen (ähnlich wie bei **ls -l**) hinzugefügt
b	block	Blockfaktor, gibt die Anzahl der Blöcke (512 Byte/Block) an, die „gleichzeitig“ bei der Arbeit mit eine Magnetband gelesen werden, voreingestellt ist ein Blockfaktor von 20
w	wait	Warten auf Bestätigung (ähnlich wie der Schalter **-i** bei z.B. **rm**)

Tabelle 4.8: Wichtigste Modifikatoren von **tar**

Die Funktion **r** erlaubt das Anfügen von Dateien an das Archivende:

```
$ tar  rv  tel-1

tar: blocksize = 20
a tel-1 1 Blocks

$ tar  tv

tar: blocksize = 20
drwxrwxr-x  2112/0        0 Oct 16 08:46:40 1995 verz
drwxrwxr-x  2112/0        0 Oct 12 11:47:53 1995 verz/verz
-rw-rw-r--  2112/0        4 Oct 12 11:12:33 1995 verz/verz/alpha
-rw-rw-r--  2112/0        4 Oct 12 11:12:43 1995 verz/verz/beta
-rw-rw-r--  2112/0      170 Oct  9 11:20:17 1995 tel-1

$
```

Um keine vorhandenen Dateien ungewollt zu überschreiben, kann durch **w** eine Abfrage gestartet werden:

```
$ tar  xwv  tel-1

tar: blocksize = 20
extract tel-1? n

$
```

Mit **tar** können auch Dateien (Verzeichnisse) im Dateibaum kopiert werden (ähnlich wie mit **cp**). Wurde z.B. durch **mount** (→ S.41) eine optische Platte in den Verzeichnisbaum gehängt, so kann auf sie durch

```
$ tar  cf  -  verz | (cd  /optical;  tar  xf  -)
$
```

das Verzeichnis **verz** kopiert werden. Das erste **tar**-Kommando liest dabei **verz** auf die Standardausgabe (**f -**), diese wird über die Pipe weitergereicht, wo dann zunächst in einer neuen SHELL (runde Klammern) ein Verzeichniswechsel auf die optische Platte durchgeführt und anschließend über die Standardeingabe von **tar** (**f -**) das Verzeichnis **verz** auf der optischen Platte angelegt wird.

4.8.1 Komprimieren und Kodieren von Dateien

Eine Reduktion der Größe von Textdatein um 50 bis 60% kann mit dem Kommando **compress** erzielt werden.

```
$ ls

verz

$ tar  cf  verz.tar  verz
$ ls  -l

drwxrwxr-x   2 leibner  system       512 Oct 12 11:47 verz
-rw-rw-r--   1 leibner  system     10240 Oct 17 13:56 verz.tar

$ rm  -r  verz
$ compress  verz.tar
$ ls  -l

-rw-rw-r--   1 leibner  system       370 Oct 17 13:56 verz.tar.Z

$
```

Die komprimierte Datei kann durch

```
$ uncompress  verz.tar.Z
$ ls

verz.tar

$
```

wieder dekomprimiert und durch

```
$ tar  xvf  verz.tar

tar: blocksize = 20
x verz
x verz/alpha, 4 bytes, 1 tape blocks
x verz/beta, 4 bytes, 1 tape blocks

$ ls

verz  verz.tar
```

aus dem Archiv in das Arbeitsverzeichnis zurückkopiert werden.

Alternativ kann zum Komprimieren auch das von GNU[3] entwickelte **gzip** verwendet werden:

```
$ gzip  verz.tar
$ ls  -l

drwxrwxr-x   2 leibner  system       512 Oct 12 11:47 verz
-rw-rw-r--   1 leibner  system       216 Oct 17 13:56 verz.tar.gz

$
```

Der Vergleich der komprimierten Dateien zeigt, daß **gzip** ein besseres Ergebnis (216 zu 370 Byte) erzielt. Die Dekomprimierung erfolgt durch

```
$ gunzip  verz.tar.gz
$
```

Mit **gunzip** können auch durch **compress** komprimierte Dateien dekomprimiert werden.

Soll die komprimierte Datei nach dem Dekomprimieren erhalten bleiben, so kann zum Dekomprimieren **zcat** verwendet werden:

```
$ ls

verz.tar.Z

$ zcat  verz.tar.Z | tar  xvf  -

tar: blocksize = 20
x verz
x verz/alpha, 4 bytes, 1 tape blocks
x verz/beta, 4 bytes, 1 tape blocks

$ ls

verz  verz.tar.Z
```

[3] GNU's not UNIX, UNIX-Entwicklung der „Free Software Foundation“

```
$
```

Mit **zcat** können auch einzelne Dateien aus dem Archiv gelesen werden,

```
$ zcat project.tar.Z | tar xvf - project/main.c
$
```

kopiert z.B. die Datei **main.c** aus dem Archiv **project.tar.Z** nach **project/main.c**.

Um binäre Dateien oder Verzeichnisse (nach **tar** und **gzip**|**compress**) mit elektronischer Post (**e-mail**) verschicken zu können, müssen sie zuvor zu einfachen Textdateien (ASCII) gemacht werden.

Das Kommando **uuencode** wandelt eine binäre Datei in eine ASCII-Datei um und macht sie so für ein Mailprogramm, wie z.B. **mailx**, verschickbar.

```
$ ls

verz.tar.Z

$ uuencode verz.tar.Z verzeichnis.tar.Z >verz.uu
$ ls

verz.tar.Z  verz.uu
```

Die neu angelegte Datei **verz.uu** ist nun lesbar:

```
$ head -2 verz.uu

begin 664 verzeichnis.tar.Z
M'YV0=LK(T0.@H,&#"!,J7,BPH<.'$!'"F'CC1@T0`";2B#$1X\2/,#R"'!D2
```

In der ersten Zeile sind neben dem Wort „begin“ die Zugriffsrechte (664) angeführt, die auch die dekodierte Datei haben wird. Damit der Empfänger der Mail diese auch lesen kann, müssen die Rechte gegebenenfalls mit einem Editor vor dem Verschicken korrigiert werden.

Neben den Zugriffsrechten steht der Name, den die Datei nach dem Dekodieren haben wird.

```
$ uudecode verz.uu  # dekodieren
$ ls
```

```
verz.tar.Z  verz.uu  verzeichnis.tar.Z
```

$ cat verzeichnis.tar.Z | uncompress | tar xf -

$# dekomprimieren, zurückschreiben

$ ls

```
verz  verz.tar.Z  verz.uu  verzeichnis.tar.Z
```

$

Mehrere Dateien können z.B. durch

$ tar cf - alpha beta | gzip | uuencode dateien.tar.gz >dateien.uu

$ ls

```
alpha  beta  dateien.uu
```

$ rm alpha beta

$

archiviert, komprimiert und kodiert werden. Die ursprünglichen Dateien gewinnt man durch

$ uudecode dateien.uu

$ ls

```
dateien.tar.gz  dateien.uu
```

$ cat dateien.tar.gz | gunzip | tar xf -

$ ls

```
alpha  beta  dateien.tar.gz  dateien.uu
```

$ rm dat*

$ ls

```
alpha  beta
```

$

wieder zurück.

4.9 Verweise

Auf jeden i-node können mehrere Namen verweisen (**links**). In einem Verzeichnis kann es folglich mehrere Dateinamen mit gleichem i-node geben. Auf ein und dieselben physikalischen Daten kann dann unter unterschiedlichen Namen zugegriffen werden. Dies kann dann von Vorteil sein, wenn z.B. mehrere Benutzer an einem gemeinsamen Projekt in unterschiedlichen Heimatverzeichnissen arbeiten, oder wenn in einem komplexen Dateibaum von unterschiedlichen Stellen der Zugriff auf bestimmte Daten möglich sein soll, ohne daß dazu jeweils das Arbeitsverzeichnis gewechselt oder lange Pfadnamen angegeben werden müssen.

Ein weiterer Vorteil eines Verweises ist sein geringer Platzbedarf.

Es wird zwischen zwei Verweisen unterschieden.

Verzeichniseinträge, die einen neuen Namen für eine bereits existierende Datei definieren, werden durch einen sogenannten „**hard link**“ erzeugt.

```
$ ls  -li

141315 -rw-rw-r--   1 leibner  system       170 Oct 11 08:51 alpha

$ ln  alpha  alphalink
$ ls  -li

141315 -rw-rw-r--   2 leibner  system       170 Oct 11 08:51 alpha
141315 -rw-rw-r--   2 leibner  system       170 Oct 11 08:51 alphalink

$
```

Das Kommando **ln** erzeugt einen neuen Eintrag im aktuellen Verzeichnis (→ Abb.4.6).

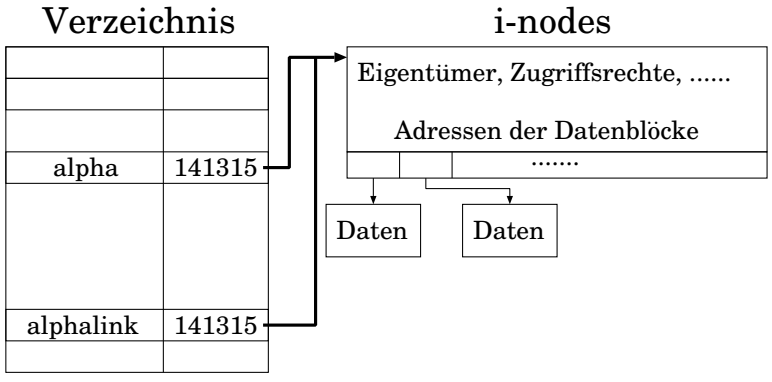


Abbildung 4.6: Verzeichniseinträge nach Eingabe von **ln alpha alphalink**

Die Zahl, die den Zugriffsrechten folgt, gibt die Anzahl der Verweise auf eine Datei wieder. War die Anzahl der Verweise auf **alpha** vor der Einrichtung des „links“ 1, so erhöht sie sich für **alpha** und **alphalink** danach auf 2. Wird einer der Namen gelöscht, so reduziert sich dieser Wert wiederum um 1:

```
$ rm  alpha
$ ls  -li

141315 -rw-rw-r--   1 leibner  system       170 Oct 11 08:51 alphalink

$
```

Da für ein Verzeichnis durch **.** immer ein zweiter Name existiert (→ Abb.4.3), ist die Anzahl der Verweise bei Verzeichnissen immer mindestens 2.

```
$ ls  -l

drwxrwxr-x   2 leibner  system       512 Oct 11 10:42 verz

$
```

Enthält ein Verzeichnis weitere Unterverzeichnisse, so wird durch die **..** der Unterverzeichnisse die Anzahl der Verweise weiter erhöht:

```
$ ls  -l  verz

drwxrwxr-x   2 leibner  system       512 Oct 11 10:42 uverz1
drwxrwxr-x   2 leibner  system       512 Oct 11 10:42 uverz2

$ ls  -l

drwxrwxr-x   4 leibner  system       512 Oct 11 10:42 verz

$
```

Da die Nummern der i-nodes nur innerhalb eines Dateisystems eindeutig sind, ist ein „hard link“ auf ein Dateisystem beschränkt.

Verweise auf Dateien, die nicht im aktuellen Dateisystem enthalten sind, können durch sogenannte „**symbolic links**“ erzeugt werden. Solche Verweise sind auch auf Dateien möglich, die auf vernetzten Rechnern liegen.

```
$ ls  -li

141315 -rw-rw-r--   1 leibner  system       170 Oct 11 08:51 alpha
```

$ ln -s alpha alphalink

```
141315 -rw-rw-r--   1 leibner system 170 Oct 11 08:51 alpha
141323 lrwxrwxrwx   1 leibner system   5 Oct 12 10:39 alphalink -> alpha

$
```

Durch den Schalter **-s** wird eine Datei **alphalink** erzeugt, deren Inhalt 5 Byte ist (die Zeichen **a l p h a**). Symbolische Verweise haben als Dateityp den Eintrag `l`. Bei der Arbeit mit ihnen wird nie auf ihren Inhalt, sondern immer auf die Datei, deren Namen sie enthalten, zugegriffen. Wird diese Datei gelöscht, so kann mit dem symbolischen Verweis nicht weitergearbeitet werden.

$ rm alpha
$ ls -li

```
141323 lrwxrwxrwx   1 leibner system 5 Oct 12 10:39 alphalink -> alpha
```

$ more alphalink

```
alphalink: No such file or directory

$
```

Neben dem Schalter **-s** gibt es noch **-n** und **-f**. Bei **-n** wird kein Verweis erzeugt, wenn es bereits eine Datei gleichen Namens gibt, bei **-f** wird die Erzeugung auch dann erzwungen.

Das Kommando **ln** kann auch in Verbindung mit einem Verzeichnis verwendet werden. Dabei werden für die angegebenen Dateien im Verzeichnis Verweise mit identischen Namen erzeugt.

$ ls -li

```
141315 -rw-rw-r--   2 leibner  system         4 Oct 12 11:12 alpha
141323 -rw-rw-r--   2 leibner  system         4 Oct 12 11:12 beta
176641 drwxrwxr-x   2 leibner  system       512 Oct 12 11:47 verz
```

$ ls -l verz

```
total 0
```

$ ln alpha beta verz
$ ls -li verz

```
141315 -rw-rw-r--   2 leibner  system         4 Oct 12 11:12 alpha
141323 -rw-rw-r--   2 leibner  system         4 Oct 12 11:12 beta
```

```
$
```

In der nachfolgenden Tabelle 4.9 ist eine Übersicht zu den Dateitypen und den Kommandos zu ihrem Anlegen und Löschen gegeben.

Dateityp	**Symbol**	**Erzeugung**	**Löschen**
gewöhnliche Datei	`-`	Editoren, **cp**	**rm**
Verzeichnis	`d`	**mkdir**	**rmdir**, **rm -r**
spez. Datei, zeichenorientiert	`c`	**mknod**	**rm**
spez. Datei, blockorientiert	`b`	**mknod**	**rm**
Pipe mit Namen	`p`	**mknod**	**rm**
symbolischer Verweis	`l`	**ln -s**	**rm**
Domain Socket (BSD)	`s`	**socket(2)**	**rm**, **unlink**

Tabelle 4.9: Kommandos zum Anlegen und Löschen von Dateien

4.10 Suchen im Dateibaum

Das Kommando **find** dient dem Durchsuchen des Dateibaums und der Auswahl von Dateien, die bestimmten Kriterien genügen.

Mehrere Kriterien zusammen bilden einen logischen Ausdruck, der den Wert **true** annehmen muß, sollen bestimmte Operationen auf die ausgesuchten Dateien angewendet werden.

Als Startpunkt der Suche kann ein beliebiger Ort im Dateibaum gewählt werden, **find** durchsucht dann rekursiv alle darunterliegenden Verzeichnisse, mit Ausnahme von Verzeichnissen, die über symbolische Verweise definiert sind.

Die Wirkung nachfolgender Kriterien (→ Tab.4.10) kann unterschiedlich sein.

Einige der Kriterien wählen bestimmte Dateien aus (z.B. nach dem Namen, Eigentümer, den Zugriffsrechten, der Größe), andere führen eine bestimmte Operation mit den ausgewählten Dateien durch (z.B. **print**, **prune**, **exec**).

Sind die einzelnen Kriterien durch Leerzeichen getrennt, so entspricht dies logisch der Konjunktion, eine Trennung durch **-o** (**or**) bedeutet Disjunktion.

Jedes Element der Kriterien ist ein eigenständiger Ausdruck und muß daher von den anderen durch ein Leerzeichen getrennt werden. Dies gilt auch für Ausrufezeichen (Negation), Klammern, usw. Falls spezielle Zeichen der SHELL verwendet werden (z.B. runde und eckige Klammern, Fragezeichen oder Stern), so müssen Sie in Hochkommas oder Anführungszeichen eingeschlossen werden. Wird die spezielle Bedeutung des Zeichens für die SHELL durch `\` aufgehoben, so muß vor- und nachher ein Leerzeichen stehen (z.B. `" \[ "`).

Am häufigsten wird **find** zum Suchen von Dateien im Dateibaum verwendet.

```
$ find . -name alpha -print
```

```
./verz/alpha
```

Als Startpunkt für die Suche wurde im obigen Beispiel das Arbeitsverzeichnis gewählt (**.**), als Kriterien **-name**, **alpha** und **-print**. Die Kriterien sind durch Leerzeichen voneinander getrennt, was einer Konjunktion gleichkommt. Erst wenn alle drei den Wert **true** annehmen, wird der Pfad zum gesuchten Dateinamen ausgegeben. Die Kriterien **-name** und **-print** müssen daher den Wert **true** bereits voreingestellt haben, der Wert von **alpha** wird **true**, sobald eine Datei mit diesem Namen gefunden wird.

Operator	Bedeutung
-print	logischer Wert **true**; die Wirkung des Operators ist die Bildschirmausgabe der Dateinamen, die mit Hilfe bestimmter Kriterien ausgesucht wurden
-prune	logischer Wert **true**; bricht die Suche ab, wenn bestimmte Kriterien erfüllt sind; ist der aktuelle Pfadname ein Verzeichnis, wechselt **find** nicht in das Verzeichnis
-ls	logischer Wert **true**; gibt für den gefundenen Pfadnamen genauere Informationen aus (wie **ls -glids**)
-name *dateiname*	der Operator erhält den Wert **true**, falls der Name der Datei *dateiname* ist; *dateiname* kann mit Hilfe der Sonderzeichen `?`, `*` und `[]`, im Sinne von Abschnitt 4.3 angegeben werden, es muß dabei jedoch ihre Interpretation durch die SHELL mittels Anführungszeichen `"` oder Hochkommas `'` ausgeschlossen werden
-perm *ooo*	der Operator erhält den Wert **true**, falls die Zugriffsrechte der Datei dem oktalen Wert *ooo* entsprechen
-type *c*	der Operator erhält den Wert **true**, falls die Datei von Typ *c* ist (→ Tab.4.9)
-links *n*	der Operator erhält den Wert **true**, falls die Datei *n* Verweise hat
-user *name*	der Operator erhält den Wert **true**, falls der Eigentümer der Datei *name* ist
-size *n*[c]	der Operator erhält den Wert **true**, falls die Größe der Datei *n* Blöcke (512 Byte/Block) ist; folgt *n* das Zeichen c, dann gibt die Zahl *n* die Größe der Datei in Byte an
-exec *kommando*	auf die ausgewählten Dateien wird das *kommando* angewendet; der Operator *exec* nimmt den Wert **true** an, wenn *kommando* den Wert 0 (**exit(0)**) zurückgibt; der Text von *kommando* muß mit `\;` beendet werden, das Argument `{}` repräsentiert die ausgewählten Dateinamen
-ok *kommando*	entspricht **exec**, vor der Ausführung von *kommando* wird jedoch eine Abfrage gestartet
(*ausdruck*)	der Wert des Operators wird **true**, wenn *ausdruck* **true** wird; da die runden Klammern Sonderzeichen der SHELL sind, muß ihre Sonderrolle durch ein vorangestelltes `\` aufgehoben werden
`!`	Negation
-a	Operator der Konjunktion (Voreinstellung)
-o	Operator der Disjunktion

Tabelle 4.10: Operatoren des Kommandos **find**

Alle Dateien, deren Name unter dem Heimatverzeichnis mit **ta** endet, erhält man durch

```
$ find ~/ -name '*ta' -print

/usr/home/leibner/verz/verz/verz/beta

$
```

Dateien, die größer als 5000 Blöcke sind, werden durch

```
$ find ~/ -size +5000 -print

/usr/home/leibner/NETZ/texte/README
/usr/home/leibner/NETZ/obrazky.uu
/usr/home/leibner/NETZ/pictures/julia_bw.ps

$
```

ausgegeben. Ist eine Zahl n Teil des Operators (wie eben), so bedeutet $+n$ größer als n, $-n$ kleiner als n und n genau n.

Alle Dateien, die mit **.dvi** enden, werden in allen Unterverzeichnissen des Heimatverzeichnisses durch

```
$ find ~/ -name '*.dvi' -exec rm {} \;
$
```

gelöscht. Durch

```
$ find ~/ \( -name 'tel*' -o -name 'al*' \) -print \
-exec head -2 {} \;

/usr/home/leibner/red/projekt/alla
read_blif -f C.dir.blif -mark
write_blif -misII -f Blif
/usr/home/leibner/verz/tel-1
Ingrid Beck                 22223
Renate Krauss               26844
/usr/home/leibner/verz/tel-2
Lothar Haschigk             44834
Irmgard Kalasch             44835
/usr/home/leibner/verz/verz/verz/alpha
erste Zeile
zweite Zeile
```

```
$
```

wird nach Dateien gesucht, die unterhalb des Heimatverzeichnisses liegen und deren Namen mit **tel** oder **al** beginnen. Werden solche Dateien gefunden, so wird entweder **'tel*'** oder **'al*' true**, worauf die Ausgabe ihres Namens und der ersten zwei Zeilen des Dateiinhalts erfolgt.

Übung 15 *Geben Sie das Kommando an, das vom aktuellen Arbeitsverzeichnis aus alle Dateien mit den Eigentümern* **jim** *oder* **jack** *sucht, die Dateinamen auf den Bildschirm ausgibt und anschließend die Dateien löscht.*

Übung 16 *Geben Sie das Kommando zum Suchen aller Dateien, die größer 10000 Byte sind, nach der Eingabe von* **y** *gelöscht werden sollen und unter* **/home/jim** *und* **/home/jack** *stehen, an.*

Übung 17 *Geben Sie ein Kommando an, das alle Informationen über die Unterverzeichnisse des aktuellen Arbeitsverzeichnisses ausgibt (nicht die Inhalte der Unterverzeichnisse, → Tab.4.11).*

4.11 Schalter des Kommandos **ls**

Zum Abschluß dieses Kapitels sollen noch die wichtigsten Schalter des Kommandos **ls** zusammengefaßt werden.

Schalter	Mnemonik	Bedeutung
-a	all	gibt auch Dateien aus, die mit `.` beginnen
-b	–	nichtdarstellbare Zeichen werden als oktale Zahl *ooo* ausgegeben
-c	change	benutzt den Zeitpunkt der letzten i-node-Änderung (Erzeugung, Modifikation, Änderung der Zugriffsrechte) beim Sortieren, wenn gleichzeitig **-t** angegeben wird
-d	directory	gibt bei Verzeichnissen nur den Verzeichnisnamen, nicht den Inhalt aus
-F	–	hängt an den Namen bei Verzeichnissen `/`, bei ausführbaren Dateien `*` und bei symbolischen Verweisen `@` an
-i	i-node	gibt die Nummer des i-nodes aus
-l	long	gibt genauere Informationen über die Datei aus (→ S.36)
-L	link	verfolgt symbolische Verweise und gibt Informationen über die Dateien aus, auf welche der „link“ verweist
-m	modified	gibt die Dateinamen, durch Kommata getrennt, aus
-n	numeric	wie **-l**, gibt jedoch für die Eigentümer und Gruppen die **uid**- und **gid**-Nummern aus
-R	recursive	wechselt rekursiv in alle Unterverzeichnisse
-q	–	nichtdarstellbare Zeichen werden als `?` dargestellt
-s	size	gibt die Größe der Dateien in Blöcken an (512 Byte/Block)
-t	time	gibt die Dateien nach ihrer Entstehungszeit geordnet aus (neueste zuerst)
-1	–	einspaltige Ausgabe (implizit, wenn nicht auf den Bildschirm geschrieben wird)

Tabelle 4.11: Schalter des Kommandos **ls**

5 Umlenkung der Standardeingabe und -ausgabe

Für UNIX sind sowohl die Tastatur als auch der Bildschirm Dateien. Die Ausgabe auf den Bildschirm erfolgt über **/dev/tty**:

```
$ cp  tel-1  /dev/tty

Ingrid Beck                22223
Renate Krauss              26844
Angela Sczesny             25820
Gabriele Heiss             27372
Helga Lohmann              26113

$
```

Dabei ist **/dev/tty** ein Synonym für das angeschlossene Terminal **/dev/ttyp1**

```
$ tty

/dev/ttyp1

$
```

Zum Schreiben auf den eigenen Bildschirm kann daher immer **/dev/tty** benutzt werden.

Die meisten Kommandos unter UNIX haben einen interaktiven Charakter, sie lesen standardmäßig von der Tastatur (**standard input**, **stdin**) und schreiben auf den Bildschirm (**standard output**, **stdout**). Aufgrund dieser Eigenschaft werden sie Filter genannt.

Jedem Prozeß (der ein Kommando ausführt) stehen zur Manipulation mit Dateien 50-60 Kanäle zur Verfügung. Beim Öffnen einer Datei wird gleichzeitig ein Kommunikationskanal geöffnet und die Kanalnummer in einer Tabelle hinterlegt. Der zur Ausführung des Kommandos gebildete neue Prozeß erbt diese Tabelle.

Sind keine weiteren Kanäle geöffnet, so enthält die Tabelle minimal drei Einträge: Den Kanal **0**, der als Standardeingabe zum Lesen von der Tastatur geöffnet wird, den Kanal **1**, der zum Schreiben auf den Bildschirm dient und Kanal **2**, der Fehlermeldungen (**standard error**, **stderr**) standardmäßig ebenfalls auf den Bildschirm ausgibt.

Die Manipulation der Standardeingabe und -ausgabe ist mit Sonderzeichen der SHELL möglich.

Zwei Arten der Manipulation werden unterschieden: Die erste wird als Umlenkung der Standardeingabe und -ausgabe bezeichnet, die zweite als Pipe.

> Mit Hilfe des Sonderzeichens > kann die Standardausgabe eines Kommandos in eine Datei umgelenkt werden.

```
$ ls -l >verzeichnisinhalt
$ cat verzeichnisinhalt

-rw-rw-r--   1 leibner  system       170 Oct  9 11:20 tel-1
-rw-rw-r--   1 leibner  system       340 Oct  9 11:22 tel-2
drwxrwxr-x   3 leibner  system       512 Oct 19 11:38 verz
-rw-rw-r--   1 leibner  system         0 Oct 23 13:44 verzeichnisinhalt

$
```

Die Umlenkung erfolgt auf SHELL-Ebene, das Programm **ls** merkt nichts davon. Die Datei **verzeichnisinhalt** wird beim Interpretieren der Eingabezeile durch die SHELL angelegt, sie ist daher vor dem Ausführen von **ls** bereits im Verzeichnis, wenn auch noch leer, vorhanden.

>> Wird die Standardausgabe in eine bereits vorhandene Datei umgelenkt, so wird deren ursprünglicher Inhalt überschrieben. Die Umlenkung kann jedoch auch so erfolgen, daß der ursprüngliche Text erhalten und der neue lediglich angehängt wird.

```
$ echo "Inhalt vom: \c" >>verzeichnisinhalt
$ date >>verzeichnisinhalt
$ cat verzeichnisinhalt

-rw-rw-r--   1 leibner  system       170 Oct  9 11:20 tel-1
-rw-rw-r--   1 leibner  system       340 Oct  9 11:22 tel-2
drwxrwxr-x   3 leibner  system       512 Oct 19 11:38 verz
-rw-rw-r--   1 leibner  system         0 Oct 23 14:04 verzeichnisinhalt
Inhalt vom:  Mon Oct 23 14:06:03 MET 1995

$
```

***n*>** Alle oben erwähnten Kommunikationskanäle können durch die Angabe ihrer Kanalnummer vor dem jeweiligen Sonderzeichen umgeschaltet werden.

Um z.B. Fehlermeldungen (fehlende Leseberechtigung) beim Kommando **find** von der gewünschten Information zu trennen, kann Kanal **2** in eine eigene Datei umgelenkt werden.

```
$ find / -print >inhalt 2>fehler
$ head -3 inhalt

/
/dev
/dev/MAKEDEV

$ head -3 fehler

find: cannot open < /etc/auth >
find: cannot open < /sbin/rc0.d >
find: cannot open < /sbin/rc2.d >

$
```

Um sowohl die Standardausgabe als auch die Fehlermeldungen in ein und dieselbe Datei zu schreiben, kann Kanal **2** mit Kanal **1** verbunden werden: *n*>&*m*

```
$ find / -print >alles 2>&1
$ head -3 alles

find: cannot open < /etc/auth >
/
/dev

$
```

Dabei ist Vorsicht geboten. Schreibt man z.B.

```
$ find / -print 2>&1 >alles
```

so erzielt man nicht den gewünschten Effekt, da durch **2>&1** lediglich Kanal **2** auf Kanal **1** geschaltet wird, wobei Kanal **1** aber nach wie vor der Bildschirm ist. Da erst durch **1>alles** die Standardausgabe auf die Datei **alles** umgelenkt wird, werden Fehlermeldungen weiterhin auf den Bildschirm ausgegeben.

Genauso wie die Standardausgabe in eine Datei umgelenkt werden kann, so kann auch die Standardeingabe von einer Datei kommen. Die Umlenkung ist jedoch nur selten notwendig, da die meisten Kommandos die direkte Übergabe einer Datei gestatten. So ist z.B. <

```
$ cat <tel-1
```

gleichwertig mit

$ cat tel-1

Die Filterwirkung eines Kommandos kann man sehr gut bei **cb** (**C-beautifier**) beobachten. Das Kommando liest von der Standardeingabe den Quellkode eines C-Programms und gibt auf die Standardausgabe ein um Einrückungen und Leerzeilen „verschönertes“ C-Programm aus, das dadurch übersichtlicher und besser lesbar wird.

$ cb <noten.c >noten.pretty.c

Wie bei >, so gibt es auch bei < ein <<. Dieses Sonderzeichen wird in der Regel nur in Scripts verwendet, wo es zur Umlenkung der Standardeingabe innerhalb des Scripts verwendet wird. Da die Eingabe innerhalb des Scripts steht, wird sie lokales Dokument (**here document**) genannt. <<

```
$ wc  -c  <<EOF
> das Kommando wc zaehlt
> die Anzahl der Zeichen
> dieses Textes
> EOF
        60
$
```

Durch die Zeichenkette **EOF** wird der Text begrenzt, der dem Kommando **wc** als Standardeingabe übergeben wird.

Die Anwendung innerhalb eines SHELL-Scripts zeigt das nachfolgende Beispiel:

```
# dieses script gibt den Text bis // aus
cat  << //
This shell script is currently under development, please
report any problems to Peter.Leibner@pd.siemens.de
//
```

□

```
$ . skript

This shell script is currently under development, please
report any problems to Peter.Leibner@pd.siemens.de

$
```

Durch **. skript** wird das SHELL-Script in der aktuellen SHELL interpretiert (es wird dazu kein neuer Prozeß gestartet). Der Text wird nach dem Lesen von // an das Kommando **cat** übergeben, **cat** gibt dann den Text auf die Standardausgabe aus.

In der nachfolgenden Tabelle 5.1 sind alle Symbole zum Umlenken der Standardeingabe und -ausgabe zusammengefaßt.

5.1 Verbindung von Filtern zu einer Kolonne

Die Datenkanäle zweier Prozesse können mit Hilfe einer Pipe zusammengeschaltet werden. Der Prozeß **PRODUZENT** schreibt seine Daten auf die Standardausgabe, der Prozeß **KONSUMENT** liest von der Standardeingabe. Mit dem Sonderzeichen | bildet die SHELL eine Pipe.

PRODUZENT | KONSUMENT

Den gleichen Effekt erreicht man durch die Umlenkung der Standardeingabe und -ausgabe über eine Hilfsdatei:

```
$ PRODUZENT  >tmp
$ KONSUMENT  <tmp
$ rm  tmp
$
```

Symbol	Bedeutung
>*dateiname*	Umlenken der Standardausgabe in die Datei *dateiname*
>>*dateiname*	Anhängen der Standardausgabe an das Ende der Datei *dateiname*
<*dateiname*	Umlenken der Standardeingabe aus der Datei *dateiname*
<<*zeiket*	**here document**
n>*dateiname*	Umlenken der Standardausgabe aus Kanal *n* in die Datei *dateiname*
n>>*dateiname*	Anhängen der Standardausgabe aus Kanal *n* an das Ende der Datei *dateiname*
n>&*m*	Verbindung der Standardausgaben der Kanäle *n* und *m*
m<&*n*	Verbindung der Standardeingaben der Kanäle *n* und *m*
<&-	Schließen des Standardeingabekanals
>&-	Schließen des Standardausgabekanals

Tabelle 5.1: Sonderzeichen der SHELL zur Umlenkung von stdin/stdout

Im Gegensatz zur Umlenkung über die Hilfsdatei, die auf der Festplatte liegt, wird die Pipe im Ausgleichsspeicher des Hauptspeichers realisiert.

Die Prozesse **PRODUZENT** und **KONSUMENT** laufen gleichzeitig. Die Pipe hat eine beschränkte Kapazität. Wurde sie vom Prozeß **PRODUZENT** aufgefüllt, so wird dieser so lange gestoppt, bis der Prozeß **KONSUMENT** die Daten gelesen hat. Ist sie leer, so wird umgekehrt verfahren.

Mehrere Filter können mit Pipes zu einer Kolonne (**Pipeline**) verbunden werden.

Mit dem Kommando **grep** kann man aus der Eingabedatei Zeilen auswählen, die ein bestimmtes Muster erfüllen. So wird z.B. durch

```
$ cat tel* | grep '^K' | wc -l

        2
```

die Anzahl der Namen aus den Dateien **tel*** ausgegeben, deren Vorname (erstes Zeichen in der Zeile) mit **K** beginnt.

Zum Sichern von Zwischenergebnissen aus Kolonnen dient das Kommando **tee**. Sein Name stammt aus der Terminologie der Installateure. Dort wird mit „tee“ ein T-Stück in einem Rohr (Pipe) bezeichnet. Dieses T-Stück leitet die Standardeingabe in eine Datei, die als Parameter dem Kommando **tee** übergeben wird und gibt sie gleichzeitig auf die Standardausgabe weiter.

```
$ cat tel* | grep '^K' | tee alleKs | wc -l

        2

$ cat alleKs

Karlheinz Reichenstetter    63325
Klaus Renno                 63315

$
```

6 Textbearbeitung

Da UNIX in einer Zeit entstand, in der es noch keine graphischen Benutzeroberflächen gab, wurde besonderes Augenmerk auf Kommandos gelegt, die eine Bearbeitung von Textdateien gestatten.

Einige dieser Kommandos werden im folgenden vorgestellt. Es handelt sich allesamt um Filter, die zum Sortieren, Durchsuchen, Vergleichen oder Formatieren von Textdateien, verwendet werden können.

6.1 Aufteilen von Textdateien

Zum Aufteilen einer Textadatei auf mehrere, dient das Kommando **split**.

```
$ ls

leute

$ cat  leute

Peter Ott               321     Muenchen
Franz Hecht             455     Augsburg
Maria Diepold           324     Hamburg
Jana Baur               788     Sontheim
Lukas Ondracek          112     Hannover
Vitus Gsell             692     Fuessen
Isidor Wolf             421     Mindelheim

$ split  -2  leute  zwei
$ ls

leute   zweiaa  zweiab  zweiac  zweiad

$ cat  zweiac

Lukas Ondracek          112     Hannover
Vitus Gsell             692     Fuessen
```

```
$
```

Die Datei **leute** wird, bedingt durch **-2**, auf zwei Zeilen lange Dateien aufgeteilt (bis auf die letzte, die nur eine Zeile enthält), an deren Namen **zwei** zur Unterscheidung **aa**, **ab**, **ac**, usw., angehängt werden (→ Tab.6.1).

Schalter	**Bedeutung**
-*n*	Aufteilung in *n*-Zeilen lange Dateien, voreingestellt ist der Wert 1000
dateiname	Name der aufzuteilenden Datei; wird *dateiname* nicht angeführt oder steht an seiner Stelle **-**, so wird von der Standardeingabe gelesen
name	Name der Ausgabedateien; die erste heißt *name***aa**, die zweite *name***ab**, usw., bis *name***zz**; *name* darf nicht länger als 12 Zeichen sein, die maximale Anzahl der Ausgabedateien ist 676; wird *name* nicht angegeben, so beginnen die Ausgabedateien mit dem Buchstaben **x**

Tabelle 6.1: **split** [**-***n*] [*dateiname* [*name*]]

Die Texteingabe von der Tastatur kann über **split** auf Dateien aufgeteilt werden:

```
$ split  -1
erste Datei xaa
zweite Datei xab
dritte Datei xac
$ ls

leute  xaa  xab  xac  zweiaa  zweiab  zweiac  zweiad

$ cat  xab

zweite Datei xab

$
```

Die spaltenweise Bearbeitung von Dateien ermöglicht das Kommando **cut**. Mit dem Schalter **-c** (**character**) können aus den Spalten bestimmte Zeichen ausgewählt werden. Sind z.B. in der Datei **leute** nur das erste, das dritte und alle Zeichen ab dem 22-ten (einschließlich), von Interesse, so ist

```
$ cut  -c1,3,22-  leute
```

```
Pten
Faburg
Mrmburg
Jnim
Lkannover
Vtsen
Iielheim


$
```

einzugeben.

Mit dem Schalter **-f** (**field**) können aus der Eingabedatei Felder ausgewählt werden. Unter einem Feld wird ein Zeilenbereich verstanden, der vom nächsten durch ein Tabulatorzeichen (**TAB**) getrennt ist. Durch

```
$ cut  -f1,4  leute
```

```
Peter Ott       Muenchen
Franz Hecht     Augsburg
Maria Diepold   Hamburg
Jana Baur       Sontheim
Lukas Ondracek  Hannover
Vitus Gsell     Fuessen
Isidor Wolf     Mindelheim


$
```

werden z.B. das erste und vierte Feld der Datei **leute** ausgegeben. Zwischen zwei aufeinanderfolgenden Trennzeichen wird ein leeres Feld angenommen (→ Abb.6.1), die Ortsnamen stehen daher erst im vierten Feld.

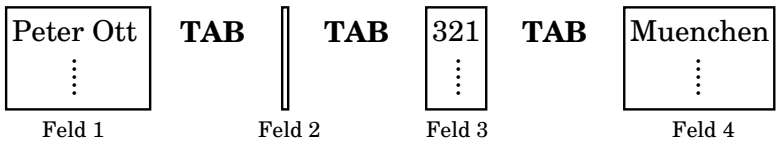


Abbildung 6.1: Felder der Datei **leute**

Schalter	Bedeutung
-c*menge*	selektiert einzelne Spalten oder einen Spaltenbereich; die einzelnen Positionen in *menge* müssen durch Kommas getrennt und in aufsteigender Reihenfolge sein; der Bereich *n1-n2* bestimmt alle Spalten von *n1* bis *n2*, einschließlich der Bereichsgrenzen, abgekürzt kann *-n* für *1-n*, bzw. *n-* für *n-*„letzte Spalte“ geschrieben werden
-f*menge*	selektiert einzelne Felder; ein Feld besteht aus allen Zeichen, die sich zwischen zwei Trennzeichen befinden, zwischen zwei aufeinanderfolgenden Trennzeichen steht ein leeres Feld; das voreingestellte Trennzeichen **TAB** kann durch **-d***zeichen* umdefiniert werden
-d*zeichen*	als Trennzeichen wird *zeichen* verwendet; falls *zeichen* ein Leerzeichen oder ein Sonderzeichen der SHELL sein soll, so muß es in Hochkommas bzw. Anführungszeichen eingeschlossen werden
-s	Zeilen, die kein Trennzeichen enthalten, werden nicht ausgegeben; ist **-s** nicht angegeben, so erfolgt ihre Ausgabe unbearbeitet

Tabelle 6.2: Wichtige Schalter des Kommandos **cut**

Wird durch **-d' '** als neues Trennzeichen das Leerzeichen definiert, so besteht die Datei **leute** bezüglich dieses Trennzeichens nur noch aus zwei Feldern (ein Leerzeichen steht nur zwischen den Vor- und Nachnamen):

```
$ cut  -f2  -d' '  leute

Ott             321     Muenchen
Hecht           455     Augsburg
Diepold         324     Hamburg
Baur            788     Sontheim
Ondracek        112     Hannover
Gsell           692     Fuessen
Wolf            421     Mindelheim

$
```

Übung 18 *Geben Sie aus der Datei* **/etc/passwd** *nur Ihren Benutzernamen und Ihre* uid *aus.*

6.2 Verbinden von Textdateien

Mehrere Textdateien können mit **paste** zu einer verbunden werden. Die Verbindung erfolgt zeilenweise.

```
$ ls

dat1  dat2

$ cat  dat1

erste dat1
zweite dat1
dritte dat1

$ cat  dat2

erste dat2
zweite dat2
dritte dat2

$ paste  dat1  dat2

erste dat1      erste dat2
zweite dat1     zweite dat2
dritte dat1     dritte dat2

$
```

Die Zeilen der Eingabedateien werden durch ein Tabulatorzeichen getrennt. Mit dem Schalter **-d** können ein oder mehrere Trennzeichen definiert werden.

```
$ paste  -d@  dat1  dat2

erste dat1@erste dat2
zweite dat1@zweite dat2
dritte dat1@dritte dat2

$
```

Mehrere Trennzeichen (→ Tab.6.3) werden zyklisch benutzt:

```
$ cut  -f1  leute  >sp1
$ cut  -f3  leute  >sp3
$ cut  -f4  leute  >sp4
$ cat  sp1
```

```
Peter Ott
Franz Hecht
Maria Diepold
Jana Baur
Lukas Ondracek
Vitus Gsell
Isidor Wolf
```

$ cat sp3

```
321
455
324
788
112
692
421
```

$ cat sp4

```
Muenchen
Augsburg
Hamburg
Sontheim
Hannover
Fuessen
Mindelheim
```

$ paste -d'&@' sp4 sp3 sp1

```
Muenchen&321@Peter Ott
Augsburg&455@Franz Hecht
Hamburg&324@Maria Diepold
Sontheim&788@Jana Baur
Hannover&112@Lukas Ondracek
Fuessen&692@Vitus Gsell
Mindelheim&421@Isidor Wolf
```

Mit Hilfe des Schalters **-s** werden die Zeilen einer Datei zu einer Zeile verbunden.

$ paste -s sp3

```
321     455     324     788     112     692     421
```

```
$
```

Wird für den Dateinamen **-** verwendet, so liest **paste** von der Standardeingabe:

```
$ ls | paste  -d'\t\t\n'  -s  -

aaaab.gif        ab4aba.gif       ab6a2b.gif
ac4ab5a.gif      apfel3D.gif      ba.gif
bac5abd2ab.gif   bac7ab3a.gif     c3ab.gif
cg3a.gif         d4a2b2a.gif      d4a2baba.gif
julia1.gif       julia_bw.gif     julia_bw.ps
mandel62.gif     satt0.jpg        satt1.jpg
satt2.gif        satt3.gif

$
```

Im Beispiel wird die Standarausgabe von **ls** (→ Tab.4.11) über eine Pipe als Standardeingabe (**-**) an **paste** weitergereicht, **paste** fügt zyklisch zwei Tabulatorzeichen und ein Zeichen für eine neue Zeile ein und gibt das Ergebnis auf die Standardausgabe aus.

Schalter	**Bedeutung**
-s	die Verbindung der Dateien erfolgt seriell, der Inhalt jeder Datei wird als eine Zeile auf die Standardausgabe gegeben
-d*menge*	definiert die Menge der Zeichen, die zyklisch als Trennzeichen verwendet werden; Voreinstellung: **TAB**, Leerzeichen und Sonderzeichen der SHELL müssen in Hochkommas oder Anführungszeichen eingeschlossen werden; zusätzlich können **\n** (neue Zeile), **\t** (Tabulator), **** und **\0** (leere Zeichenkette) verwendet werden
dateiname	Name, bzw. Namen der Eingabedateien, bei **-** wird von der Standardeingabe gelesen

Tabelle 6.3: **paste** [**-s**] [**-d***menge*] *dateiname...*

Die Inhalte mehrerer Dateien werden durch **-s** zeilenweise ausgegeben:

```
$ paste  -s  dat1  dat2

erste dat1      zweite dat1     dritte dat1
erste dat2      zweite dat2     dritte dat2

$
```

6.3 Sortieren von Textdateien

Das Kommando **sort** ist eines der Filter mit den meisten Schaltern. Im folgenden werden nur die wichtigsten angeführt, eine vollständige Beschreibung findet man z.B. im „on-line-manual“.

Ohne Schalter sortiert **sort** entsprechend der Reihenfolge in der ASCII-Tabelle:

```
$ cat  aleute

Peter Otte      10       Muenchen
Joshua Baur     788      Sontheim
Lukas Ondracek 112       Hannover
%Vitus Gsell    692      Fuessen
Isidor Wolf     421      Mindelheim

$ sort  aleute

%Vitus Gsell    692      Fuessen
Isidor Wolf     421      Mindelheim
Joshua Baur     788      Sontheim
Lukas Ondracek 112       Hannover
Peter Otte      10       Muenchen

$
```

Jede Zeile der Eingabedatei wird beim Sortieren als Menge von Feldern gesehen. Jedes Feld stellt eine Reihenfolge von Zeichen dar, als Begrenzung dienen ein oder mehrere Leerzeichen bzw. Tabulatoren (abhängig von der Version).

Das Sortieren kann nach einem Teil eines Feldes, einem ganzen Feld oder mehreren Feldern einer Zeile erfolgen.

Der Sortierbereich wird durch ein Paar von Schaltern der Form **+f**[**.c**] **-f**[**.c**] (**fskip**) bestimmt. Wird der zweite Schalter weggelassen, so ist der Sortierbereich durch das Zeilenende begrenzt. Mit **+f** wird, gerechnet vom Zeilenanfang, die Anzahl der Felder angegeben, die übersprungen werden soll; mit **c** (**cskip**) die Anzahl der zu überspringenden Zeichen, gerechnet vom Ende des letzten übersprungenen Feldes. Durch **-f** wird das Feld angegeben, an dessen Ende das Sortieren beendet werden soll, durch **-f.c** das letzte zu sortierende Zeichen im Feld **f**.

Soll z.B. in der Datei **aleute** nach dem zweiten Zeichen des Nachnamens sortiert werden, so müssen ein Feld und zwei Zeichen übersprungen werden (→ Abb.6.2).

```
$ sort  +1.2  aleute
```

```
Joshua Baur     788       Sontheim
Lukas Ondracek  112       Hannover
Isidor Wolf     421       Mindelheim
%Vitus Gsell    692       Fuessen
Peter Otte      10        Muenchen
```

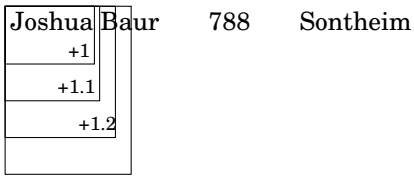


Abbildung 6.2: **sort +1.2 aleute**

Um nach den Ortsnamen zu sortieren, müssen drei Felder übersprungen werden:

$ sort +3 aleute

```
%Vitus Gsell    692       Fuessen
Lukas Ondracek  112       Hannover
Isidor Wolf     421       Mindelheim
Joshua Baur     788       Sontheim
Peter Otte      10        Muenchen

$
```

Da führende Leer- oder Tabulatorzeichen (in der ASCII-Tabelle steht **TAB** mit $(011)_8$ vor dem Leerzeichen $(040)_8$) als Teil der Felder betrachtet werden, erscheint `Muenchen` aufgrund des führenden Leerzeichens an letzter Stelle (→ Abb.6.3).

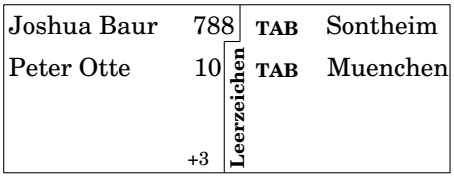


Abbildung 6.3: **sort +3 aleute**

Wird der Schalter **-b** angegeben, so werden führende Leer- oder Tabulatorzeichen ignoriert.

$ sort -b +3 aleute

```
%Vitus Gsell    692       Fuessen
Lukas Ondracek  112       Hannover
Isidor Wolf     421       Mindelheim
Peter Otte      10        Muenchen
Joshua Baur     788       Sontheim
```

```
$
```

Ohne die Angabe des Schalters **-n** (**numeric**) kann das numerische Sortieren fehlerhaft sein,

```
$ sort  +2  aleute

Peter Otte      10       Muenchen
Isidor Wolf     421      Mindelheim
Joshua Baur     788      Sontheim
%Vitus Gsell    692      Fuessen
Lukas Ondracek 112       Hannover

$
```

da führende Leer- oder Tabulatorzeichen (vor 788 stehen mehr Leerzeichen als vor 692) und nicht die zu sortierenden Zahlen, das Ergebnis bestimmen.

```
$ sort  +2n  aleute

Peter Otte      10       Muenchen
Lukas Ondracek 112       Hannover
Isidor Wolf     421      Mindelheim
%Vitus Gsell    692      Fuessen
Joshua Baur     788      Sontheim

$
```

Bei Verwendung des Schalters **-n** unterscheidet **sort** nicht nur ganze Zahlen, sondern auch reelle Zahlen mit Vorzeichen.

Werden gleichzeitig mehrere Felder angegeben, so wird zuerst nach dem ersten, danach nach dem zweiten, usw., sortiert. Stehen z.B. in einer Datei **adr** zeilenweise Namen, Alter und Postleitzahl,

```
$ cat  adr

Peter Ott 45 80805
Jana Baur 5 87750
Lukas Ondracek 50 67000
Vitus Gsell 49 72204
Isidor Wolf 45 87410

$
```

so ergibt das Sortieren nach dem Alter und den Postleitzahlen

```
$ sort  -n  +2  -3  +3  adr
```

```
Jana Baur 5 87750
Peter Ott 45 80805
Isidor Wolf 45 87410
Vitus Gsell 49 72204
Lukas Ondracek 50 67000
```

Das Sortieren nach Feld 3 ergibt bei `Ott` und `Wolf` Gleichheit, das Sortieren nach Feld 4 ordnet dann `Ott` vor `Wolf` ein.

Mehrere Dateien können gemeinsam sortiert (**-m**, **merge**) und das Ergebnis in einer neuen Datei gesichert werden. So werden z.B. durch

```
$ sort -m -o telefone +1 -2 tel-[12]
$ cat telefone

Ingrid Beck                22223
Lothar Haschigk            44834
Irmgard Kalasch            44835
Renate Krauss              26844
Peter Leibner              63436
Friedrich Leicher          47194
      .
      .
      .
      .

$
```

die Dateien **tel-1** und **tel-2** nach dem zweiten Feld sortiert und das Ergebnis in die Datei **telefone** geschrieben (**-o**, **output**).

Übung 19 *Geben Sie das Kommando (Kolonne) an, mit dessen Hilfe die Größe und die Namen der fünf größten Dateien unter* **/usr** *auf den Bildschirm ausgegeben werden.*

6.4 Suche in Textdateien

Der Filter **grep** (**global regular expression printer**)[1] dient zum Durchsuchen von Dateien nach bestimmten Zeichenmustern. Das einfachste Suchmuster ist eine Zeichenkette, kompliziertere Muster können mit Hilfe sogenannter regulärer Ausdrücke (**regular expressions**) gebildet werden.

[1] Modus der letzten Zeile im **vi**. Globale Suche (**g**) nach einem regulären Ausdruck (**r***egular* **e***xpression*) und deren Ausgabe (**p**):
g/re/p

Reguläre Ausdrücke dürfen nicht mit dem sogenannten **globbing** verwechselt werden, das zum Erfassen mehrerer Dateinamen durch die SHELL verwendet wird (→ Kap.4.3). Ihre Verwendung ist nicht nur auf **grep** (**egrep**) beschränkt. Sie werden z.B. auch im **ed** (**ex**, **vi**) (→ S.23), **sed** oder **awk**, mit unterschiedlichem Umfang der Möglichkeiten, eingesetzt.

Um eine Interpretation des Suchmusters durch die SHELL beim Lesen der Eingabezeile zu verhindern, ist es empfehlenswert, reguläre Ausdrücke und Leerzeichen in Hochkommas (`'`*ausdruck*`'`) einzuschließen.

Neben **grep** gibt es zwei weitere Filter mit ähnlicher Funktion: **egrep** (**extended grep**) und **fgrep** (**fixed character grep**). Die drei Kommandos unterscheiden sich in der Komplexität der erlaubten regulären Ausdrücke. Während **fgrep** überhaupt keine regulären Ausdrücke zuläßt, können bei **egrep** komplexere Suchmuster als bei **grep** gebildet werden.

Da **grep** ein guter Kompromiß zwischen der Schnelligkeit von **fgrep** und der Versatilität von **egrep** ist, wird **grep** im folgenden genauer behandelt. Insbesondere wird dabei auf die Bildung von Suchmustern mittels regulärer Ausdrücke Wert gelegt.

Als Beispielsdateien werden

```
$ more *

::::::::::::::
frauen
::::::::::::::
Beate Hecht      777           Ansbach
Maria Diepold    190           Hamburg
Jana Baur        472           Mindelheim
Iva Kraus        101           Krumbach

frauen: END (next file: maenner)<return>

Jan Maly         421           Freiburg
Peter Ott        567           Muenchen
Franz Hecht      421           Augsburg
Joshua Baur      564           Sontheim
Lukas Ondracek   692           Hannover
Vitus Gsell      112           Fuessen
Isidor Wolf      775           Mindelheim
maenner: END

maenner: END<return>
$
```

verwendet.

Bei der Angabe einer Zeichenkette als Suchmuster sucht **grep** nach allen Zeilen, in denen das Suchmuster vorkommt und gibt diese dann auf die Standardausgabe aus.

```
$ grep  Jan  maenner

Jan Maly         421           Freiburg
```

```
$
```

Werden gleichzeitig mehrere Dateinamen angegeben, so werden zusätzlich die Namen der Dateien, in denen das Suchmuster gefunden wurde, ausgegeben.

```
$ grep  Jan  *

frauen:Jana Baur        472             Mindelheim
maenner:Jan Maly         421            Freiburg

$
```

Kommen Leerzeichen im Suchmuster vor, so müssen sie in Hochkommas (oder Anführungszeichen) gesetzt werden.

```
$ grep  'Jan Maly'  maenner

Jan Maly         421            Freiburg

$
```

Mit dem Schalter **-v** (→ Tab.6.5) gibt **grep** nur diejenigen Zeilen an, die das Suchmuster **nicht** enthalten.

```
$ grep  -v  a  *

maenner:Peter Ott        567            Muenchen
maenner:Vitus Gsell      112            Fuessen
maenner:Isidor Wolf      775            Mindelheim

$
```

Die meisten Sonderzeichen zur Angabe des Suchmusters stimmen mit denen des **vi** (→ Tab.3.10) überein. So wird z.B. durch ^ der Zeilenanfang gekennzeichnet.

^

```
$ grep  '^J'  *

frauen:Jana Baur        472             Mindelheim
maenner:Jan Maly         421            Freiburg
maenner:Joshua Baur      564            Sontheim

$
```

$ Durch $ wird das Zeilenende angegeben.

```
$ grep  'rg$'  *
```

```
frauen:Maria Diepold    190             Hamburg
maenner:Jan Maly         421            Freiburg
maenner:Franz Hecht      421            Augsburg

$
```

Werden die Zeichen ^ und $ an anderer Stelle als am Anfang bzw. Ende des Suchmusters verwendet, so verlieren sie ihre besondere Bedeutung.

Ein weiteres Sonderzeichen ist der **.** (Punkt). Durch den Punkt wird im Suchmuster **ein** beliebiges Zeichen repräsentiert:

.

```
$ grep '.6.' maenner

Peter Ott        567            Muenchen
Joshua Baur      564            Sontheim
Lukas Ondracek   692            Hannover

$
```

Da das Leerzeichen auch ein „beliebiges“ Zeichen ist, wurde durch **grep** auch die Zeile mit der Zeichenkette „ **69**“ gefunden.

Die Sonderzeichen können in den Suchmustern auch kombiniert werden.

Übung 20 *Geben Sie eine Pipe zum Auffinden aller Unterverzeichnisse im aktuellen Arbeitsverzeichnis, die andere Benutzer als ihr Arbeitsverzeichnis einstellen können, an.*

Durch [, ^, - und] können Mengen von Suchmustern definiert werden. Durch [und] kann eine Zeichenmenge definiert werden, aus der jeweils **ein** Zeichen zur Bildung des Suchmusters Verwendung findet.

[]

```
$ grep 'M' frauen

Maria Diepold    190            Hamburg
Jana Baur        472            Mindelheim
Iva Kraus        101            Muenchen

$ grep 'M[iu]' frauen

Jana Baur        472            Mindelheim
Iva Kraus        101            Muenchen
```

```
$
```

Mit - kann ein Zeichenbereich angegeben werden. So können z.B. zum Suchen einer Dezimalzahl [**0-9**] und zum Suchen von Zeilen, in denen Großbuchstaben vorkommen [**A-Z**], angegeben werden. [-]

```
$ grep  '[4-6][5-9][2-4]'  *

frauen:Jana Baur        472             Mindelheim
maenner:Joshua Baur      564            Sontheim
maenner:Lukas Ondracek   692            Hannover

$
```

Bei der Angabe von [**A-z**] ist zu beachten, daß in der ASCII-Tabelle in diesem Bereich nicht nur Buchstaben vorkommen.

[^] Das Sonderzeichen ^ bedeutet innerhalb der eckigen Klammern den Ausschluß der folgenden Zeichen aus dem Suchmuster.

Aus den Dateien **frauen** und **maenner** erhält man durch

```
$ grep  '^[^JVPEI]'  *

frauen:Beate Hecht      777             Ansbach
frauen:Maria Diepold    190             Hamburg
maenner:Franz Hecht      421            Augsburg
maenner:Lukas Ondracek   692            Hannover

$
```

alle Zeilen, die am Zeilenbeginn nicht mit **J**, **V**, **P**, **E** oder **I** beginnen.

* Durch * wird das unmittelbar davor stehende Zeichen bzw. der reguläre Ausdruck, beliebig oft (0,1,...) wiederholt.

```
$ grep  '77*'  frauen

Beate Hecht      777            Ansbach
Jana Baur        472            Mindelheim

$
```

\ Mit \ wird die spezielle Bedeutung eines Zeichens aufgehoben.

```
$ ls | grep  '\.'
```

```
cesnet.ps
hot-convert.sh
smilies.ps
soubory.txt

$
```

In der nachfolgenden Tabelle 6.4 sind die im Zusammenhang mit **grep** erlaubten regulären Ausdrücke zusammengefaßt.

Symbol	Bedeutung beim Suchen
c	Zeichen *c*
c	hebt spezielle Bedeutung des Zeichens *c* auf
^	Zeilenanfang
$	Zeilenende
.	ein beliebiges Zeichen außer NEWLINE
[*menge*]	ein beliebiges Zeichen aus *menge*
[^*menge*]	ein beliebiges Zeichen, das nicht in *menge* vorkommt
*r**	beliebige Anzahl (0,1,...) von Wiederholungen des regulären Ausdrucks *r*
r1r2	*r1* gefolgt von *r2*
\(*menge*\)	durchnumerierter regulärer Ausdruck
n	entspricht durchnumeriertem regulären Ausdruck *n*

Tabelle 6.4: Reguläre Ausdrücke des Filters **grep**

Neben dem zuvor erwähnten Schalter **-v**, gibt es noch eine ganze Reihe weiterer (→ Tab.6.5).

Der Schalter **-n** gibt die Zeilennummern der Eingabedateien an, in denen das Suchmuster gefunden wurde.

```
$ grep -n 'Ha' *

frauen:2:Maria Diepold    190          Hamburg
maenner:5:Lukas Ondracek   692         Hannover

$
```

Durch **-l** werden lediglich die Namen der Dateien auf die Standardausgabe ausgegeben, in denen das Suchmuster gefunden wurde.

```
$ grep -l M *

maenner
```

```
$
```

Der Schalter **-c** zählt die Anzahl der Zeilen in den Eingabedateien, in denen das Suchmuster gefunden wurde.

```
$ grep -c '[PT]'

frauen:0
maenner:1

$
```

Schalter	Mnemonik	Bedeutung
-b	block	gibt die Nummer des Datenblocks auf der Festplatte an, in dem die Zeile gefunden wurde
-c	count	gibt die Anzahl der Zeilen aus, in denen das Suchmuster gefunden wurde
-i	ignore	ignoriert Unterschiede zwischen Klein- und Großbuchstaben
-l	list	gibt nur die Dateinamen aus, in denen das Suchmuster gefunden wurde
-n	number	gibt die Zeilennummern aus, in der das Suchmuster gefunden wurde
-s	silent	keine Ausgabe, es wird nur der Rückkehrkode **0** (true), falls das Suchmuster gefunden wurde, **1** (false), falls es nicht gefunden wurde und **2** (false), falls auf die Datei nicht zugegriffen werden konnte oder ein Fehler aufgetreten ist, zurückgegeben
-v	reverse	gibt diejenigen Zeilen aus, die nicht das Suchmuster enthalten

Tabelle 6.5: Schalter des Kommandos **grep**

Sonderzeichen müssen nur außerhalb von eckigen Klammern durch \ ihrer besonderen Bedeutung enthoben werden.

```
$ cat  geld

Einnahmen:

$       550.-
$        77.-

$ grep  '[$]'  geld

$       550.-
$        77.-

$ grep  '\$'  geld

$       550.-
$        77.-

$ grep  '$'  geld

Einnahmen:

$       550.-
$        77.-

$
```

Im letzten Beispiel steht `$` für das Zeilenende, daher werden alle Zeilen ausgegeben.

Übung 21 *Geben Sie eine Pipe an, mit deren Hilfe alle mit einem Punkt anfangenden Dateien, mit Ausnahme von* **.** *und* **..***, in eine Datei* **punkt** *geschrieben werden.*

Übung 22 *Geben Sie eine Pipe an, die alle UNIX-Kommandos aus* **/bin** *ausgibt, die zwei Zeichen lang sind.*

6.5 Vergleich von Dateien

Textdateien können mit dem Filter **diff** verglichen werden. Ohne die Angabe eines Schalters erzeugt **diff** aus dem Vergleich Instruktionen, die angeben, wie aus der einen Datei die andere gewonnen werden kann.

Die Instruktionen können

a - add (hinzufügen),
d - delete (löschen),
c - change (ändern, ersetzen),

lauten. Nach jeder solcher Zeile folgen die Zeilen, die hinzugefügt, gelöscht oder geändert werden müssen. Das Symbol < kennzeichnet Zeilen aus der ersten, das Symbol > aus der zweiten Datei.

Die Funktion von **diff** wird an den Dateien **maenner1** und **maenner2** gezeigt.

```
$ cat  maenner1

Jan Maly          421            Freiburg
Peter Ott         567            Muenchen
Franz Hecht       421            Augsburg
Joshua Baur       564            Sontheim
Lukas Ondracek    692            Hannover
Vitus Gsell       112            Fuessen
Isidor Wolf       775            Mindelheim

$ cat  maenner2

Jan Maly          421            Freiburg
Peter Ott         567            Muenchen
Joshua Baur       564            Sontheim
Lukas Oldracek    692            Hannover
Vitus Gsell       112            Fuessen
Isidor Wolf       775            Mindelheim
Jiri Vidim        654            Prag

$
```

Die Unterschiede zwischen den Dateien sind:

```
$ diff  maenner1  maenner2

3d2
< Franz Hecht       421          Augsburg
5c4
< Lukas Ondracek    692          Hannover
---
> Lukas Oldracek    692          Hannover
7a7
> Jiri Vidim        654          Prag

$
```

Der Vergleich erfolgt zeilenweise. Die ausgegebenen Zahlen beziehen sich auf die Zeilennummern. Das Programm **diff** geht davon aus, daß durch das Editieren der ersten Datei (**maenner1**) die zweite Datei (**maenner2**) entstehen soll. Die Zahlen links von

den Anweisungen beziehen sich auf die erste Datei, die rechts davon auf die zweite. Um aus **maenner1** die Datei **maenner2** zu machen, müssen daher nur die Zahlen links von den Anweisungen beachtet, die rechts davon können ignoriert werden.

Die Ausgabe von **diff** ist wie folgt zu lesen:

1. Lösche in **maenner1** Zeile 3 (die Zeile in der „Franz Hecht ...“ steht).
2. Ändere in **maenner1** Zeile 5 von „Lukas Ondracek ...“ nach „Lukas Oldracek ...“.
3. Füge an Zeile 7 in **maenner1** „Jiri Vidim ...“ aus **maenner2** an.

Wird der Schalter **-e** angegeben, so gibt **diff** die **ed**-Befehle zur Änderung der ersten in die zweite Datei auf die Standardausgabe aus.

```
$ diff -e maenner1 maenner2 >aenderungen
$ cat aenderungen

7a
Jiri Vidim       654            Prag
.
5c
Lukas Oldracek   692            Hannover
.
3d
w

$ ed maenner1 <aenderungen
$ diff maenner1 maenner2
$
```

In manchen Versionen von **diff** werden durch den Schalter **-e** nur die notwendigen Änderungen ausgegeben. Sollen diese als Eingabe von **ed** verwendet werden, so müssen noch **w** (**write**) und gegebenenfalls **q** (**quit**, je nach **ed**-Version) angefügt werden.

Es ist zu beachten, daß **diff** beliebige Unterschiede in den verglichenen Zeilen findet, auch z.B. ein angefügtes Leerzeichen am Zeilenende. Sollen die Dateien ohne Rücksicht auf die Anzahl von Leer- oder Tabulatorzeichen verglichen werden, so muß der Schalter **-b** verwendet werden.

6.5.1 Vergleich von Verzeichnissen

Ebenso wie Dateien können Verzeichnisse miteinander verglichen werden.

```
$ diff verz verzcp
```

```
diff verz/maenner1 verzcp/maenner1
1c1
< Jan Maly          421          Freiburg
---
> Jan Maly          421          Freiburg
Only in verz: uverz

$
```

Das Programm meldet sowohl Unterschiede in den Inhalten der Verzeichnisse als auch in den dort abgelegten gleichnamigen Dateien.

Um aus der Datei **/verz/maenner1** die Datei **/verzcp/maenner1** zu erhalten, muß offensichtlich die erste Zeile in **/verz/maenner1** geändert werden. Bei oberflächlicher Betrachtung kann man aber keinen Unterschied zwischen den beiden Zeilen feststellen (bei genauerer Betrachtung stellt man jedoch fest, daß in **/verz/maenner1** am Zeilenende ein Leerzeichen mehr als in **/verzcp/maenner1** steht!).

Wird der Schalter **-b** angegeben, so ignoriert **diff** die unterschiedliche Anzahl von Leerzeichen und stellt zwischen den Dateien Gleichheit fest.

$ diff -b verz verzcp

```
diff -b verz/maenner1 verzcp/maenner1
Only in verz: uverz

$
```

Jetzt wird nur noch der Unterschied im Verzeichnisinhalt gemeldet (ein **uverz** gibt es nur in **verz**).

Drei Textdateien können mit dem Kommando **diff3** verglichen werden.

6.5.2 Vergleich von beliebigen Dateien

Der Filter **diff** kann nur auf Textdateien angewendet werden. Möchte man beliebige Dateien miteinander vergleichen, so kann dafür **cmp** (**compare**) verwendet werden.

$ diff maenner1 maenner2

```
3d2
< Franz Hecht       421          Augsburg
5c4
< Lukas Ondracek    692          Hannover
---
> Lukas Oldracek    692          Hannover
7a7
> Jiri Vidim        654          Prag
```

```
$ cmp   maenner1   maenner2

maenner1 maenner2 differ: char 63, line 3

$
```

Das Programm stellt zunächst fest, ob beide Eingabedateien gleich lang sind. Ist dies der Fall, so werden beide Dateien Zeichen für Zeichen verglichen. Sind beide gleich, so erfolgt keine Meldung und das Programm wird mit dem Rückkehrkode **0** (true) beendet. Sind sie unterschiedlich, erfolgt die Ausgabe der Stelle, an der die erste Differenz aufgetreten ist und der Rückkehrkode des Programms ist **1** (false).

Mit dem Schalter **-l** (**list**) erhält man die Ausgabe aller Unterschiede in oktaler Darstellung.

```
$ cmp   -l   maenner1   maenner2

    63 106 112
    64 162 157
    65 141 163
         .
         .
         .

$
```

In der ersten Spalte wird angegeben, um das wievielte Zeichen es sich handelt, in der zweiten steht der Wert des Zeichens in der ersten $(106)_8 =$ **F***ranz*, in der dritten Spalte der Wert in der zweiten ($(112)_8 =$ **J***oshua*) Datei.

6.6 Übersetzung von Zeichen in Dateien

Mit dem Filter **tr** (**translate**) können die Zeichen in einer Datei verändert, ersetzt oder gelöscht werden. Er ist einer der Filter, die nur von der Standardeingabe lesen können. Die Übersetzung wird durch zwei Zeichenketten bestimmt. Wird in der Standardeingabe eines der Zeichen aus der ersten Zeichenkette gefunden, so wird es durch ein an entsprechender Stelle in der zweiten Zeichenkette stehendes ersetzt (z.B. das dritte Zeichen aus der ersten Zeichenkette durch das dritte aus der zweiten).

```
$ tr   aeiou   uoiea   <maenner1

Jun Muly          421            Froibarg
Potor Ott         567            Maonchon
Frunz Hocht       421            Aagsbarg
Jeshau Buar       564            Senthoim
```

```
Lakus Ondrucok   692          Hunnevor
Vitas Gsoll      112          Faosson
Isider Welf      775          Mindolhoim

$
```

Enthält dabei die zweite Zeichenkette weniger Zeichen als die erste, so werden eben nur die korrespondierenden Zeichen ausgetauscht (das Ergebnis hängt von der Version des Filters ab).

```
$ tr  aeiou  u  <maenner1

Jun Muly         421          Freiburg
Peter Ott        567          Muenchen
Frunz Hecht      421          Augsburg
Joshuu Buur      564          Sontheim
Lukus Ondrucek   692          Hunnover
Vitus Gsell      112          Fuessen
Isidor Wolf      775          Mindelheim
```

Zur Definition der Zeichenketten können auch einfache reguläre Ausdrücke verwendet werden: [*a*-*z*] [*a***n*]

[*a*-*z*]	Zeichen zwischen *a* und *z*, einschließlich *a* und *z*
[*a***n*]	*n*-fache Wiederholung des Zeichens *a*, z.B. [**a*3**]=[**aaa**]; ist die erste Ziffer von *n* eine Null, so wird die Zahl als Oktalzahl interpretiert

Alle Kleinbuchstaben können z.B. durch

```
$ tr '[a-z]' '[x*]' <maenner1

Jxx Mxxx          421             Fxxxxxxx
Pxxxx Oxx         567             Mxxxxxxx
Fxxxx Hxxxx       421             Axxxxxxx
Jxxxxx Bxxx       564             Sxxxxxxx
Lxxxx Oxxxxxxx    692             Hxxxxxxx
Vxxxx Gxxxx       112             Fxxxxxx
Ixxxxx Wxxx       775             Mxxxxxxxxx

$
```

zu x und durch

```
$ tr '[a-z]' '[A-Z]' <maenner1

JAN MALY          421             FREIBURG
PETER OTT         567             MUENCHEN
FRANZ HECHT       421             AUGSBURG
JOSHUA BAUR       564             SONTHEIM
LUKAS ONDRACEK    692             HANNOVER
VITUS GSELL       112             FUESSEN
ISIDOR WOLF       775             MINDELHEIM

$
```

zu Großbuchstaben gemacht werden. Sollen beim Übersetzen Wiederholungen unterbleiben, so ist der Schalter **-s** (**squeeze**) zu verwenden (→ Tab.6.6).

```
$ tr -s '[a-z]' '[x*]' <maenner1

Jx Mx          421              Fx
Px Ox         567               Mx
Fx Hx       421           Ax
Jx Bx       564           Sx
Lx Ox    692              Hx
Vx Gx       112           Fx
Ix Wx       775           Mx
```

Schalter	Mnemonik	Bedeutung
-d	delete	alle Zeichen der Standardeingabe, die in der ersten Zeichenkette enthalten sind, werden gelöscht; falls **-s** nicht angegeben ist, wird die zweite Zeichenkette ignoriert, ansonsten dient die erste zum Löschen und die zweite zur Angabe derjenigen Zeichen, deren mehrfache Wiederholung auf ein Zeichen reduziert werden soll
-c	complement	aus den Zeichen in der ersten Zeichenkette wird das Komplement zur Menge der ASCII-Zeichen gebildet und anschließend das n-te Zeichen aus der Komplementmenge durch das n-te aus der zweiten Zeichenkette ersetzt; zusammen mit **-d** werden die Zeichen aus der Komplementmenge aus der Standardeingabe gelöscht, eine zweite Zeichenkette wird dabei ignoriert
-s	squeeze	mehrfache Wiederholung von Zeichen, die in der zweiten Zeichenkette spezifiziert sind, werden auf ein Zeichen reduziert

Tabelle 6.6: Schalter des Kommandos **tr**

Mit dem Schalter **-c** (**complement**) wird die komplementäre Menge zur ersten angegebenen Zeichenmenge in bezug auf die ASCII-Tabelle, gebildet. Mit **-d** (**delete**) werden alle Zeichen, die in der ersten Zeichenkette stehen, aus der Standardeingabe gelöscht.

```
$ tr -cd '[0-9]' <maenner1

421567421564692112775$

$
```

Auf die Standardausgabe werden wegen **[0-9]** nur die Ziffern weitergegeben, alle anderen Zeichen werden gelöscht. Davon sind auch alle Leer- und Tabulatorzeichen sowie NEWLINE betroffen.

Da die Zeichen in den Zeichenmengen auch durch ihren oktalen Wert angegeben werden können, kann NEWLINE vom Löschen ausgeschlossen werden:

```
$ tr -cd '\012[0-9]' <maenner1

421
567
421
564
692
112
775
```

Um alle Tabulatorzeichen in **maenner1** in Leerzeichen zu verwandeln, kann z.B.

```
$ tr '\011' ' ' <maenner1

Jan Maly          421  Freiburg
Peter Ott         567  Muenchen
Franz Hecht       421  Augsburg
Joshua Baur       564  Sontheim
Lukas Ondracek    692  Hannover
Vitus Gsell       112  Fuessen
Isidor Wolf       775  Mindelheim

$
```

eingegeben werden.

Übung 23 *Geben Sie ein Kommando an, durch welches Sie alle Worte in einer Textdatei* **dat** *auf eigene Zeilen bekommen, ohne daß dazwischen Leerzeilen auftreten würden. (Die Worte können in der Datei durch Komma, Punkt, Doppelpunkt, Strichpunkt, Ausrufezeichen und Fragezeichen sowie durch Leer- oder Tabulatorzeichen voneinander getrennt sein.)*

6.7 Kontinuierliches Editieren einer Datei

Gegenüber dem interaktiven zeilenorientierten Editieren einer Datei durch den **ed** bzw. **ex**, erfolgt das kontinuierliche Editieren durch den **sed** (**stream editor**) nicht interaktiv. Der Filter liest von der Standardeingabe eine Zeile nach der anderen (daher der Name), führt mit ihr gegebenenfalls eine Reihe von Transformationen durch und gibt sie dann auf die Standardausgabe aus.

Diese Vorgehensweise ist insbesondere bei großen Dateien oder wenn komplizierte Änderungen vorgenommen werden sollen, von Vorteil.

Dem **sed** kann beim Aufruf eine Datei mit den Editierkommandos übergeben werden. Stehen diese z.B. in der Datei **kommandos**, so sieht der Aufruf wie folgt aus:

```
$ sed -f kommandos datalt >datneu
```

Der Schalter **-f** gibt an, daß die nachfolgende Datei die Editierkommandos enthält. Eine Variante, welche die Editierkommandos von der Standardeingabe liest, lautet:

```
$ sed programm datalt >datneu
```

Bei *programm* handelt es sich um eine Befehlsfolge, die direkt in der Eingabezeile angegeben wird.

$ sed '/^$/d' datein >dataus

Dieses Kommando löscht z.B. alle leeren Zeilen in der Datei **datein** und schreibt das Ergebnis nach **dataus**.

Die allgemeine Form eines Editierkommandos ist

[*adresse1*[**,***adresse2*]]*operation*[*parameter*]

Die Adressen sind optional. Falls keine Adresse angegeben wird, werden alle Zeilen der Eingabedatei bearbeitet. Ansonsten suchen die Adressen diejenigen Zeilen heraus, auf welche die gegebene *operation* ausgeführt werden soll. Die Anzahl und Art der *parameter* hängt von der jeweiligen Operation ab. Für **w** (**write**) ist es z.B. der Name der Datei, in die geschrieben werden soll, für **q** (**quit**) gibt es dagegen keinen entsprechenden Parameter.

Der **sed** arbeitet wie folgt:

1. Es wird eine Zeile der Eingabedatei gelesen.
2. Das erste Kommando aus der Datei **kommandos** wird gelesen. Falls die Adresse des Kommandos die Zeile der Eingabedatei auswählt, wird auf sie die *operation* angewendet.
3. Aus der Datei **kommandos** wird das nächste Kommando gelesen. Paßt die Adresse des Kommandos zur aktuellen Zeile aus der Eingabedatei, so wird die *operation* auf sie ausgeführt.
4. Der Schritt 3 wird so lange wiederholt, so lang weitere Befehle in der Datei **kommandos** stehen.
5. Falls in der Eingabedatei eine weitere Zeile steht, wird mit Schitt 1 fortgefahren, falls nicht, so wird der **sed** beendet.

Im einfachsten Fall ist die Adresse eine Nummer, welche die Anzahl der Zeilen, gerechnet von der ersten Zeile der ersten Eingabedatei (dem **sed** können gleichzeitig mehrere Eingabedateien übergeben werden), angibt. Ein spezieller Fall ist die Adresse **$**, sie gibt die letzte Zeile der letzten Eingabedatei an.

Als Adresse kann auch ein regulärer Ausdruck (eingeschlossen in Schrägstriche) angegeben werden. Durch diesen werden diejenigen Zeilen der Eingabedatei ausgewählt, welche die spezifizierte Zeichenkette aus dem regulären Ausdruck enthalten.

Falls mit der Operation keine Adresse verbunden ist, wird sie auf jede Zeile angewendet. Falls mit ihr eine Adresse verbunden ist, wird sie auf jede Zeile angewendet, die durch die Adresse ausgewählt wird. Falls zwei Adressen angegeben sind, wird die Operation auf Zeilengruppen angewendet. Dabei wählt die erste Adresse die erste Zeile und die zweite die letzte der ersten Gruppe aus. Nach der Bearbeitung der letzten Zeile der ersten Gruppe wird die nächste Gruppe gesucht und der Vorgang wiederholt sich.

In den nachfolgenden Beispielen soll die Datei **maenner** bearbeitet werden.

$ cat maenner

```
Jan Maly                421
Peter Ott               567
Franz Hecht             421
Joshua Baur             564
Lukas Ondracek          692
Vitus Gsell             112
Isidor Wolf             775

$
```

In **ed.com** stehen die Kommandos zum Editieren der Datei **maenner**.

$ cat ed.com

```
1i\
NAME                    PUNKTE:
/Joshua/s//Jeremias/
/Lukas/d
$a\
Egon Schiele            111

$
```

Die Kommandos (→ Tab.6.7) beschreiben folgende Transformationen:

- Füge vor die erste Zeile `NAME PUNKTE:` ein.
- Ersetze in allen Zeilen, die `Joshua` enthalten, `Joshua` durch `Jeremias` (→ S.22).
- Lösche alle Zeilen, die `Lukas` enthalten.
- Füge hinter die letzte Zeile `Egon Schiele 111` an.

Bei **a** (**add**) sowie **i** (**insert**) müssen alle Zeilen, außer der letzten, mit \ abgeschlossen werden. Steht vor **a** oder **i** keine Adresse, so wird der angegebene Text vor (bei **i**) bzw. nach (bei **a**) jeder Zeile eingefügt.

$ sed -f ed.com maenner

```
NAME                    PUNKTE:
Jan Maly                421
Peter Ott               567
Franz Hecht             421
Jeremias Baur           564
Vitus Gsell             112
Isidor Wolf             775
Egon Schiele            111
```

Mit der Kommandodatei **edit.com**,

```
$ cat  edit.com

s/^/|/
s/[0-9]/|&/
s/$/|/
1i\
\ ___________________________
a\
|_________________________|___|

$
```

erhält man folgende Ausgabe:

```
$ sed  -f  edit.com  maenner

 ___________________________
|Jan Maly                 |421|
|_________________________|___|
|Peter Ott                |567|
|_________________________|___|
|Franz Hecht              |421|
|_________________________|___|
|Joshua Baur              |564|
|_________________________|___|
|Lukas Ondracek           |692|
|_________________________|___|
|Vitus Gsell              |112|
|_________________________|___|
|Isidor Wolf              |775|
|_________________________|___|

$
```

Die ersten drei Befehle zeichnen die vertikalen Striche, **i** fügt Text vor der ersten Zeile ein, **a** ergänzt den Text nach jeder Zeile. Das Leerzeichen bei **i** wird nur dann als Teil des Textes aufgefaßt, wenn ein \ davor steht.

6.7.1 Schalter des **sed**-Filters

Wie bereits erwähnt, können Editierkommandos nicht nur aus einer speziellen Datei, sondern auch direkt an den **sed** übergeben werden. Handelt es sich dabei um mehr als eine Anweisung (s.o.), so muß vor den Befehlen jeweils der Schalter **-e** stehen.

```
$ sed  -e  /^J/d  -e  s/Franz/Friedrich/  maenner

Peter Ott              567
Friedrich Hecht        421
Lukas Ondracek         692
Vitus Gsell            112
Isidor Wolf            775

$
```

Die Schalter **-e** und **-f** können auch gemeinsam verwendet werden.

```
$ sed  -e  /^J/d  -e  s/Franz/Friedrich/  -f  edit.com  maenner
```

```
|Peter Ott              |567|
|_______________________|___|
|Friedrich Hecht        |421|
|_______________________|___|
|Lukas Ondracek         |692|
|_______________________|___|
|Vitus Gsell            |112|
|_______________________|___|
|Isidor Wolf            |775|
|_______________________|___|

$
```

Die Eingaben an den **sed** werden von links nach rechts abgearbeitet. Da mit **-e /^J/d** die erste Zeile gelöscht wird, bevor die Kommandofolge aus **edit.com** auf die Datei angewendet wird, gibt es für **1i** keine passende Adresse mehr. Ändert man die Anweisung in **2i** so entsteht wieder die gewünschte Tabelle:

$ sed -e /^J/d -e s/Franz/Friedrich/ -f edit_mod.com maenner

```
 _________________________
|Peter Ott              |567|
|_______________________|___|
|Friedrich Hecht        |421|
|_______________________|___|
|Lukas Ondracek         |692|
|_______________________|___|
|Vitus Gsell            |112|
|_______________________|___|
|Isidor Wolf            |775|
|_______________________|___|

$
```

Die Operation **p** (**print**) läßt jede Zeile auf die Standardausgabe durch und schickt anschließend noch eine Kopie der ausgesuchten Zeile hinterher.

$ sed /2$/p maenner

```
Jan Maly              421
Peter Ott             567
Franz Hecht           421
Joshua Baur           564
Lukas Ondracek        692
Lukas Ondracek        692
Vitus Gsell           112
Vitus Gsell           112
Isidor Wolf           775
```

```
$
```

Sollen nur die für **p** ausgesuchten Zeilen auf die Standardausgabe weitergegeben werden, so muß man zusätzlich den Schalter **-n** verwenden.

```
$ sed -n /2$/p maenner

Lukas Ondracek          692
Vitus Gsell             112

$
```

Diese Konstruktion arbeitet wie das Kommando **grep**:

```
$ grep '2$' maenner

Lukas Ondracek          692
Vitus Gsell             112

$
```

In Verbindung mit einer absoluten Adresse und der Operation **q** (**quit**), arbeitet der **sed** wie das Kommando **head**.

```
$ sed 4q maenner

Jan Maly                421
Peter Ott               567
Franz Hecht             421
Joshua Baur             564

$
```

Es werden die ersten 4 Zeilen ausgegeben und danach der **sed** beendet. Da **q** nur die Angabe einer Adresse erlaubt, muß zur Ausgabe eines Zeilenbereichs eine andere Konstruktion gewählt werden.

```
$ sed -n 3,4p maenner

Franz Hecht             421
Joshua Baur             564

$
```

Ein Zeilenintervall kann auch mit Hilfe eines regulären Ausdrucks angegeben werden.

```
$ sed -n '/Lu/,$p' maenner
```

```
Lukas Ondracek          692
Vitus Gsell             112
Isidor Wolf             775

$
```

Durch den angegebenen Adreßbereich werden die Zeilen, beginnend mit der Zeile, welche die Zeichenkette **Lu** enthält, bis zum Ende der Datei (Adresse $), ausgegeben.

6.7.2 Anwendung des **sed** auf mehrere Dateien

Dem **sed** können gleichzeitig mehrere Eingabedateien übergeben werden. Die Zeilen der Eingabedateien werden dabei gemeinsam durchnumeriert.

```
$ cat  frauen

Jana Mala               777
Maria Rot               190
Beate Hecht             101
Paula Roth              456

$ sed  -n  7,8p  maenner  frauen

Isidor Wolf             775
Jana Mala               777

$
```

Eine Pipe ermöglicht z.B. die Anwendung bestimmter Operationen auf ausgewählte Dateien.

```
$ ls | sed  -n  /x/p

xx
xy
xz

$ ls | sed  -n  /x/p | sed  's/^/rm  /'

rm xx
rm xy
rm xz
```

```
$ ls | sed -n /x/p | sed 's/^/rm /' | sh
$
```

Durch die letzte Pipe in der Kolonne werden die Kommandos zur Ausführung an eine SHELL weitergeleitet.

Die Verwendung regulärer Ausdrücke zeigt das folgende Beispiel:

```
$ sed 's/[^a]*//g' maenner | paste -s -d' ' - \
| tr -d '\040\012' | tr a '\012' | wc -l

        7

$
```

Durch den **sed** werden zuerst alle Zeichen bis auf die **a**s gelöscht, dann werden die **a**s in einer Zeile zusammengefaßt, alle Leerzeichen und NEWLINE gelöscht, die **a**s zu NEWLINEs gemacht und die Zeilen gezählt. Das Ergebnis ist die Anzahl aller **a**s in der Datei **maenner**.

Mit der Operation **y** erhält man die Funktion von **tr**. Die Parameter der Operation sind zwei gleich lange Zeichenketten, aus denen die erste die Zeichen enthält, die durch die aus der zweiten ersetzt werden sollen.

```
$ sed y/aeiou/AEIOU/ maenner | head -3

JAn MAly                421
PEtEr Ott               567
FrAnz HEcht             421

$
```

Zeichenintervalle werden durch **y** falsch interpretiert:

```
$ sed y/[a-z]/[A-Z]/ maenner | head -3

JAn MAly                421
Peter Ott               567
FrAnZ Hecht             421

$
```

Die Operation **r** (**read**) liest eine angegebene Datei hinter die ausgewählte Zeile. Steht in **leerz** z.B. eine Leerzeile,

```
$ cat  leerz

$
```

so kann diese durch

```
$ sed  'r  leerz'  maenner | head  -4
Jan Maly                421

Peter Ott               567


$
```

zu jeder Zeile der Datei **maenner** hinzugefügt werden.

Die Operation **s** (**substitute**) kann zusammen mit dem Parameter **w** (**write**) verwendet werden.

```
$ sed  -n  's/Vitus/Xaver/w  xaver'  maenner
$ cat  xaver
Xaver Gsell             112

$
```

Auf ähnliche Art und Weise arbeitet die Operation **w** (**write**).

```
$ sed  -n  -e  '1,5w  teil1'  -e  '6,10w  teil2'  frauen  maenner
$ more  teil*
::::::::::::::
teil1
::::::::::::::
Jana Mala               777
Maria Rot               190
Beate Hecht             101
Paula Roth              456
Jan Maly                421

teil1: END (next file: teil2)<return>

Peter Ott               567
Franz Hecht             421
Joshua Baur             564
Lukas Ondracek          692
Vitus Gsell             112
```

```
teil2: END<return>

$
```

Mit diesem Befehl wurden die Dateien **frauen** und **maenner** auf die Dateien **teil1** und **teil2** aufgeteilt. Diese Vorgehensweise ersetzt folglich das Kommando **split**.

Kommando	**Mnemonik**	**Bedeutung**
a	append	Anfügen einer oder mehrerer Zeilen an die ausgewählte Zeile; falls keine Adresse angegeben ist, werden die Zeilen bei allen Zeilen angefügt; es darf nur eine Adresse angegeben werden; die allgemeine Form sieht wie folgt aus: [*adresse*]**a** **text1** **text2** **.** **textn**
c	change	wie **a** bzw. **i**, mit dem Unterschied, daß die ausgewählten Zeilen so verändert werden, daß sie den neuen Text erhalten

Tabelle 6.7: Kommandos des **sed** (Teil 1)

Kommando	Mnemonik	Bedeutung
d	delete	Löschen der ausgewählten Zeile mit anschließendem Lesen der nächsten Zeile aus der Eingabedatei und Neubeginn der Abarbeitung der Kommandos aus der Datei **kommandos**
i	insert	stimmt mit **a** überein, die Zeilen werden jedoch vor die auswählte Zeile geschrieben
l	list	Ausgabe der Zeilen einschließlich nichtdarstellbarer Zeichen, wie z.B. des Tabulatorzeichens oder des Zeilenendes
p	print	Ausgabe der ausgewählten Zeilen auf die Standardausgabe
q	quit	Beenden des **sed**
r *dateiname*	read	Lesen der Datei *dateiname* und Ausgabe des Inhalts hinter der ausgewählten Zeile; es ist nur eine Adresse erlaubt
s/*alt*/*neu*/*parameter*	substitute	Ersatz der Zeichenkette *alt* durch *neu*; für *alt* kann ein regulärer Ausdruck angegeben werden, der duch die Angabe von **&** in *neu* wieder verwendet werden kann; als *parameter* können **g** globaler Ersatz in der ausgewählten Zeile **p** alle Zeilen, in denen ein Ersatz stattgefunden hat, werden auf die Standardausgabe ausgegeben; bei mehrfachem Ersatz wird die Zeile mehrfach ausgegeben **w** *name* die Ausgabe erfolgt statt auf die Standardausgabe in eine Datei *name*
y/*zeiket1*/*zeiket2*/	–	Ersatz der Zeichen aus *zeiket1* durch Zeichen aus *zeiket2* in gleicher Position
w *dateiname*	write	Schreiben der ausgewählten Zeilen in die Datei *dateiname*
=	–	Ausgabe der Nummer der ausgewählten Eingabezeile
! *kommando*	–	*kommando* wird nur auf Zeilen angewendet, die nicht ausgewählt wurden

Tabelle 6.7: Kommandos des **sed** (Teil 2)

Übung 24 *Geben Sie an, wie mit Hilfe des Kommandos* **sed** *für alle Einträge aus* **/etc/passwd** *folgende Bildschirmausgabe entsteht:* **Kommentar:uid**

7 Prozesse

Heutige Prozessoren unterscheiden mindestens zwei Arbeitsmodi, den privilegierten und den Benutzermodus. Arbeitet der Prozessor im privilegierten Modus, so kann er auch sogenannte privilegierte Befehle ausführen. Diese führen kritische Operationen, wie z.B. die Ein- und Ausgabe oder die Änderung von Zugriffsrechten, durch. Im Benutzermodus laufende Programme können diese Operationen nicht ausführen.

Der Systemkern läuft im privilegierten Modus. Er bedient die technischen Mittel des Rechners und bietet den Programmen eine Schnittstelle zur Übergabe von Systembefehlen an. Die im Benutzermodus laufenden Programme können mittels diesen den Systemkern zur Durchführung von Operationen auffordern, deren Ausführung ihnen selbst nicht gestattet ist. Während der Ausführungszeit steht der Systemkern ausschließlich dem Prozeß zur Verfügung, der seine Hilfe mittels Systembefehl angefordert hat.

Die Ausführung eines Systembefehls aus einem Programm heraus erfolgt in zwei Schritten. In einem ersten Schritt werden die Befehlsargumente in den Registern des Prozessors abgelegt, danach wird durch eine spezielle Anweisung an den Systemkern vom Benutzermodus in den priviligierten Modus umgeschaltet. Der Systemkern überprüft anschließend die übergebenen Argumente und führt dann die angeforderte Operation durch. Die Programme selbst können keine Operationen ausführen, mit denen sie andere Programme oder die Sicherheit der Daten gefährden würden.

Die Systembefehle sind in einer C-Bibliothek (**/lib/libc.a**) zusammengefaßt. Deren Schnittstelle wurde in der Norm POSIX 1003.1 definiert.

Der Benutzer kann mit dem System nur über Programme der Anwendungsschicht (→ Abb.1.1) kommunizieren.

Das zentrale Element, das im Betriebssystem verarbeitet wird, ist der Prozeß. Der Systemkern bietet den Prozessen folgende Dienste an:

- **Prozeßverwaltung.** Bildung, Beendigung und Steuerung von Prozessen.
- **Speicherverwaltung.** Zuteilung des Hauptspeichers. Reicht der benötigte Speicherplatz nicht aus, so erfolgt eine zwischenzeitliche Auslagerung der Daten auf die Festplatte (**swap out/swap in**).
- **Dateisystem.** Hierarchische Anordnung und Verwaltung der Dateien und des Ausgleichsspeichers.
- **Prozeßkommunikation.** Der Systemkern enthält die notwendigen Mechanismen sowohl zur Kommunikation zwischen Prozessen eines Rechners als auch zwischen Prozessen unterschiedlicher zu einem Rechnernetz verbundener Rechner.

Jedes Programm wird durch einen Prozeß ausgeführt. Das Mehrbenutzersystem UNIX teilt die Rechenzeit auf die Prozesse mehrerer Benutzer auf. Da innerhalb kurzer Zeit alle Prozesse bedient werden, entsteht der Eindruck, daß alle gleichzeitig laufen. In Wirklichkeit stellt jedoch der Systemkern Rechenleistung immer nur einem Prozeß und das nur für kurze Zeit, zur Verfügung.

Neben diesen Benutzerprozessen laufen noch die bereits erwähnten (→ S.46) „daemons“, die beim Hochfahren des Systems automatisch gestartet werden.

Ein neuer Prozeß wird vom Systemkern durch den Systembefehl **fork()**[1] (verzweige) erzeugt. Der neue Prozeß

- ist identisch mit dem erzeugenden Prozeß, er unterscheidet sich lediglich durch seine PID (**Process Identification Number**),
- er wird unmittelbar nach **fork()** gestartet und
- als Sohn des erzeugenden Prozesses (Vater) bezeichnet.

Von diesem Augenblick an laufen Vater und Sohn gleichzeitig. Aus dem Rückkehrkode von **fork()** können die Prozesse erkennen, ob sie Vater oder Sohn sind. Ist der Rückkehrkode 0, so handelt es sich um den Sohn, ist es die PID des Sohnes, um den Vater. Ein Vater kann mehrere Söhne haben, jeder Sohn jedoch nur einen Vater.

Synchronisieren lassen sich Prozesse mit dem Systembefehl **wait()**[2]. Als Rückkehrkode liefert **wait()** die PID des beendigten Sohnes.

Beendigen kann sich ein Prozeß durch den Systembefehl **exit(***status***)**[3]. Falls der Vaterprozeß ihn durch **wait** anfordert, so erhält er über die Variable *status* den Rückkehrkode des Sohnes.

Der Sohnprozeß kann sich (seine Text- und Datensegmente) durch den Systemdienst *exec*[4] selbst überschreiben (→ **getty, login**).

Informationen können Prozesse über den Systembefehl **kill**[5] austauschen.

Auf ein ankommendes Signal kann ein Prozeß unterschiedlich reagieren. Er kann das Signal zur Erledigung dem Systemkern überlassen. Dies führt in den meisten Fällen zur Beendigung des Prozesses (daher der Name „kill“). Ferner kann er das Signal ignorieren oder abfangen. Will der Prozeß das Signal abfangen, so muß er zuvor eine Prozedur bestimmen, die nach dem Eintreffen des Signals ausgeführt werden soll. In diesem Fall verhalten sich die Signale wie eine Unterbrechung (**interrupt**). Nach Beendigung

[1] int fork(void)
[2] int wait(int *stat_loc), stat_loc enthält die Ursache der Beendigung und den Rückkehrkode des Sohnes
[3] void exit(int status)
[4] durch die Bibliotheksfunktion execl("/bin/ls", "ls", "-l", NULL) wird z.B. der Inhalt des Arbeitsverzeichnisses ausgegeben
[5] int kill(int pid, int sig), das Signal sig wird dem Prozess mit der PID pid geschickt

der Prozedur wird der Prozeß an der Stelle fortgesetzt, an der er durch das Signal unterbrochen wurde.

Wird der Prozeß durch ein Signal des Systemkerns beendigt (z.B. bei einem fehlerhaften Verhalten des Programms), so legt der Systemkern im Arbeitsverzeichnis eine Datei mit dem Namen **core** ab, die ein Speicherabbild des Prozesses beim Eintreffen des Signals enthält.

7.1 Prozeßhierarchie

Dem bisher gesagten kann man entnehmen, daß Prozesse ebenso wie Dateien einen hierarchischen Aufbau haben (→ Abb.7.1).

Ist der Systemkern geladen und damit das Betriebssystem gestartet, entsteht als erster ein Prozeß namens **swapper**. Er hat die PID 0. Seine Hauptaufgabe ist die Speicherverwaltung (s.o.).

Ganz zu Beginn seines Laufes erzeugt er mittels fork() und execl("/etc/init", "init") den Prozeß **init**, der die PID 1 hat und bereits im Benutzermodus läuft. Dem Prozeß **init** werden durch den Systemkern Parameter übergeben, die bestimmen, ob das System als Einbenutzer- oder Mehrbenutzersystem arbeiten soll. Für ein Einbenutzersystem erzeugt **init** nur wenige weitere Prozesse, die dem Systemverwalter das Anmelden am System ermöglichen. Der Systemverwalter hat dann z.B. die Möglichkeit, Wartungsarbeiten am System durchzuführen.

Im allgemeinen läuft das System im Mehrbenutzermodus, in dem sich mehrere Benutzer interaktiv am System anmelden können, um dann in ihm gleichzeitig zu arbeiten.

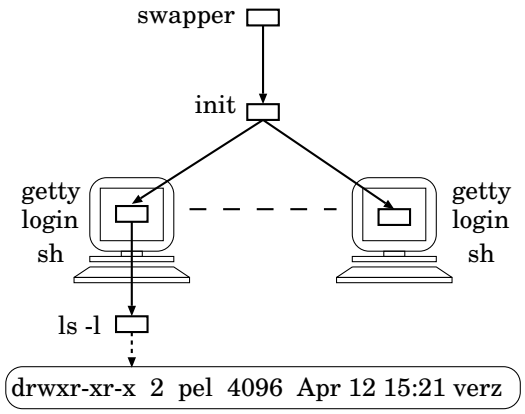


Abbildung 7.1: Hierarchie der Prozesse

Der Prozeß **init** liest als nächstes die Datei **/etc/inittab**, in der festgelegt ist, welche Prozesse gestartet werden sollen. Üblicherweise stehen darin die SHELL-Scripts **/etc/rc** (**runtime commands**), die eine Konsistenzkontrolle des Dateisystems durchführen und die „daemons“ starten.

Danach startet **init** für jedes angeschlossene Terminal den Prozeß **getty**, der auf den Bildschirm die Aufforderung zum Anmelden (`login:`) ausgibt. Gibt der Benutzer seinen Loginnamen ein, so wird **getty** durch den Prozeß **login** überschrieben. Der Loginname wird **login** als Parameter übergeben, wonach **login** zur Eingabe des Paßwortes (`password:`) auffordert. Das eingegebene Paßwort wird dann verschlüsselt und mit dem Eintrag in **/etc/passwd** (evtl. in **/etc/shadow**) verglichen. Falls das Paßwort den Vergleich besteht, wird der Prozeß **login** durch den Kommandointerpreter überschrieben, der in der Datei **/etc/passwd** für den Benutzer angegeben ist. Steht dort z.B. **sh**, so wird der interaktive Kommandointerpreter der Bourne-SHELL gestartet.

Der Prozeß **sh** schreibt dann auf den Bildschirm den Prompt `$` und beginnt Kommandos, die ihm nach der Eingabe von **<return>** übergeben werden, auszuwerten.

Die Interpretation der Eingabe (→ Abb.7.1)

$ ls -l

führt z.B. zum Start eines Prozesses namens **ls** (mittels **fork()**), der als Sohn von **sh** den Verzeichnisinhalt auf den Bildschirm ausgibt.

Während der Zeit, in der **ls** läuft, können keine weiteren Kommandos an die SHELL übergeben werden. Bei Befehlen, deren Ausführung sehr viel Zeit in Anspruch nimmt, ist dies sicher nicht wünschenswert. Es gibt daher die Möglichkeit, Kommandos sozusagen „im Hintergrund“ laufen zu lassen. Dazu muß das Kommando mit dem Sonderzeichen **&** abgeschlossen werden.

$ du

```
2       ./verz/uverz
22      ./verz
8       ./verzcp
52      .

$
```

Mit dem Kommando **du** (**disc usage**) kann der Verbrauch an Speicherplatz ausgehend von einem Verzeichnis rekursiv für alle Dateien ermittelt werden. Das Kommando läuft umso länger, umso mehr Dateien überprüft werden müssen. Möchte man z.B. wissen, wieviel Speicherplatz die einzelnen Dateien der Benutzer unter **/home** einnehmen, so ist es sinnvoll, das Kommando im Hintergrund zu starten:

$ du -k /home >du.out 2>/dev/null &

```
1243
```

$ tail -2 du.out

```
30723   /home/bfs
69298   /home
```

Als Rückmeldung gibt die SHELL die PID des Kommandos aus. Der letzte Eintrag in der Datei **du.out** gibt die Summe des unter **/home** verbrauchten Speicherplatzes in Kilobyte (Schalter **-k**) an.

Leider meldet die Bourne-SHELL weder das Ende des im Hintergrund laufenden Prozesses, noch kann dieser wieder in den Vordergrund geholt und dort fortgesetzt werden.

7.2 Prozeßstatus

Das wichtigste Kommando für die Arbeit mit Prozessen ist **ps** (**process status**). Wird es ohne Schalter verwendet, so gibt es Informationen über die vom Benutzer gestarteten Prozesse aus.

```
$ ps

  PID TTY STAT  TIME COMMAND
   83 pp0 S     0:00 -bash
  129 pp0 R     0:00 ps

$
```

Für jeden Prozeß wird seine PID, das Terminal, von dem er gestartet wurde, die CPU-Zeit, die er während seines Laufs angesammelt hat und der Name des Kommandos, das der Prozeß ausführt, ausgegeben.

An dieser Stelle sei angemerkt, daß die PID des Prozesses, der die aktuelle SHELL ausführt, auch in der vordefinierten SHELL-Variablen **$** hinterlegt ist.

```
$ echo  $$

83

$
```

Die Variable **$** wird zur Erzeugung eindeutiger Namen bei temporären Hilfsdateien verwendet. Diese Hilfsdateien werden im Verzeichnis **/tmp** angelegt. Ein Programm, das dieses Verzeichnis häufig benutzt, ist der Compiler. Während seines Laufes legt er diese Dateien unter **/tmp** an, nach dem Ende seiner Tätigkeit löscht er sie dann wieder. Der gleiche Compiler kann jedoch gleichzeitig von mehreren Benutzern verwendet werden. Um nach der Übersetzung nicht die Hilfsdateien der anderen Benutzer zu löschen, müssen sich die Dateien namentlich unterscheiden.

```
$ cat  /dev/null  >/tmp/sh$$
$ ls  /tmp | grep  sh$$
```

`sh83`

Zurück zum Kommando **ps**. Leider gibt es auch hier, wie bei den Druckerkommandos, Unterschiede zwischen der Benutzung unter BSD oder System V. Die Unterschiede betreffen nicht das Kommando selbst, sondern seine Schalter. In der nachfolgenden Tabelle 7.1 sind die wichtigsten Schalter zusammengefaßt.

Schalter		**Mnemonik**	**Bedeutung**
Sys.V	**BSD**		
-a	**a**	all	alle Prozesse aller Benutzer
-e		every	alle Prozesse eines Benutzers
	e	environment	gibt die Umgebung und die Argumente der Kommandos aus
-f		full	vollständige Liste; Untermenge der Ausgabe bei **-l**, zusätzlich wird bei COMMAND die vollständige Eingabezeile ausgegeben
	g	global	gibt alle Prozesse aus; ohne diesen Schalter werden nur *interessante* Prozesse ausgegeben; als uninteressant werden Prozesse betrachtet, die eine Gruppe anführen
-l	**l**	long	detaillierte Ausgabe
-u *user*		user	Prozesse, die von *user* gestartet wurden
	u	user	benutzerorientierte Ausgabe (mit **user**, **pcpu**, etc.)
-t *tty*	**t***tty*	terminal	Prozesse, die vom Terminal *tty* gestartet wurden

Tabelle 7.1: Schalter des Kommandos **ps**

Spezielle Schalter des verwendeten Systems sind gegebenenfalls dem on-line-manual zu entnehmen.

Der Schalter **-l** erlaubt eine detaillierte Ausgabe der aktuellen Prozesse:

```
$ ps  -l

 F    UID   PID  PPID PRI NI SIZE  RSS WCHAN   STAT TTY   TIME COMMAND
 0    501    83    82   1  0  396  544 11cb0c  S    pp0   0:00 -bash
 0    501   366    83  19  0   57  188 0       R    pp0   0:00 ps -l

$
```

Die einzelnen Informationen bedeuten:

`F`	„Flag“ des Prozesses (siehe on-line-manual)
`UID`	Benutzeridentifikationsnummer (User Identification Number)
`PID`	Prozeßidentifikationsnummer (Process Identification Number)
`PPID`	PID des Vaterprozesses (Parent PID)
`PRI`	Priorität des Prozesses; eine höhere Zahl bedeutet eine niedrigere Priorität
`NI`	Wert von **nice**

`SIZE`	virtuelle Größe (Kode+Daten+Stack) eines Prozesses
`RSS`	wirkliche Größe (resident size) eines Prozesses in Einheiten von Kilobyte
`WCHAN`	Adresse eines Ereignisses, auf das der Prozeß wartet, wenn er schläft oder wartet
`STAT`	Zustand des Prozesses (z.B. `R` für lauffähig, `S` für schlafend, `T` für gestoppt, `H` für angehalten)
`TTY`	Name des Terminals, von dem aus der Prozeß gelenkt wird
`TIME`	verbrauchte CPU-Zeit
`COMMAND`	Name des Kommandos, das von dem Prozeß ausgeführt wird

7.3 Prozeßpriorität

Die Prozesse werden in Systemprozesse und andere eingeteilt. Systemprozesse sind z.B. **swapper** und **init**, andere z.B. **sh** und **cat**. Systemprozesse unterscheiden sich von anderen durch eine in der Regel höhere dynamische Priorität. Diese wird für jeden Prozeß aus seiner Priorität (einer Zahl zwischen 0 und 39) und der Verweildauer des Prozesses im System, berechnet. Die Berechnung erfolgt jede Sekunde oder in Abhängigkeit von der Interaktion des Prozesses mit dem Systemkern.

Da jede Interaktion am Terminal eine Bearbeitung durch den Systemkern bedeutet, laufen Prozesse, welche die Arbeit am Terminal unterstützen, mit hoher Priorität. Wegen dieser Bevorzugung interaktiver Prozesse wird UNIX als „interaktives Betriebssystem“ bezeichnet.

Die Grenze zwischen Systemprozessen und anderen bildet eine Priorität von 20. Der gewöhnliche Benutzer kann die Priorität seiner Prozesse nur senken, nicht erhöhen. Eine Erhöhung der Priorität unter den Wert von 20 (0 ist höchste, 39 ist niedrigste Priorität) ist nur dem Systemverwalter gestattet.

Der maximale Wert, um den die Priorität erniedrigt werden kann, ist 19. Die Änderung der Priorität erfolgt mit dem Kommando **nice**. Dieses Kommando muß vor dem auszuführenden Befehl stehen.

```
$ ps  -l

 F    UID   PID  PPID PRI  NI SIZE  RSS WCHAN   STAT TTY   TIME COMMAND
 0    501  2048  2047   6   0  396  544 11cb0c  S    pp0   0:00 -bash
 0    501  2208  2048  21   0   57  188 0       R    pp0   0:00 ps -l

$ nice  ps  -l

 F    UID   PID  PPID PRI  NI SIZE  RSS WCHAN   STAT TTY   TIME COMMAND
 0    501  2048  2047   5   0  396  544 11cb0c  S    pp0   0:00 -bash
 0    501  2206  2048  23  10   57  188 0       R N  pp0   0:00 ps -l
```

Die Änderung der Priorität für **ps** steht in der Spalte `NI` (s.o.). Voreingestellt hat **nice** einen Wert von 10. Im Beispiel oben wurde die dynamische Priorität der SHELL[6] erhöht (5), die von **ps** gesenkt (23).

Sinnvoll ist die Senkung der Priorität insbesondere bei Kommandos, die im Hintergrund laufen. Die Senkung der Priorität von **du** um 17 erfolgt z.B. durch

```
$ nice  -17  du  /  >du.out  2>/dev/null &

2226

$
```

Eine Erhöhung der Priorität ist dem Systemverwalter vorbehalten. Sie geschieht durch die Angabe eines „negierten“ Wertes: $--wert$.

```
$ nice  --15  ps  -l

nice: cannot set priority: Permission denied

$
```

Übung 25 *Sie sollen ihr Arbeitsverzeichnis mit* **tar** *auf ein Magnetband archivieren. Dies soll durch einen Hintergrundprozeß mit der niedrigsten Priorität geschehen. Das Magnetband kann über die Datei* **/dev/rmt0** *angesprochen werden. Wie lautet das Kommando?*

[6] -bash, bourne again shell von GNU, - kennzeichnet die Bash als login-SHELL

7.4 Prozeßbeeinflussung durch Signale

Mit dem Kommando **kill** können Prozessen Signale geschickt werden (→ Tab.7.2). Es sind in der Regel Signale, die zur Beendigung der Prozesse führen.

Bezeichnung	**Nummer**	**Reaktion**	**Bedeutung**
SIGHUP	1	Beenden	Abtrennen (logout) des Terminals (hangup)
SIGINT	2	Beenden	Unterbrechen von der Tastatur (interrupt), entspricht **<ctrl c>**
SIGQUIT	3	*	Ende mit Ablage des Speicherabbilds des Prozesses beim Eintreffen des Signals (quit), entspricht **<ctrl \>**
SIGILL	4	*	unbekannte Anweisung (illegal instruction)
SIGTRAP	5	*	Kontrollunterbrechung (trace/breakpoint trap)

Tabelle 7.2: Signale der Version SVR4 (Teil 1)

Bezeichnung	Nummer	Reaktion	Bedeutung
SIGABRT	6	*	Abbruch (abort)
SIGEMT	7	*	EMT-Anweisung (emulation trap)
SIGFPE	8	*	Fehler bei Fließkommaoperation (floating point exception)
SIGKILL	9	Beenden	sofortiges Beenden des Prozesses (kann nicht abgefangen werden) (kill)
SIGBUS	10	*	Busfehler (bus error)
SIGSEGV	11	*	Segmentierungsfehler (segmentation fault)
SIGSYS	12	*	fehlerhafter Systembefehl (bad system call)
SIGPIPE	13	Beenden	in die Pipe wird geschrieben, ohne das gelesen wird (broken pipe)
SIGALRM	14	Beenden	Ende des Zeitintervalls (alarm clock)
SIGTERM	15	Beenden	Programm regulär beenden (terminated)
SIGUSR1	16	Beenden	erstes, benutzerdefiniertes Signal (user signal 1)
SIGUSR2	17	Beenden	zweites, benutzerdefiniertes Signal (user signal 2)
SIGCHLD	18	Ignorieren	Änderung des Zustandes des Sohnprozesses (child status changed)
SIGPWR	19	Ignorieren	Maschinenfehler (power fail/restart)
SIGWINCH	20	Ignorieren	Änderung der Fenstergröße (window size change)
SIGPOLL	21	Beenden	Kennzeichen von *streams* (pollable event)
SIGSTOP	22	Anhalten	Prozeß anhalten (stopped signal)
SIGTSTP	23	Anhalten	Prozeß durch Benutzersignal anhalten (stopped user), entspricht **<ctrl z>**
SIGCONT	24	Ignorieren	Fortsetzung der Tätigkeit (continued)
SIGTTIN	25	Anhalten	Warten auf Eingabe (tty input)
SIGTTOU	26	Anhalten	Warten auf Ausgabe (tty output)
SIGXCPU	27	*	Ende der verfügbaren CPU-Zeit (cpu time limit exceeded)
SIGXFSZ	28	*	Überschreitung der erlaubten Dateilänge (file size limit exceeded)

Tabelle 7.2: Signale der Version SVR4 (Teil 2)

Wird der Prozeß durch den Systemkern beendet, so wird bei den durch * gekennzeichneten Signalen das Speicherabbild des Prozesses zum Zeitpunkt der Unterbrechung in der Datei **core** hinterlegt.

Die Zuordnung der Nummern zu den Signalen (bis auf die fett gedruckten) und ihre Anzahl, kann sich von System zu System unterscheiden.

Die Signalnamen und ihre Zuordnung erhält man durch **kill -l**. Sie kann für das eine System

```
$ kill  -l

1) HUP            12) SYS           23) POLL
2) INT            13) PIPE          24) XCPU
3) QUIT           14) ALRM          25) XFSZ
4) ILL            15) TERM          26) VTALRM
5) TRAP           16) URG           27) PROF
6) LOST           17) STOP          28) WINCH
7) EMT            18) TSTP          29) PWR
8) FPE            19) CONT          30) USR1
9) KILL           20) CHLD          31) USR2
10) BUS           21) TTIN
11) SEG           22) TTOU

$
```

sein, für ein anderes

```
$ kill  -l

1) SIGHUP        2) SIGINT       3) SIGQUIT      4) SIGILL
5) SIGTRAP       6) SIGIOT       7) SIGBUS       8) SIGFPE
9) SIGKILL      10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR

$
```

Wird das Kommando **kill** ohne Schalter verwendet, so schickt es an den Prozeß das Signal 15.

```
$ du  /  >du.out  2>/dev/null    # kill  962

Terminated

$
```

Im Beispiel wurde das Signal 15 durch **kill 962** an den Prozeß, der **du** ausführt und die PID 962 hat, von einem anderen Terminal geschickt und damit das Programm beendet. Die Bestätigung `Terminated` wurde von einer Korn-SHELL ausgegeben.

Das Signal 2 entspricht der Unterbrechung eines Prozesses im Vordergrund durch eine Eingabe von **<ctrl c>** an der Tastatur. Das gleiche gilt für das Signal 23 und **<ctrl z>**.

Bei der Angabe der Signale können die Signalnummern oder die Bezeichnungen verwendet werden.

```
$ du  /  >du.out  2>/dev/null &

[1] 3873

$ ps

  PID TTY STAT  TIME COMMAND
 2048 pp0 S     0:01 -bash
 3873 pp0 R     0:02 du /
 3874 pp0 R     0:00 ps

$ kill  -TSTP  3873
$ ps

  PID TTY STAT  TIME COMMAND
 2048 pp0 S     0:01 -bash
 3873 pp0 T     0:08 du /
 3875 pp0 R     0:00 ps

<return>

[1]+  Stopped                 du / >du.out 2>/dev/null

$
```

Ein Signal, das weder ignoriert noch maskiert werden kann, ist das Signal 9.

```
$ kill  -9  3873
$

<return>

[1]+  Killed                  du / >du.out 2>/dev/null

$
```

Die Form der Meldungen (Stopped, Killed) hängt von der verwendeten SHELL ab. Die Bourne-SHELL gibt z.B. überhaupt keine Meldungen aus, die Bash erst nach der Eingabe von RETURN.

Durch

```
$ kill -9 0
```

werden alle vom Benutzer gestarteten Programme beendet und der Benutzer vom System abgemeldet (logout).

Kann ein im Vordergrund laufender Prozeß durch **<ctrl c>** nicht beendet werden, so sollte sich der Benutzer an einem anderen Terminal anmelden und nach Feststellung der PID des Prozesses (**ps**), diesen durch **kill -9** *PID* beenden.

Die Reaktion der SHELL auf ankommende Signale kann durch das Kommando **trap** geändert werden. Dieses wird am häufigsten innerhalb von SHELL-Scripts verwendet. Es hat die allgemeine Form

trap [[*befehlsfolge*] *signalfolge*]

Enthält *befehlsfolge* mehrere Befehle, so müssen diese durch Anführungszeichen oder Hochkommas eingeschlossen werden. Das Argument *signalfolge* enthält diejenigen Signale, deren Abfangen (Maskierung) die Abarbeitung der Befehle aus *befehlsfolge* induziert.

Eine Reaktion auf Signal 11 kann z.B. durch

```
$ trap 'echo Ich reagiere auf Signal 11' 11
$
```

eingestellt werden. Fängt die SHELL das Signal ab, so führt sie die spezifizierte *befehlsfolge* aus.

```
$ kill -11 $$
Ich reagiere auf Signal 11
$
```

Die Anwendung innerhalb eines SHELL-Scripts zeigt das nachfolgende Beispiel:

```
# Verwendung von trap
trap 'echo Programm unterbrochen; exit 1' 2
while true
do
   echo Programm arbeitet
done
```

□

In der ersten Zeile wird die Reaktion auf ein ankommendes Signal 2 eingestellt. Nach seinem Abfangen werden die beiden in Anführungszeichen gesetzten Kommandos ausgeführt. Zuerst wird die Meldung „Programm unterbrochen“ ausgegeben, danach wird der Prozeß mit dem Rückkehrkode 1 beendet. Ohne **exit** würde das Script in der unendlichen Schleife bleiben (**true** hat den Rückkehrkode 0 und ist folglich immer wahr).

Das Script wird in einer eigenen SHELL (Sohnprozeß) interpretiert:

```
$ sh  skript

Programm arbeitet
Programm arbeitet
Programm arbeitet
Programm arbeitet

<ctrl c>

Programm unterbrochen
```

Ohne Schalter gibt **trap** die eingestellten Reaktionen auf ankommende Signale aus:

```
$ trap

trap -- 'echo Ich reagiere auf Signal 11' SIGSEGV

$
```

Durch die Angabe einer „leeren“ *befehlsfolge* werden diejenigen Signale definiert, die durch die SHELL ignoriert werden sollen.

```
$ trap  ''  12
$ kill  -12  $$
$ trap

trap -- 'echo Ich reagiere auf Signal 11' SIGSEGV
trap -- '' SIGUSR2

$
```

Das Signal 9 kann nicht ignoriert werden.

```
$ trap  ''  9
$ kill  -9  $$
```

```
Killed

$
```

Wird das Argument *befehlsfolge* nicht angeführt, so wird den Signalen ihre ursprüngliche Reaktion auf ankommende Signale eingestellt.

```
$ trap  9  11  12
$ trap
$
```

Für das Signal 9 wurde zwar durch **trap '' 9** eine neue Reaktion (ignorieren) eingestellt, sie wird bei der Verwendung des Signals jedoch nicht beachtet.

Wird für *signalfolge* das Signal 0 angegeben, so wird *befehlsfolge* nach dem Beendigen der aktuellen SHELL ausgeführt.

```
$ trap  'echo  Exit from current shell'  0
$<ctrl d>

Exit from current shell

login:
```

Das Signal 1 (SIGHUP) wird an die Prozesse des Benutzers geschickt, sobald die Verbindung zwischen dem Terminal und dem Rechner unterbrochen wird (logout). Soll ein Programm nach dem Abmelden von System weiterlaufen, so muß das Signal 1 abgefangen werden.

```
$ (trap  ''  1; du  /  >du.out  2>/dev/null) &

[1]     1522

$<ctrl d>

login:
```

Die runden Klammern starten zusammen mit **&** eine neue SHELL im Hintergrund.

Wegen **trap '' 1** wird beim Abmelden des Benutzers das Signal 1 abgefangen und das Kommando **du** im Hintergrund zu Ende geführt.

Den gleichen Effekt erzielt man mit dem Kommando **nohup** (**no hangup**). Kommandos, die als Parameter von **nohup** gestartet werden, ignorieren das Signal 1.

```
$ nohup  du  /  >du.out  2>/dev/null &
```

```
[1]   6081
```

$ kill -1 6081

$ ps

```
  PID TTY STAT  TIME COMMAND
 4362 pp0 S     0:00 -bash
 6081 pp0 R N   0:05 du /
 6083 pp0 R     0:00 ps
```

```
$
```

Erfolgt die Ausgabe des Kommandos auf dem Bildschirm, so wird automatisch die Datei **nohup.out** erzeugt.

$ nohup du / &

```
[1]   6099
```

```
$ nohup: appending output to 'nohup.out'
```

<return>

$ ps -l

```
 F    UID   PID  PPID PRI NI SIZE  RSS WCHAN   STAT TTY   TIME COMMAND
 0    501  4362  4361   2  0  396  548 11cb0c  S    pp0   0:00 -bash
 0    501  6099  4362  25  5  101  268 0       R N  pp0   0:03 du /
 0    501  6102  4362  19  0   57  188 0       R    pp0   0:00 ps -l
```

```
$
```

Bei der Ausführung eines Kommandos mit **nohup** wird seine Priorität automatisch um 5 gesenkt (s.o.). Durch das Voranstellen von **nice** kann eine weitere Senkung erzielt werden.

$ nice -15 nohup du / &

```
[1]   6121
```

```
$ nohup: appending output to 'nohup.out'
```

<return>

```
$
```

7.5 Vorgabe der Startzeit für einen Prozeß

Zur Arbeitsplanung bietet UNIX mehrere Programme an. Sollen bestimmte Tätigkeiten periodisch wiederholt werden, so kann dies mit dem „daemon“ **cron** veranlaßt werden. Dies ist jedoch in der Regel nur dem Systemverwalter erlaubt.

Mit Hilfe des Kommandos **at** kann veranlaßt werden, daß Befehle oder Scripts zu bestimmten Zeiten ausgeführt werden. Der einzige zwingende Parameter des Kommandos ist die Zeit.

```
$ at  12:45
echo  Mittagessen!
<ctrl d>

job leibner.817818600.a at Fri Dec 01 12:50:00 1995

$
```

Jedes Kommando wird in einer eigenen Zeile aufgeführt und die Eingabe durch **<ctrl d>** beendet. Die Kommandos werden zum nächsten Zeitpunkt, der der Zeitangabe entspricht, ausgeführt. Falls **at** Ausgaben erzeugt, so werden diese dem Benutzer per „e-mail“ zugestellt.

```
$ mailx
"/usr/spool/mail/leibner": 1 message 1 new
>N  1 root              Fri Dec  1 12:50  15/437
?t

Message  1:
From root Fri Dec  1 12:50:01 1995
Date: Fri, 1 Dec 1995 12:50:01 +0100
From: system PRIVILEGED account <root>
Apparently-To: leibner

Mittagessen!


**********************************************
Cron: The previous message is the standard output
      and standard error of one of your cron commands.

?d
?q
```

$

Stehen mehrere Kommandos in einem Script, so ist der Schalter **-f** zu verwenden:

$ at -f skript 13

```
job leibner.817819200.a at Fri Dec 01 13:00:00 1995
```

$

Mit dem Schalter **-k** kann anstatt der voreingestellten Bourne- eine Korn-SHELL, mit **-c** eine C-SHELL, gewählt werden.

Zur Angabe der Zeit gibt es vielerlei Möglichkeiten. Eine oder zwei Ziffern bedeuten ganze Stunden, fehlt die Angabe von **pm** oder **am**, so wird ein 24-Stunden Zyklus angenommen. Drei oder vier Ziffern geben Stunden und Minuten an. Erlaubt sind z.B. folgende Angaben:

$ at 5pm
$ at 1735
$ at 14:07

Vordefinierte Zeiten sind **now** (aktuelle Zeit), **noon** (12:00) und **midnight** (24:00).

Neben der Zeit kann auch ein Datum angegeben werden. Vordefiniert ist **tomorrow**. Folgende Angaben sind z.B. erlaubt:

$ at 23:57 next Friday
$ at noon
$ at midnight
$ at noon tomorrow
$ at 712 Jan 24, 1997

Mit einem Pluszeichen kann ein Inkrement bestimmt werden.

$ at now + 7 days
$ at now next week

Vordefiniert ist **next** als **+ 1**.

Die durch **at** erzeugten „jobs“ werden in eine Warteschlange geschrieben. Mit dem Schalter **-l** erhält man Informationen über die eingereihten Aufgaben.

$ at -l

```
leibner.817902120.a     Sat Dec 02 12:02:00 1995
leibner.817902180.a     Sat Dec 02 12:03:00 1995
```

Eine ähnliche Funktion hat das Kommando **atq**.

Mit Hilfe des Schalters **-r** oder dem Kommando **atrm** können Einträge aus der Warteschlange gelöscht werden.

```
$ atrm   leibner.817902120.a

at file: leibner.817902120.a deleted

$
```

If many faultes in this book you fynde,
Yet think not the correctors blynde;
If Argos heere hymselfe had beene
He should perchance not all have seene.

Richard Shacklock, 1565

8 Die SHELL

Die SHELL ist sowohl Kommandointerpreter als auch Programmiersprache. Arbeitet sie als Kommandointerpreter, so setzt sie Angaben in der Eingabezeile in Befehle an das Betriebssystem um. Als Programmiersprache verarbeitet sie Befehlsgruppen aus einer Datei, dem sogenannten Script.

Die interaktive SHELL dient als Schnittstelle zwischen dem Benutzer und dem Systemkern (→ Abb.7.1). Sie ist somit kein Bestandteil des Betriebssystems UNIX. Diese Tatsache führte zur Entwicklung von über einem Dutzend SHELLs, von denen jedoch nur wenige eine weitere Verbreitung erfahren haben.

Die erste SHELL wurde nach ihrem Erfinder, Steven Bourne, Bourne-SHELL genannt (Programmname: **sh**). Sie wurde zum ersten Mal bei AT&T in die UNIX-Version 7 aus dem Jahre 1979 implementiert (→ Abb.1.2). Auch nach vielen Änderungen des Betriebssystems UNIX ist die praktisch unveränderte Bourne-SHELL die Standardshell von UNIX geblieben. Sie ist Bestandteil jeder UNIX-Implementation.

Mitte der achziger Jahre (1986) hat David Korn, der ebenfalls in den Bell Labs von AT&T tätig war, eine Alternative zur Bourne-SHELL geschrieben (Programmname: **ksh**). Die nach ihm benannte Korn-SHELL ist eine um eigene und um die besten Eigenschaften der C-SHELL (History- und Job-Control-Mechanismus, Aliasbildung) aufwärts kompatible Erweiterung der Bourne-SHELL. Die Korn-SHELL ist eine kommerzielle SHELL. Sie wird in der Regel zusammen mit UNIX System V Release 4 aus dem Jahre 1989 ausgeliefert. Die üblicherweise mitgelieferte Version 11/16/88 ist nicht POSIX[1]-kompatibel. Eine Verbesserung der Kompatibilität zu POSIX wurde mit der Version 12/28/93 erzielt, in die weitere neue Eigenschaften implementiert wurden.

Ein weiteres Mitglied der Bourne-SHELL-Familie ist die Bourne-Again-SHELL[2] – Bash (Programmname: **bash**). Sie wurde von Brian Fox im Rahmen des GNU-Projekts der Free Software Foundation (FSF) 1988 geschrieben. Ab 1989 beteiligte sich Chet Ramey an ihrer Entstehung, heute ist er für sie der verantwortliche Ansprechpartner.

Obwohl die Bourne-SHELL nach wie vor die Standardshell ist, nimmt die Popularität der Bourne-Again-SHELL immer mehr zu. Als Programmiersprache ist sie zur Bourne-SHELL vollkommen kompatibel, als interaktive SHELL vereinigt sie die besten Eigenschaften der C- und Korn-SHELL. So hat sie z.B. die History-Mechanismen beider

[1] IEEE 1003.2 Portable Operating System Interface
[2] WWW-Adresse: ftp://prep.ai.mit.edu/pub/gnu/bash-1.14.6.tar.gz (derzeitige Version der Bash)

SHELLs (die Bourne-SHELL bietet nichts Vergleichbares), sowie die Bildung von Aliasnamen und den Job-Control-Mechanismus der C-SHELL, übernommen. Ihre Programmiereigenschaften wurden um Funktionen, weitere Kontrollstrukturen, die Arithmetik ganzer Zahlen, eine bessere Kontrolle der Ein- und Ausgabe und vieles mehr, erweitert.

Es wird daher empfohlen, in der Bash zu arbeiten, die Programmierung jedoch aus Kompatibilitätsgründen in der Syntax der Bourne-SHELL durchzuführen.

Das derzeit letzte Mitglied der Bourne-SHELL-Familie ist die 1990 entstandene Z-SHELL (Programmname: **zsh**). In ihr hat ihr Entwickler, Paul Falstad, die meisten Eigenschaften aller anderer SHELLs zusammengefaßt. Zur Zeit arbeitet Falstad an einer kommerziellen Version dieser SHELL.

Die erste weitverbreitete Alternative zur Bourne-SHELL war die C-SHELL (Programmname: **csh**). Sie wurde von Bill Joy an der University of California at Berkeley einige Jahre nach dem Erscheinen der Bourne-SHELL entwickelt und in allen BSD-UNIX-Versionen implementiert. Der Name der SHELL soll an die Programmiersprache C erinnern, deren Syntax derjenigen der SHELL ähnlich ist. Diese Tatsache sollte den C-Programmierern die Arbeit mit der C-SHELL erleichtern, durch die neue Syntax wurde sie jedoch zu den SHELLs der Bourne-Familie inkompatibel. Wegen des erstmals verfügbaren History- und Job-Control-Mechanismus sowie der Möglichkeit Aliasnamen zu bilden, wurde die C-SHELL schnell zur Standardshell in wissenschaftlichen Einrichtungen und Hochschulen.

Da die Bash alle diese Eigenschaften und weitere mehr vereint, ist sie gerade dabei, die C-SHELL vielerorts zu verdrängen.

Das zweite Mitglied der C-SHELL-Familie ist die TC-SHELL[3] (Programmname: **tcsh**). Das Entwicklungsprojekt wurde Ende der siebziger Jahre von Ken Greer an der Carnegie-Mellon University gestartet und in den achziger Jahren von Paul Placeway an der Ohio State University fortgesetzt. Die TC-SHELL wird derzeit von der Cornell University gewartet und verbessert.

Gemäß der ausgesprochenen Empfehlung, werden im folgenden kurz die speziellen Eigenschaften der Bash angesprochen, anschließend wird auf die Programmierung der Bourne-SHELL genauer eingegangen.

8.1 Besonderheiten der Bash

Nicht alle Versionen der Bourne-SHELL haben die gleichen Eigenschaften. So unterstützt z.B. die mit dem System V verteilte SHELL Funktionen sowie manche weiteren Eigenschaften der Bash bzw. Korn-SHELL. Der folgende grobe Vergleich erfolgt daher zwischen der aktuellen Bash (1.14.6) und der Bourne-SHELL der Version 7.

[3] Das T soll für „Tenex“ stehen, einem Betriebssystem, das auf alten PDP-10-Rechnern verwendet wurde (die Originalarbeiten an der TC-SHELL wurden auf so einem System durchgeführt)

Die Bash ist fast vollständig kompatibel zur Bourne-SHELL. Die einzige Eigenschaft der Bourne-SHELL, die von der Bash nicht unterstützt wird, ist die Verwendung von ^ für das Pipesymbol |.

In der Bash ist eine Modifikation des Kommandos **cd** implementiert. So kann durch **cd -** in das Verzeichnis zurückgewechselt werden, in dem vorher gearbeitet wurde.

```
~/mails/fahrplaene$ cd
~$ cd  -
~/mails/fahrplaene$ cd  -
~$
```

Die Tilde (~) kann zum Vervollständigen des Pfades zum eigenen Heimatverzeichnis

```
~/mails/fahrplaene$
~/mails/fahrplaene$ cd  ~/NETZ
~/NETZ$ pwd

/ws30/zeta/usr/home/leibner/NETZ

~/NETZ$
```

bzw. zum Heimatverzeichnis eines anderen Benutzers

```
~/NETZ$ cd  ~pel
/ws30/klapptnix/home/pel$
```

verwendet werden.

Der Job-Control-Mechanismus vergibt neben der PID den Prozessen zusätzlich eine Job-Nummer.

```
~$ du  /usr  >du.out  2>/dev/null &

[1] 2211

~$
```

Die Nummer ([1]) wird von der Bash vergeben und gibt an, wieviele Prozesse im Augenblick in der SHELL im Hintergrund laufen. Beim Start eines weiteren Hintergrundprozesses wird diese Nummer erhöht:

```
~$ skript &
```

```
[2] 2217

~$
```

Die Beendigung eines Hintergrundprozesses wird nach einer Eingabe (z.B. **<return>**) gemeldet.

```
~$ <return>

[2]+  Done                    skript
```

Falls der Rückkehrstatus ungleich Null ist, so wird dies durch die SHELL gemeldet.

```
~$ <return>

[1]+  Exit 1                  du /usr >du.out 2>/dev/null

~$
```

Mit **jobs -l** können die im Hintergrund laufenden Prozesse angezeigt werden (der Schalter **-l** bewirkt die zusätzliche Ausgabe der PID).

```
~$ jobs  -l

[1]-  2246 Running                 skr1 &
[2]+  2247 Running                 skr2 &

~$
```

Das Pluszeichen gibt an, welcher Jobstatus sich zuletzt geändert hat, das Minuszeichen gibt an, welcher Jobstatus sich zuvor geändert hat.

Durch **fg %**n (**foreground**) kann der Job mit der Nummer n in den Vordergrund geholt werden.

```
~$ fg  %1

skr1

<ctrl c>
~$
```

Ein im Vordergrund laufender Job kann durch **<ctrl z>** angehalten werden (→ Tab.7.2).

```
~$ skript
<ctrl z>
```

```
[1]+  Stopped                 skript

~$
```

Ein angehaltener Job kann durch **fg** wieder im Vordergrund fortgesetzt werden.

```
~$ fg

skript
```

<ctrl c>

```
~$
```

Dies ist z.B. von Vorteil, wenn im **vi** ein Programm bearbeitet wird, das zwischendurch auf Fehler getestet werden soll. Man kann dann durch **<ctrl z>** den **vi** verlassen, das Programm testen und durch **fg** in den **vi** wieder zurückkehren.

Wurden mehrere Jobs angehalten, so muß **fg** %n mit der entsprechenden Jobnummer n verwendet werden.

Um ein im Vordergrund laufendes Programm weiter im Hintergrund laufen zu lassen, muß es zuerst gestoppt und dann durch **bg** (**background**) in den Hintergrund geschickt werden[4].

```
~$ skript
```

<ctrl z>

```
[1]+  Stopped                 skript

~$ bg

[1]+ skript &

~$
```

Soll ein im Hintergrund laufender Prozeß beendet werden, so kann dazu neben der PID auch die Jobnummer verwendet werden.

```
~$ skript &

[1] 2390
```

[4] wenn das Programm auf einem entfernten Rechner in einem Netz gestartet wurde, so kann dies zu Fehlern führen

```
~$ kill  %1
~$ <return>

[1]+  Terminated              skript

~$
```

Neben dem on-line-manual hat die Bash noch eine **help**-Funktion. Diese Funktion gibt Informationen über alle „inneren“ (in der SHELL implementierten) Kommandos aus.

Detaillierte Informationen über ein Kommando erhält man z.B. durch **help cd**, durch Angabe eines partiellen Namens, z.B. **help re**, werden Informationen zu den Kommandos, die mit **re** anfangen, nämlich **read**, **readonly** und **return**, ausgegeben.

```
~$ help  help

help: help [pattern ...]
    Display helpful information about builtin commands.  If PATTERN is
    specified, gives detailed help on all commands matching PATTERN,
    otherwise a list of the builtins is printed.
```

Durch die Bildung von Aliasnamen können Abkürzungen für komplexere Anweisungen definiert werden. Dies kann sowohl in der Kommandozeile als auch in der Datei **.bash_profile**[5] oder **.bashrc**[6] erfolgen.

```
~$ alias  cdfahr='cd  ~/mails/fahrplaene'
~$ cdfahr
~/mails/fahrplaene$
```

Zwischen dem Aliasnamen (**cdfahr**), dem Gleichheitszeichen und dem Kommando (**cd ~/mails/fahrplaene**) dürfen keine Leerzeichen stehen. Da das Kommando aus mehr als einem Wort besteht, muß es z.B. in Hochkommas eingeschlossen werden.

```
~/verz$ alias  ls='ls  -F'
~/verz$ ls

tel-1   tel-2   verz/

~$ alias
```

[5] diese Datei entspricht der Datei **.profile** der Bourne- und Korn-SHELL, sie wird nur beim Anmelden am System gelesen; ein alternativer Name ist **.bash_login**, abgeleitet von der Datei **.login** der C-SHELL

[6] wird bei jedem Start einer Subshell gelesen

```
alias cdfahr='cd ~/mails/fahrplaene'
alias ls='ls -F'

~$
```

Die Bash untersucht nur das erste Wort des Kommandos auf weitere Aliasbildung. Ist dieses Wort identisch mit dem Aliasnamen, so wird es kein zweites Mal expandiert. Dies verhindert eine Rekursion, die sonst in dem obigen Beispiel aufgetreten wäre.

Durch **alias** werden alle getroffenen Aliasbildungen aufgelistet, durch **unalias** kann ein Aliasname wieder gelöscht werden:

```
~$ unalias cdfahr
~$ alias
alias ls='ls -F'

~$
```

Gegenüber der C-SHELL erlaubt die Bash keine Argumente in Aliasbildungen. Die Möglichkeit, Funktionen zu definieren, kompensiert diesen Mangel jedoch bei weitem.

Wie an den Beispielen in diesem Abschnitt gezeigt, können die Prompt-Variablen (**PS1-4**) durch den Benutzer umdefiniert werden. So wurde der bisher verwendete Prompt durch

```
PS1='\w$ '
```

so definiert, daß immer das aktuelle Arbeitsverzeichnis (**\w**) ausgegeben wird (Voreinstellung: **bash$**). Die Zuweisung kann entweder in der Datei **.bash_profile** oder in der Eingabezeile erfolgen.

In der nachfolgenden Tabelle 8.1 sind einige weitere Möglichkeiten zur Umdefinition der Prompt-Variablen zusammengefaßt.

Kommando	Bedeutung
\d	Datum im Format „Wochentag Monat Tag“ (z.B. Mon Mar 4)
\h	Rechnername (z.B. zeta)
\s	Name der SHELL (z.B. bash)
\t	Aktuelle Zeit (z.B. 11:21:49)
\u	Benutzername (z.B. leibner)
\w	Arbeitsverzeichnis (z.B. ~)
\!	Nummer des aktuellen Kommandos in der History-Liste

Tabelle 8.1: Zeichen zur Prompt-Anpassung

So erhält man z.B. durch

PS1='\w[\!]$ '

als Prompt

~/skriptum/unix[135]$

Dabei steht **135** für das 135-te Kommando in der History-Liste (wie diese Zahl zur Wiederholung des Kommandos benutzt werden kann, wird im nächsten Abschnitt erklärt).

Zum Abschluß dieser kurzen Einführung seien noch die sehr nützlichen SHELL-Kommandos **pushd** und **popd**[7] erwähnt, mit deren Hilfe in einem Kellerspeicher (Stack) Verzeichnisse gespeichert (**pushd**) und wieder aus ihm entfernt (**popd**) werden können.

Durch die Bildung von Aliasnamen (in **.bash_profile**) können die beiden Kommandos abgekürzt werden:

```
~$ alias  pd=pushd
~$ alias  pop=popd
~$
```

Durch **dirs** wird der aktuelle Inhalt des Speichers ausgegeben.

```
~$ dirs

~

~$ pd  ~/skriptum/unix

~/skriptum/unix ~

~/skriptum/unix$ pd  ~/Latex/verschiedenes

~/Latex/verschiedenes ~/skriptum/unix ~

~/Latex/verschiedenes$ dirs

~/Latex/verschiedenes ~/skriptum/unix ~

~/Latex/verschiedenes$ pd
```

[7] übernommen aus der C-SHELL

```
~/skriptum/unix ~/Latex/verschiedenes ~

~/skriptum/unix$ pd  +2

~ ~/skriptum/unix ~/Latex/verschiedenes

~$ pop

~/skriptum/unix ~/Latex/verschiedenes

~/skriptum/unix$ pop  +1

~/skriptum/unix

~/skriptum/unix$
```

Jeder Wechsel in ein neues Verzeichnis mittels **pd** bewirkt einen Eintrag im Kellerspeicher, wodurch die anderen Verzeichnisse weiter in die Tiefe (nach rechts) verschoben werden. Die erneute Eingabe von **pd** tauscht Platz eins und zwei aus. Soll ein Wechsel in ein tiefer liegendes Verzeichnis erfolgen, so muß **pd** $+n$ eingegeben werden, wobei n die Tiefe im Kellerspeicher (beginnend bei 0, die Voreinstellung ist 1) angibt. Mit **dirs** wird der aktuelle Stand des Kellerspeichers ausgegeben. Mit **pop** wird der oberste Eintrag, mit **pop** $+n$ der n-te, aus dem Speicher entfernt.

Von den zusätzlichen vordefinierten SHELL-Variablen sei hier nur **CDPATH** erwähnt, die es erlaubt, häufig benutzte Pfade für das Kommando **cd** zur Verfügung zu stellen. Durch

```
~$ CDPATH=:~/skriptum/unix:~/skriptum:~/Latex:~
~$
```

wird z.B. für **cd** das Arbeitsverzeichnis (leeres Feld vor dem ersten **:**), das Heimatverzeichnis (~), sowie weitere Verzeichnisse als Pfade eingestellt.

```
~$ cd  unix

/ws30/zeta/usr/home/leibner/skriptum/unix

~/skriptum/unix$ cd  NETZ

/ws30/zeta/usr/home/leibner/NETZ

~/NETZ$
```

Aus der Menge zusätzlicher implementierter Eigenschaften (→ [15]) soll hier nur noch auf den History-Mechanismus genauer eingegangen werden.

8.1.1 History-Mechanismus der Bash

Die Bash bietet die Möglichkeit, frühere Kommandos sowohl unverändert als auch modifiziert wiederzuverwenden. (Die im Abschnitt 3.4 angeführten Kommandos der Korn-SHELL (→ Tab.3.13) werden von der Bash nicht unterstützt.)

Die Einstellung des **vi** als Editor erfolgt entweder in der Datei **.bash_profile** oder in der Eingabezeile durch

```
~$ set  -o  vi
~$
```

Zu den vordefinierten SHELL-Variablen des History-Mechanismus zählt unter anderen

```
~$ echo  $HISTCONTROL
ignoredups
~$
```

dessen Wert `ignoredups` angibt, daß sukzessiv wiederholte Kommandos nur einmal in die History-Liste eingetragen werden sollen. Die im Zusammenhang mit dem History-Mechanismus verwendeten SHELL-Variablen können (→ Tab.8.2) in der Datei **.bash_profile** (oder in der Eingabezeile) auf einen neuen Wert gesetzt werden.

SHELL-Variable	Bedeutung
HISTCONTROL=*wert*	für *wert* kann **ignorespace** stehen, wonach keine Kommandos mit führenden Leerzeichen in die History-Liste übernommen werden, **ignoredups**, wodurch sukzessiv wiederholte Kommandos nur einmal eingetragen werden, bzw. **ignoreboth**, was sowohl **ignorespace** als auch **ignoredups** aktiviert
HISTFILE=*dateiname*	für *dateiname* kann ein neuer Name für die Datei gewählt werden, in der beim Verlassen des Systems der Inhalt der History-Liste gespeichert werden soll; voreingestellt ist **.bash_history**
HISTFILESIZE=*wert*	für *wert* kann ein neuer Wert für die maximale Anzahl der in der History-Datei zu speichernden Kommandos festgelegt werden; voreingestellt ist 500
HISTSIZE=*wert*	für *wert* kann ein neuer Wert für die maximale Anzahl der in der History-Liste zu speichernden Kommandos festgelegt werden; voreingestellt ist 500

Tabelle 8.2: Änderung der Werte vordefinierter SHELL-Variablen

Ist der Wert von **HISTFILESIZE** kleiner als derjenige von **HISTSIZE**, so wird die Länge der während der Arbeit in der Bash im Speicher gehaltenen History-Liste beim Schreiben in die History-Datei auf den kleineren Wert gekürzt.

8.1.1.1 Vervollständigung von Eingaben

Mit Hilfe von TAB kann ein unvollständiger Name in der Eingabezeile vervollständigt werden. Verzeichnissen wird dabei ein / angehängt:

```
~$ cd  sk<tab>riptum/
~/skriptum$
```

Steht der Name für ein Kommando, eine Funktion oder einen Dateinamen (Script), so folgt der Vervollständigung ein Leerzeichen, so daß gegebenenfalls weitere Optionen eingegeben werden können.

```
~/verz$ sk<tab>ript    #   = Cursorposition
```

Ist der Name mehrdeutig, so werden nach zweimaliger Eingabe von TAB die Alternativen angezeigt.

```
~$ cd  s<tab><tab>

skriptum     soubory.txt

~$ cd  s
```

Ist dem zu vervollständigenden Text ein ~ vorangestellt, so versucht die Bash den Namen zu einem Benutzernamen zu expandieren, ist es ein $, so erfolgt die Expansion zu einer SHELL-Variablen, bei einem @ zu einem Rechnernamen.

8.1.1.2 Editieren der Eingabezeile

Neben der Möglichkeit, Eingaben in der Kommandozeile zu vervollständigen, besteht auch die Option, sie zu editieren. Dabei gilt für das Editieren mit dem **vi** das bereits für den History-Mechanismus der Korn-SHELL (→ S.28) gesagte. Zusätzlich dazu wurde in der Bash der History-Mechanismus der C-SHELL implementiert.

Der Zugriff auf Kommandos in der History-Liste erfolgt dabei über Ereignis- (Zeile) und Wortselektoren (Spalte).

Das Verhalten des selektierten Kommandos kann durch die Angabe von **:***zeichen* zusätzlich modifiziert werden. So bewirkt z.B. die Angabe von **:p**, daß das selektierte Kommando lediglich ausgegeben nicht jedoch ausgeführt wird.

Wurde für den Prompt **PS1='\w[\!]$ '** eingestellt, so können die Nummern der Kommandos direkt am Prompt abgelesen und danach als Ereignisselektor für das betreffende Kommando verwendet werden.

```
~/Latex/org[6]$ history
```

```
        1  cd unix
        2  ls *tex
        3  cd org
        4  ls
        5  cp windows.tex ~/
        6  history

~/Latex/org[7]$ !1

cd unix
/ws30/zeta/usr/home/leibner/skriptum/unix

~/skriptum/unix[8]$ !2

kap1.tex*  kap4.tex*  kap7.tex    titel.tex*      uebloes.tex*
kap2.tex*  kap5.tex*  kap8.tex*   titelgen.tex*   unix.tex*
kap3.tex*  kap6.tex*  skdef.tex*  titelseite.tex  visp.tex*

~/skriptum/unix[9]$ !5:0  unix.tex  !5:2

cp unix.tex ~/

~/skriptum/unix[10]$ !3

cd org
/ws30/zeta/usr/home/leibner/Latex/org

~/Latex/org[11]$ cd  t<tab>dot
~/Latex/org/tdot[12]$ dvips  tdot  -r  -o  tdot.dvi.ps

This is dvipsk 5.58f Copyright 1986, 1994 Radical Eye Software
' TeX output 1996.02.21:0941' -> tdot.dvi.ps
<texc.pro>. [1]

~/Latex/org/tdot[13]$ !!:gs/tdot/tdotxx/

dvips tdotxx -r -o tdotxx.dvi.ps
This is dvipsk 5.58f Copyright 1986, 1994 Radical Eye Software
' TeX output 1996.03.04:1439' -> tdotxx.dvi.ps
<texc.pro><8r.enc><texps.pro><special.pro>. [1<prswkr.eps><betrag.eps>
<phase.eps>]

~/Latex/org/tdot[14]$ xdvi  tdot
~/Latex/org/tdot[15]$ ^tdot^tdotxx

xdvi tdotxx
```

```
~/Latex/org/tdot[16]$ !dv:p

dvips tdotxx -r -o tdotxx.dvi.ps

~/Latex/org/tdot[17]$
```

Mit **history** wird die aktuelle History-Liste ausgegeben.

Der Ereignisselektor für das erste Kommando ist **!1**, für das zweite **!2** und für das dritte **!3**. Durch **!5:0** wird das erste Wort des fünften Ereignisses selektiert (das Zählen beginnt wieder bei 0), durch **!5:2** das dritte.

Nach Eingabe von **!!** wird das vohergehende Kommando unverändert wiederholt, nach **!!:gs/tdot/tdotxx/** erfolgt die Wiederholung nach dem Ersatz von **tdot** durch **tdotxx**. Dabei ersetzt **s** (**substitute**) nur das erste Vorkommen von **tdot** gegen **tdotxx**, da aber zusätzlich **g** (**global**) angegeben ist, geschieht dies auch für jedes weitere Vorkommen von **tdot** in der Kommandozeile.

Nur das erste Vorkommen einer Zeichenkette wird durch **^tdot^tdotxx^** ersetzt. Das letzte ^ kann dabei weggelassen werden.

Durch **!dv:p** wird auf das erste (rückwärts gerechnet) Kommando zugegriffen, das mit **dv** beginnt, wegen **:p** (**print**) wird es nur ausgegeben aber nicht ausgeführt.

Einige häufig verwendete History-Kommandos sind in der nachfolgenden Tabelle 8.3 zusammengefaßt.

Kommando	**Bedeutung**
history	gibt die History-Liste auf den Bildschirm aus
!*N*	ruft das Kommando mit der Nummer *N* auf
!*zeiket*	ruft das erste Kommando auf, das mit *zeiket* beginnt
!!	ruft das vorhergehende Kommando auf
!*N***:***n*	das *n*-te Wort des Kommandos mit der Nummer *N*; **0** steht für das erste Wort (üblicherweise der Kommandoname), **$** für das letzte
!*N***:p**	Kommando mit der Nummer *N* ausgeben ohne es auszuführen
^*zeiket1***^***zeiket2***^**	ersetze im vorhergehenden Kommando *zeiket1* durch *zeiket2*
!*N***:s/***zeiket1***/***zeiket2***/**	ersetze im Kommando mit der Nummer *N* *zeiket1* durch *zeiket2*
!*N***:gs/***zeiket1***/***zeiket2***/**	ersetze im Kommando mit der Nummer *N* alle *zeiket1* durch *zeiket2*

Tabelle 8.3: History-Kommandos (übernommen aus der C-SHELL)

8.1.1.3 Das **fc**-Kommando

Mit dem **fc**-Kommando (**fix command**) können frühere Kommandos ausgegeben, eins oder mehrere davon editiert und gegebenenfalls modifiziert ausgeführt werden.

Wie beim History-Mechanismus der C-SHELL so können auch **fc** die Nummern der Kommandos oder teilweise spezifizierte Kommandonamen übergeben werden. Die Nummern beziehen sich dabei auf die Nummer des Kommandos in der History-Liste, die Zeichenketten auf das erste Vorkommen eines Kommandonamens, der mit der spezifizierten Zeichenkette beginnt.

Fehlt das Argument, so werden die letzten 16 Kommandos aus der History-Liste selektiert.

Bei Angabe eines Argumentes (Zahl oder Zeichenkette) wird das entsprechende Kommando ausgewählt. Zusammen mit dem Schalter **-l** wird damit der Beginn des Ausgabebereichs festgelegt.

Werden **fc** zwei Argumente übergeben, so wird dadurch ein Bereich aus der History-Liste bestimmt.

Mit dem Schalter **-l** können die Kommandos aufgelistet werden.

```
~/skriptum/unix[5]$ fc  -l

1       ls -l
2       cd unix
3       vi unix.tex
4       wc -l kap*.tex

~/skriptum/unix[6]$ fc  -l  3  3

3       vi unix.tex

~/skriptum/unix[7]$ fc  -l  3

3       vi unix.tex
4       wc -l kap*.tex
5       fc -l
6       fc -l 3 3

~/skriptum/unix[8]$ fc  -l  vi

3       vi unix.tex
4       wc -l kap*.tex
5       fc -l
6       fc -l 3 3
7       fc -l 3
```



```
~/skriptum/unix[9]$ fc  -l  vi  3

3       vi unix.tex

~/skriptum/unix[10]$ fc  -l  c  v

2       cd unix
3       vi unix.tex

~/skriptum/unix[11]$
```

Im Gegensatz zum direkten Editieren der Eingabezeile wird bei Verwendung des **fc**-Kommandos das selektierte Kommando (bzw. Kommandos) in den **vi** geladen. Das Kommando kann dann im **vi** bearbeitet werden, nach dem Verlassen des Editors wird es ausgeführt. Das Editieren eines größeren Kommandobereichs wird dadurch relativ gefährlich. In der Regel ist es daher besser, die Kommandos durch

fc -l *bereichs_anfang bereichs_ende* >*dateiname*

zuerst in eine Datei zu schreiben, die Modifikationen darin vorzunehmen und dann erst die Datei auszuführen.

```
~/skriptum/unix[11]$ fc  w    # Editieren von  wc -l kap*.tex

wc -c kap*.tex
     69096 kap1.tex
     64926 kap2.tex
     37100 kap3.tex
     11616 kap4.tex
     63251 kap5.tex
     36035 kap6.tex
     26525 kap7.tex
     98425 kap8.tex
     406974 total

~/skriptum/unix[12]$
```

Wird der Schalter **-s** verwendet, so führt **fc** das selektierte Kommando (bzw. Kommandos) aus.

```
~/skriptum/unix[12]$ fc  -s  3    # vi unix.tex
~/skriptum/unix[13]$
```

8.2 SHELL-Programmierung

Die Befehlssprache der SHELL ist in Wirklichkeit eine Programmiersprache, deren Syntax derjenigen vom ALGOL68 sehr ähnlich ist. Die SHELL-Programme werden in Dateien, den sogenannten **Scripts**[8], abgelegt, die dann während ihres Laufes durch die SHELL interpretiert werden. Die Scripts können sequentielle Abfolgen von Kommandos, Steuerbefehle (Verzweigungen, Schleifen), Schachtelungen und Rekursionen enthalten, für die Datenstrukturen ist allerdings nur ein Typ, nämlich Zeichenketten, erlaubt (wenn z.B. eine Variable den Wert 10 hat, so handelt es sich dabei nicht um die ganze Zahl 10, sondern um die zwei Zeichen 1 und 0).

Die Ausführung eines SHELL-Scripts kann auf unterschiedliche Art und Weise erfolgen.

1. Die einfachste Möglichkeit ist, das Script durch

   chmod +x *scriptname*

   ausführbar zu machen und es dann durch

   scriptname

   zu starten. Die SHELL überprüft, ob es sich bei *scriptname* um ein ausführbares Programm handelt (Binärdatei oder Script), falls ja, so wird mittels **fork** ein Sohnprozeß gestartet (**Subshell**), der das Programm dann ausführt.

2. Auch bei der zweiten Möglichkeit

   sh *scriptname*

   wird das Script in einer Subshell (**sh**) ausgeführt, dies entspricht dem Einschluß
( ) in runde Klammern (→ S.77):

   (*scriptname***)**

   Runde Klammern sind Trennzeichen (wie **;**) zwischen Befehlen, sollen sie innerhalb von Befehlen verwendet werden, so muß ihnen ein umgekehrter Schrägstrich vorangestellt werden.

3. Durch

   . *scriptname*

   wird das Script im Rahmen der aktuellen SHELL interpretiert, dies entspricht
{ } dem Einschluß in geschweifte Klammern:

   { *scriptname***;}**

   Geschweifte Klammern sind eigenständige (→ **find**) Ausdrücke, vor und hinter ihnen muß deshalb entweder ein Trennzeichen (wie **;**, **;;**, **( )**, **&**, | oder NEWLINE) oder ein Leerzeichen (bzw. TAB) stehen.

4. Das Kommando **exec** überschreibt (→ S.138) den aktuellen SHELL-Prozeß durch das Programm *scriptname*

   exec *scriptname*

[8] dieser Begriff wurde aus dem Englischen übernommen, SHELL-Prozeduren wäre treffender

Da es beim Programmende die SHELL nicht mehr gibt, ist eine Rückkehr in sie auch nicht mehr möglich (für **exec** muß *scriptname* ausführbar sein).

5. Ein im Hintergrund gestarteter Prozeß

   scriptname **&**

   wird ebenfalls durch eine Subshell ausgeführt. Gegenüber erstens und zweitens kann jedoch währen der Ausführung des Scripts die aktuelle SHELL weiter benutzt werden.

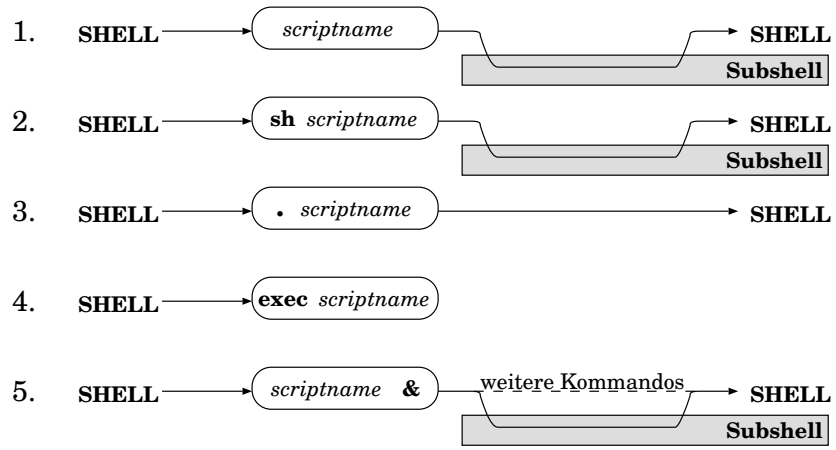


Abbildung 8.1: Ausführungsmöglichkeiten für ein Script

In der mit dem System V ausgelieferten Bourne-SHELL besteht zusätzlich die Möglichkeit, Funktionen zu definieren. Funktionen sind inhaltlich den Scripts ähnlich. Im Unterschied zu Scripts werden sie jedoch nicht auf der Festplatte sondern im Hauptspeicher abgelegt, so daß auf sie ein schnellerer Zugriff möglich ist. Ein weiterer Unterschied besteht darin, daß zu ihrer Ausführung keine Subshell gestartet wird, ihre Abarbeitung erfolgt in der aktuellen SHELL.

Die Definition einer Funktion erfolgt üblicherweise in einer der Systemdateien (z.B. in **.profile**), sie kann jedoch auch direkt in der SHELL eingegeben werden. Bei der Abmeldung vom System wird sie zusammen mit der SHELL aus dem Hauptspeicher gelöscht, sie ist folglich beim nächsten Anmelden nur dann erneut verfügbar, wenn sie in einer Datei definiert wurde, die beim Anmelden gelesen wird.

Die Syntax einer Funktionsdefinition sieht wie folgt aus:

funktionsname ()

{

kommandos

}

bzw.

funktionsname () { *kommandos*;}

Eine einmal definierte Funktion kann durch **unset -f** *funktionsname* wieder gelöscht werden.

Durch die nachfolgende Funktion soll das aktuelle Datum ausgegeben werden. Dazu wird das Kommando **date** in **umgekehrte Hochkommas** eingeschlossen. Die SHELL ersetzt dann das Kommando durch seine Standardausgabe, die **echo** anschließend auf den Bildschirm ausgibt.

‘ ‘

$ datum ()
> {
> echo Heutiges Datum: ‘date‘
> }
$ datum

```
Heutiges Datum: Thu Mar 21 11:46:02 MET 1996

$
```

Durch **return** [*n*] kann für eine Funktion ein Rückkehrstatus (Voreinstellung: Wert des letzten Kommandos in der Funktion) angegeben und die Funktion beendet werden. Die Angabe von **exit** [*n*] an beliebiger Stelle in einem Script beendet den ausführenden Prozeß mit dem Rückkehrstatus *n* (Voreinstellung: Wert des letzten Prozesses im Script). Da die Funktion im vorhergehenden Beispiel in der aktuellen SHELL läuft, beendet die Angabe von **exit** gleichzeitig auch sie.

8.2.1 Variable

Als Programmiersprache stehen der SHELL unterschiedliche Variable des Typs *zeichenkette* zur Verfügung. Sie können in frei definierbare Benutzervariable und „innere“, namentlich vordefinierte, SHELL-Variable unterteilt werden.

Benutzervariable werden durch

name=*wert*

gesetzt. Zwischen *name*, = und *wert* darf dabei kein Leerzeichen stehen.

$ person=Peter
$ echo $person

```
Peter
```

```
$
```

$** Der Wert der Variablen wird durch das Voranstellen des SHELL-Sonderzeichens **$ ausgelesen.

Die Eingabe von **echo $person** bewirkt, daß die SHELL **$person** durch den Wert der Variablen (**Peter**) ersetzt und **echo** dann **Peter** auf den Bildschirm ausgibt.

Das Kommando **echo** wurde schon des öfteren verwendet. Es tut nichts anderes, als seine zuvor durch die SHELL ausgewerteten Argumente auf den Bildschirm auszugeben. Die Auswertung durch die SHELL kann durch den Einschluß in Hochkommas oder Anführungszeichen und durch das Voranstellen eines umgekehrten Schrägstriches beeinflußt werden.

```
$ echo  \$person

$person

$ echo  "$person"

Peter

$ echo  '$person'

$person

$
```

Durch den **umgekehrten Schrägstrich** (\) wird die spezielle Bedeutung eines eventuell nachfolgenden Sonderzeichens aufgehoben (die SHELL „schluckt“ den umgekehrten Schrägstrich und interpretiert das folgende Zeichen beim Lesen nicht). ****

Die **Anführungszeichen erhalten** die spezielle Bedeutung der eingeschlossenen Sonderzeichen zur Variablen- und Befehlssubstitution (**$** und ‘ ‘), sie erlauben außerdem die Auswertung arithmetischer Ausdrücke (nur Bash). **" "**

Sollen ", **$**, ‘, NEWLINE oder \ innerhalb von Anführungszeichen explizit vorkommen, so muß ihnen ein \ vorangestellt werden[9].

Hochkommas, die weiteren Sonderzeichen der SHELL und Wortbegrenzungszeichen, werden innerhalb von Anführungszeichen wie normale Zeichen behandelt.

Sollen z.B. Zeichenketten mehrere Leerzeichen zwischen einzelnen Wörtern enthalten, so müssen sie z.B. durch Anführungszeichen eingeschlossen werden,

```
$ fred="vier Leerzeichen    dazwischen"
$ echo  $fred
```

[9] der umgekehrte Schrägstrich hat nur in Verbindung mit diesen Zeichen seine aufhebende Wirkung, ansonsten wird er wie ein ganz normales Zeichen behandelt

```
vier Leerzeichen dazwischen
```

```
$ echo  "$fred"
```

```
vier Leerzeichen    dazwischen
```

da sonst die SHELL beim Interpretieren der Eingabezeile die Zeichenkette in die einzelnen Worte zerlegt (die Wortbegrenzungszeichen sind in der vordefinierten SHELL-Variablen **IFS** (**Internal Field Separator**) als Leerzeichen, TAB und NEWLINE vordefiniert) und dabei die überflüssigen Leerzeichen eliminiert.

' ' **Hochkommas** heben die Wirkung von **allen** SHELL-Sonderzeichen auf. Hochkommas innerhalb von Hochkommas sind nicht möglich. Einen Ausweg zeigt das folgende Beispiel:

```
$ echo  'abc'\''def'
```

```
abc'def
```

Die Ausgabe von **echo** kann durch Kontrollzeichen (→ Tab.8.4) beeinflußt werden.

Kontrollzeichen	Bedeutung
\a	Alarm (Glocke)
\b	Zeichen zurück (backspace)
\c	kein abschließendes NEWLINE ausgeben, entspricht dem Schalter **-n**
\f	Zeilenvorschub (formfeed); bei manchen Bildschirmen verursacht es ein Löschen des Bildschirminhalts (entspricht dem Kommando **clear**), bei Druckern in der Regel die Ausgabe einer neuen Seite
\n	NEWLINE
\r	RETURN
\t	TAB
\0*n*	oktale Angabe eines Zeichens aus dem ASCII-Zeichensatz, *n* kann ein bis drei Ziffern haben

Tabelle 8.4: Kontrollzeichen des Kommandos **echo**

```
$ echo "kein NEWLINE am Ende\c"
kein NEWLINE am Ende$
```

Damit die SHELL den Kontrollzeichen ihre besondere Bedeutung durch das Entfernen des umgekehrten Schrägstriches nicht nehmen kann, muß diesen ein umgekehrter Schrägstrich vorangestellt oder sie müssen in Anführungszeichen bzw. Hochkommas eingeschlossen werden.

Der Name einer Variablen kann aus Buchstaben, Ziffern und dem Unterstrich (_) gebildet werden. Folgt solch ein Zeichen unmittelbar hinter dem Namen der Variablen, so ist der Name in geschweifte Klammern zu setzen.

```
$ lehrer=Alt
$ echo der ${lehrer}e kommt!

der Alte kommt!

$
```

Zum Auswerten einer SHELL-Variablen gibt es zusammen mit dem **$** und den geschweiften Klammern zusätzliche Möglichkeiten (→ Tab.8.5).

Form	**Bedeutung**
$*var*	Wert der Variablen *var*
${*var***}**	Wert der Variablen *var*; die Form wird dann verwendet, wenn unmittelbar nach *var* ein Zeichen (Buchstabe, Ziffer, _) folgt
${*var***:-***zeiket***}**	existiert *var* und ist sie nicht leer, wird der Wert von *var* ausgegeben, ansonsten *zeiket*
${*var***:=***zeiket***}**	existiert *var* und ist sie nicht leer, wird der Wert von *var* ausgegeben, ansonsten wird *var* auf *zeiket* gesetzt und der Wert ausgegeben
${*var***:?***meldung***}**	existiert *var* und ist sie nicht leer, wird der Wert von *var* ausgegeben, ansonsten wird *var***:** *meldung* ausgegeben und das aktuelle Kommando oder Script beendet; wird *meldung* weggelassen, so gibt die SHELL `parameter null or not set` aus
${*var***:+***zeiket***}**	existiert *var* und ist sie nicht leer, wird *zeiket* ausgegeben, ansonsten eine leere Zeichenkette

Tabelle 8.5: Auswertung der Variablen

Der Doppelpunkt in den Ausdrücken in Tab.8.5 ist optional. Mit dem Doppelpunkt bedeutet die Form „existiert und ist nicht leer“, ohne Doppelpunkt nur „existiert“.

```
$ otto=otto
$ echo  ${otto:-0}

otto

$ echo  ${xaver:-0}

0

$ echo  ${xaver:=0}

0

$ echo  $xaver

0

$ echo  ${liesl:?"nicht definiert"}

liesl: nicht definiert

$ echo  ${liesl:?}

liesl: parameter null or not set

$ echo  ${xaver:+1}

1

$ echo  ${liesl:+1}



$
```

SHELL-Variable sind vordefinierte Variable der SHELL. Um sie von Benutzernamen zu unterscheiden, werden sie in der Regel in Großbuchstaben geschrieben (z.B. **HOME**, **MAIL**, **PS1**). Sie können auf einen bestimmten Wert gesetzt sein, falls dies nicht der Fall ist, enthalten sie wie jede andere ungesetzte Variable die leere Zeichenkette "".

Die Belegung der SHELL-Variablen erfolgt beim Start einer SHELL durch Kopie der Werte der **Umgebungsvariablen** (**environment variables**). Diese können aus der SHELL direkt (z.B. Name der SHELL, Version, Arbeitsverzeichnis) oder aus Systemdateien (z.B. **/etc/profile**, **/etc/passwd**) stammen (z.B. Heimatverzeichnis, Benutzeridentifikation).

Der Benutzer arbeitet daher mit SHELL-Variablen, die lediglich Kopien der Umgebungsvariablen sind. Ändert ein Benutzer den Wert einer SHELL-Variablen, so ist diese Änderung in Prozessen (Subshells), die von der SHELL gestartet werden, *a priori* nicht bekannt.

Es gibt jedoch die Möglichkeit, eine Variable (Benutzer- und SHELL-Variable) durch das Kommando **export**[10] zur Umgebungsvariablen zu machen. Werden dann neue Prozesse in der SHELL gestartet, so übernehmen sie durch das Kopieren der Umgebungsvariablen automatisch auch die neu gesetzten Werte.

```
$ TERM=vt100; export TERM
$ echo $TERM

vt100

$ sh                # neue SHELL
$ echo $TERM

vt100
```

Der im Beispiel verwendete Strichpunkt (**;**) dient als Trennzeichen (wirkt wie NEWLINE), wenn mehrere Befehle in der Kommandozeile eingegeben werden sollen (s.u.). **;**

Es besteht auch die Möglichkeit, Variable in Systemdateien generell zu exportieren.

Durch die alleinige Eingabe von **export** kann überprüft werden, welche neuen bzw. geänderten Variablen exportiert werden[11].

Soll die Änderung eines Variablenwertes nur für ein einziges Kommando gelten, so hat die Zuweisung des neuen Wertes unmittelbar vor dem Aufruf des Befehls zu erfolgen.

```
$ TERM=xxx vi datei

xxx: Unknown terminal type
I don't know what kind of terminal you are on - all I have is 'xxx'.
[Using open mode]
     1
     "datei" [New file]
  1
  :q
$ echo $TERM

xterm
```

[10] nach Eingabe von **set -a** werden alle nachfolgend definierten Variablen automatisch exportiert
[11] in der Bash werden bei **export** alle exportierten Variablen ausgegeben

```
$
```

Es können mehrere Variable gemeinsam exportiert werden.

```
$ export  TEMP  PATH  MAIL
$
```

SHELL-Variable	Bedeutung
HOME	Heimatverzeichnis; wird **cd** ohne Argument eingegeben, so liest es den Pfad aus **HOME**
PATH	Menge von Verzeichnissen, in denen die SHELL nach ausführbaren Dateien sucht, falls das Befehlsargument nicht mit / beginnt (absoluter Pfad)
PS1, **PS2**	Primäre und sekundäre Aufforderung zur Eingabe (Prompt); **PS1** wird von der SHELL ausgegeben, wenn sie auf eine Eingabe wartet, **PS2** wird bei Kommandos ausgegeben, die über mehrere Zeilen gehen
IFS	**Internal Field Separator**, enthält die Wortbegrenzungszeichen (Leerzeichen, TAB, NEWLINE); IFS kann nicht an eine Subshell vererbt werden

Tabelle 8.6: Einige vordefinierte SHELL-Variable

```
$ echo  $HOME

/ws30/zeta/usr/home/leibner

$ echo  $PATH

.:/bin:/usr/bin:/usr/bin/X11:/usr/local/bin:/sbin:/usr/ucb

$
```

Die Suche nach einer ausführbaren Datei beginnt im Arbeitsverzeichnis (**.**) und wird dann von links nach rechts fortgesetzt.

```
$ IFS=x
$ echox$HOSTTYPE

alpha

$
```

Mit **set** werden alle Variablen und ihre Werte ausgegeben, mit **unset** können die meisten Variablen gelöscht werden.

```
$ unset  HOME
$ echo  $HOME
$ cd

no home directory

$
```

Die Variablen **PS1**, **PS2**, **PATH**, **MAILCHECK** und **IFS** können nicht gelöscht werden.

```
$ unset PATH

PATH: cannot unset

$
```

Da außerdem verschiedene SHELL-Variable durch **unset** ihre spezielle Bedeutung verlieren (und auch durch erneutes Setzen nicht wiedererlangen), sollte **unset** möglichst nicht verwendet werden.

Das Kommando **set** besitzt Schalter, welche unter anderem auch das Verhalten der aktuellen SHELL beeinflussen (→ Tab.8.7). Die mit den Schaltern verbundenen Optionen sind in der Regel nicht aktiviert, sie werden durch **set -***argument* gesetzt, bzw. durch **set +***argument* wieder gelöscht.

Schalter	Mnemonik	Bedeutung
-a	**allexport**	Exportieren aller nachfolgenden Definitionen
-e	**errexit**	Beenden der aktuellen SHELL, falls ein einfaches Kommando einen Rückkehrstatus ungleich 0 hat
-v	**verbose**	Ausgabe der Eingabezeile auf den Bildschirm vor der Ausführung des Kommandos
-x	**xtrace**	Ausgabe der Kommandos (nach Expansion der Sonderzeichen) auf den Bildschirm, bevor sie ausgeführt werden
--	–	falls keine Argumente folgen: Zurücksetzen der Werte der Positionsparameter der Script-Argumente; andernfalls: Setzen der angegebenen Positionsparameter auf die folgenden Werte

Tabelle 8.7: Einige Schalter des Kommandos **set**

8.2.1.1 Spezielle SHELL-Variable

Zu den speziellen SHELL-Variablen zählen Positionsparameter und Variable, deren Wert vom Benutzer nicht verändert werden kann.

Positionsparameter

Wie der Name schon sagt, enthalten die Positionsparameter die Argumente, die einem SHELL-Script beim Aufruf übergeben werden. Ihre Namen sind **1**, ..., **9**, **10**, ...

Den speziellen SHELL-Variablen kann kein Wert durch = zugewiesen werden. Eine Zuweisung ist nur mit Hilfe des Kommandos **set** (**set --** *argumente*) möglich (→ Tab.8.7).

```
$ set  --  a  b  c
$ echo  $1  $2  $3

a b c

$
```

Durch **set --** ohne Argumente, werden die Positionsparameter zurückgesetzt.

```
$ set  --
$ echo  $1  $2  $3

$
```

Das nachfolgende Script (Name: **lt**)

```
#!/bin/sh
# lt
echo  -n  'Name des Verzeichnisses: '
echo  $1
echo  -n  'Gesamtzahl der Zeichen:'
cat  $1/* | wc  -c
```

□

wird in einer Bourne-SHELL ausgeführt (**#!/bin/sh**).

```
$ lt  netze

Name des Verzeichnisses: netze
Gesamtzahl der Zeichen:    375272

$
```

Ein weiteres Beispiel zeigt die Verwendung von geschweiften Klammern innerhalb eines Scripts.

```
#!/bin/sh
# sortieren
dateiname=$1
zeilenanzahl=${2:-5}
titel=${3:+"Person          Nebenstelle\n"}
echo  "$titel"
sort  +2n  $dateiname | head  -$zeilenanzahl
```

□

Als erstes Argument muß der Dateiname angeführt werden. Folgt kein weiteres Argument, so werden die ersten fünf Zeilen der (numerisch) sortierten Datei ausgegeben. Wird die Anzahl der Zeilen als zweites Argument übergeben, so erfolgt die gewünschte Ausgabe ohne Überschrift (Voreinstellung). Soll die Überschrift ausgegeben werden, so muß als drittes Argument eine nichtleere Zeichenkette folgen (beliebig).

```
$ sortieren  maenner  3  blb

Person          Nebenstelle

Vitus Gsell             112
Fritz Hecht             210
Jan Maly                421

$
```

Werden mehr als neun Argumente dem Script übergeben, so kann auf die folgenden Parameter nicht ohne weiteres zugegriffen werden[12]. Mit dem Befehl **shift** kann jedoch das Fenster über den Parametern nach rechts verschoben werden, so daß nach einmaliger Angabe von **shift** Argument 2 zu Argument 1, Argument 3 zu Argument 2, usw.,

[12] in der Bash ist dies durch ${10}, ${11}, usw., möglich

gemacht wird, wobei Argument 1 verloren geht. Nach **echo $1** wird folglich der Inhalt von **2**, nach **echo $9** derjenige von **10**, usw., ausgegeben.

Parameter ohne Schreibrecht des Benutzers

Zu den Variablen, deren Wert auf keine Weise durch den Benutzer verändert werden kann, zählt #. Diese Variable enthält die Anzahl der Positionsparameter (nach jedem **shift** wird diese Anzahl um 1 reduziert).

```
#!/bin/sh
# count
echo Anzahl der Parameter: $#
```

□

```
$ count alpha beta gamma

Anzahl der Parameter: 3

$
```

Weitere Variable mit nur Leserechten sind * und @. Beide enthalten die Argumente des Scripts.

```
#!/bin/sh
# params
count $*
count "$*"
count $@
count "$@"
```

□

```
$ params 1 2 '3 4' 5

Anzahl der Parameter: 5
Anzahl der Parameter: 1
Anzahl der Parameter: 5
Anzahl der Parameter: 4

$
```

Der Parameter **$*** wird in alle angegebenen Positionsparameter expandiert, **count** wird folglich **1 2 3 4 5** übergeben (`Anzahl der Parameter: 5`).

Durch den Einschluß in Anführungszeichen (**"$*"**) wird **count** nur ein Ausdruck (**"1 2 3 4 5"**) übergeben (`Anzahl der Parameter: 1`).

Die Wirkung von **$@** entspricht derjenigen von **$*** (`Anzahl der Parameter: 5`).

Ein Unterschied zu **"$*"** tritt dann auf, wenn **$@** ebenfalls in Anführungszeichen eingeschlossen wird. Da **"$@"** zu **"$1" "$2"...** expandiert wird und der Inhalt von **$3** dem in Hochkommas eingeschlossenen Ausdruck **3 4** entspricht, zählt **count** in diesem Fall nur 4 Positionsparameter (`Anzahl der Parameter: 4`).

Der Parameter **0** enthält entweder den Namen der aktuellen SHELL

```
$ echo $0
sh
```

oder innerhalb eines Scripts den Namen dieses Scripts.

```
#!/bin/sh
# meinname
echo scriptname: $0
echo parameter: $*
echo anzahl der parameter: $#
```

□

```
$ meinname a b c d e
scriptname: ./meinname
parameter: a b c d e
anzahl der parameter: 5
$
```

Die Variable **0** wird nicht wie ein Positionsparameter behandelt und daher nicht in die Anzahl der Positionsparameter (#) eingerechnet.

SHELL-Variable	Bedeutung
0	Name der SHELL oder des SHELL-Scripts
#	Anzahl der Argumente eines SHELL-Scripts
*****	**$*** enthält alle Argumente der Eingabezeile, bei Einschluß in Anführungszeichen entspricht **"$*"** dem einen Argument "$1 $2..."
@	**$@** enthält alle Argumente der Eingabezeile, bei Einschluß in Anführungszeichen entspricht **"$@"** den Argumenten "$1" "$2"...
-	(Minus) alle Schalter, mit denen die SHELL aufgerufen wurde
?	Rückkehrkode des zuletzt im Vordergrund ausgeführten Befehls
$	PID der aktuellen SHELL
!	PID des zuletzt im Hintergrund gestarteten Prozesses

Tabelle 8.8: Spezielle SHELL-Variable ohne Schreibrechte

Wird ein Kommando nicht erfolgreich (Rückkehrkode ungleich 0) beendet, so kann dies vom nachfolgenden Kommando aus dem Wert der Variablen **?** abgelesen werden.

```
$ rm xx; echo $?

rm: xx: No such file or directory
2
```

Die PID der aktuellen SHELL steht in **$**.

```
$ ps

  PID TTY      S           TIME COMMAND
 1841 ttyp1    I        0:07.61 -bash (bash)
 3010 ttyp1    S  +     0:00.23 sh

$ echo $$

3010

$ (cd $HOME; pwd; echo $$)

/usr/home/leibner
3010
```

Dieser Wert ändert sich auch innerhalb einer Subshell nicht.

8.2.2 Steuerbefehle

Die Befehlssprache der SHELL bietet ähnliche Steuerbefehle wie bekannte prozedurale Programmiersprachen an.

Es sind dies die Verzweigungsbefehle **if** und **case** und die Schleifenbefehle **for**, **while** und **until**. Neuere Versionen der Bourne-SHELL (→ S.173) erlauben zusätzlich die Definition von Funktionen.

Jeder Prozeß (Programm, Script) übergibt nach seiner Beendigung demjenigen, der ihn gestartet hat, einen Rückkehrkode. Der Rückkehrkode ist eine ganze positive Zahl. Im allgemeinen wird eine 0 als erfolgreiches Beenden, eine Zahl größer 0, als nichterfolgreiches Beenden eines Prozesses verstanden. Ein einfacher Befehl übergibt in der Regel eine Zahl zwischen 0 und 127. Der Wert 128 und größer bedeutet, daß der Prozeß durch ein Signal (→ S.145) mit der Nummer n ($128 + n$) beendet wurde.

Der Rückkehrkode eines einfachen Befehls (z.B. **cd verz**) ist entweder 0 (das Verzeichnis **verz** existiert und der Wechsel dahin war erfolgreich) oder 1 (im Falle eines Mißerfolgs).

8.2.2.1 Schleifenbefehl **for**

Das Kommando **for** hat folgende Form:

for *name* [**in** *menge*]
do
    kommandos
done

bzw.

for *name* [**in** *menge*]**; do** *kommandos***; done**

Bei der Ausführung des Befehls wird zuerst *menge* in eine durch Leerzeichen voneinander getrennte Folge von Belegungen expandiert, die anschließend eine nach der anderen der Variablen *name* zugewiesen werden. Für jede Belegung der Variablen wird dann die Folge von *kommandos* ausgeführt.

Wird der optionale Teil (**in** *menge*) weggelassen, so wird die *kommandofolge* für alle Argumente des Scripts ausgeführt.

Die Schleife endet, wenn die Menge der Belegungen aus *menge* oder die der Argumente abgearbeitet wurde.

Das vorher bereits verwendete Script **lt** soll nun dahingehend modifiziert werden, daß für ein beliebiges Verzeichnis für die darin enthaltenen Dateien zusätzlich die Zugriffsrechte, der Besitzer, die Größe der Datei in Byte sowie die Art der Datei, ausgegeben werden.

```
#!/bin/sh
# lt_for
echo  -n  'Name des Verzeichnisses: '
echo  $1
dateinamen=`ls  $1`
for  i  in  $dateinamen
do
      zeile=`ls  -ld  $1/$i | cut  -c1-10,15-23,32-41`
      zeile=${zeile}`file  $1/$i`
      echo  $zeile
done
echo  -n  'Gesamtzahl der Zeichen:'
cat  $1/* | wc  -c
```

□

Es werden zuerst alle Namen der Dateien aus dem angegebenen Verzeichnis in die Variable **dateinamen** geschrieben. Dies erfolgt durch den Ersatz des Kommandos **ls $1** durch seine Standardausgabe. Der Name des Verzeichnisses ist der einzige Parameter des Scripts, er steht in der Variablen **1**.

Anschließend wird aus der Menge der Belegungen in **$dateinamen** eine nach der anderen der Variablen **i** zugewiesen und die Kommandofolge für jede abgearbeitet.

Bei jedem Schleifendurchgang wird eine Ausgabezeile generiert. Diese wird in der Variablen **zeile** abgelegt. Zuerst wird in diese Variable ein Teil (**cut**) des Textes gespeichert (die Angabe der Spaltenbereiche hängt von der Form der Ausgabe des Kommandos **ls** ab), der durch **ls** erzeugt wurde. Im nächsten Befehl wird an diese Zeichenkette das Ergebnis des Kommandos **file** angehängt. Der Inhalt von **zeile** wird dann auf die Standardausgabe ausgegeben.

$ lt_for .

```
Name des Verzeichnisses: .
-rwxr-xr-x leibner 234 ./ascript: c program text
-rwxr-xr-x leibner 40 ./count: /bin/sh shell script -- commands text
-rwxr-xr-x leibner 105 ./lt: /bin/sh shell script -- commands text
-rwxr-xr-x leibner 262 ./lt_for: /bin/sh shell script -- commands text
-rw-rw-r-- leibner 117 ./maenner: ascii text
-rwxr-xr-x leibner 79 ./meinname: /bin/sh shell script -- commands text
-rwxr-xr-x leibner 50 ./params: /bin/sh shell script -- commands text
-rwxr-xr-x leibner 191 ./sortrn: /bin/sh shell script -- commands text
Gesamtzahl der Zeichen:      1078

$
```

Übung 26 *Schreiben Sie ein Script, mit dessen Hilfe sie alle Großbuchstaben in Dateinamen zu Kleinbuchstaben machen können.*

8.2.2.2 Verzweigungsbefehl **if**, Operatoren **&&** und ||

Das Kommando **if** realisiert eine Verzweigung. Am häufigsten wird es zusammen mit dem Befehl [bzw. **test** verwendet.

Die Kommandos **test** und [(s.o.) werden zur Bildung einer Verzweigungsbedingung verwendet. Da das Kommando [häufig nur ein Verweis (hard oder symbolic link) auf die Datei **test** ist, erfüllt es die gleiche Funktion wie **test**. Im Gegensatz zu **test** muß es jedoch durch] beendet werden.

Die allgemeine Form des Kommandos **if** lautet:

if *kommandos*
then
    kommandos
[**elif** *kommandos*
then
    kommandos]...
[**else**
    kommandos]
fi

bzw.

if *kmds*; **then** *kmds*; [**elif** *kmds*; **then** *kmds*;]... [**else** *kmds*;] **fi**

Zuerst werden die Kommandos hinter **if** ausgeführt. Ist ihr Rückkehrkode 0, so werden die **then** folgenden Kommandos abgearbeitet, ist er ungleich 0, so werden nacheinander die Kommandos hinter jedem **elif** ausgeführt. Ist ihr Rückkehrkode 0, so werden die **then** folgenden Kommandos abgearbeitet und das Kommando beendet. Andernfalls werden die Kommandos hinter **else** (falls vorhanden) ausgeführt und danach **if** beendet.

Das **lt_for** soll nun so erweitert werden, daß es im Falle eines fehlenden Argumentes eine Fehlermeldung ausgibt.

```
#!/bin/sh
# lt_if
if [ $# -eq 0 ]    # bzw. if test $# -eq 0
then
        echo "Argument fehlt\nGeben Sie einen Verzeichnisnamen an!"
        exit 1
else
        lt_for $1
        exit 0
fi
```

□

Werden dem Kommando keine Argumente übergeben, so meldet es jetzt einen Fehler:

$ lt_if

```
Argument fehlt
Geben Sie einen Verzeichnisnamen an!

$
```

Mit dem nachfolgenden Script soll überprüft werden, ob ein Kommando erfolgreich beendet wurde (exit 0).

```
#!/bin/sh
# betest
if  $*
then
        echo  "(exit  $?)"
        echo  "Befehl erfolgreich beendet"
else
        echo  "(exit  $?)"
        echo  "Befehl fehlerhaft"
fi
```

□

$ betest file maenner

```
maenner:        ascii text
(exit 0)
Befehl erfolgreich beendet
```

$ betest lt_if

```
Argument fehlt
Geben Sie einen Verzeichnisnamen an!
(exit 1)
Befehl fehlerhaft
```

Übung 27 *Schreiben Sie ein kurzes Script, mit dessen Hilfe Sie überprüfen können, ob eine Datei vorhanden ist (Bildschirmausgaben nach /dev/null umleiten). Existiert die Datei, so soll deren Inhalt mit* **more** *ausgegeben werden, existiert sie nicht, so soll sie unter dem angegebenen Namen angelegt (Kommando* **touch***) und ihre Zugriffsrechte auf* 755 *gesetzt werden.*

Die Operatoren **&&** und || haben eine ähnliche Funktion wie **if**.

Der Operator **&&** stellt eine UND-Verknüpfung dar. Die allgemeine Syntax lautet: **&&**

kommando1 **&&** *kommando2*

Es wird *kommando1* ausgeführt. Ist sein Rückkehrkode 0, so wird *kommando2* ebenfalls ausgeführt.

$ test -d bin && mv maenner bin/maenner

Existiert das Verzeichnis **bin** (**test -d**), so wird die Datei **maenner** nach **bin** verschoben, existiert es nicht, so bleibt **maenner** im aktuellen Arbeitsverzeichnis.

|| Der Operator || stellt eine ODER-Verknüpfung dar. Ihre Syntax lautet:

kommando1 || *kommando2*

Es wird *kommando1* ausgeführt. Ist sein Rückkehrkode ungleich 0, so wird *kommando2* ebenfalls ausgeführt.

$ test -d bin || mkdir bin

Gibt es das Verzeichnis **bin** im aktuellen Arbeitsverzeichnis nicht, so wird es angelegt.

Beide Verknüpfungen werden durch **&**, **;** oder NEWLINE beendet. Sowohl das UND als auch das ODER haben die gleiche Priorität. Ihre Priorität ist höher als diejenige von **&** und **;**, die wiederum gleiche Priorität haben.

$ test -d bin || mkdir bin ; cp lt bin

In das Verzeichnis **bin** wird die Datei **lt** kopiert. Falls **bin** nicht existiert, so wird es zuvor angelegt.

8.2.2.3 Verzweigungsbefehl **case**

Mit Hilfe des Befehls **case** kann aufgrund einer vorgegebenen Zeichenkette eine Verzweigung realisiert werden. Die allgemeine Form des Kommandos sieht wie folgt aus:

case *zeiket* **in**
zeiket11[| *zeiket12*]...) *kommandos1***;;**
zeiket21[| *zeiket22*]...) *kommandos2***;;**

⋮

zeiketn1[| *zeiketn2*]...) *kommandosn***;;**

esac

bzw.

case *zeiket* **in** [*zeiket11*[| *zeiket12*]...) *kommandos1***;;**] ... **esac**

Das Kommando **case** expandiert zuerst *zeiket* in eine Reihenfolge von Belegungen. Danach vergleicht es jede Belegung mit den angegebenen Mustern (*zeiket11*, usw.). Es können mehrere Muster, durch | getrennt, für eine bestimmte Kommandofolge angegeben werden. Die Bildung der Muster kann mit Hilfe von Sonderzeichen (→ S.47) erfolgen.

Wird das passende Muster gefunden, so werden die dazugehörigen Kommandos ausgeführt. Danach wird die nächste Belegung aus *zeiket* auf die gleiche Weise verglichen.

Wird für eine Belegung kein Muster gefunden, so wird zur nächsten übergegangen. Stimmt keine Belegung mit einem Muster überein, so ist der Rückkehrkode des Kommandos 0, ansonsten hat er den Wert des zuletzt ausgeführten Kommandos.

Mit dem Kommando **read**

read [**-r**] [*name* ...]

können Variablen Werte aus der Eingabezeile zugewiesen werden. Das Kommando liest von der Standardeingabe eine Zeile (diese kann durch **\<return>** fortgesetzt werden) und setzt den ersten *namen* auf das erste Wort der Zeile, den zweiten auf das zweite, usw. Werden keine *namen* aufgeführt, so wird die ganze Zeile der SHELL-Variablen **REPLY** zugewiesen.

Durch den Schalter **-r** wird \ Teil der Eingabe. Dadurch können auch Dateien gelesen werden, deren Zeilen mit einem \ enden oder die z.B. ein **\n** enthalten.

Als Beispiel für die Verwendung von **case** zusammen mit **read** sei ein Script angeführt, mit dessen Hilfe im Arbeitsverzeichnis ein Verzeichnis **kopie** angelegt werden kann, in das anschließend alle Dateien kopiert werden oder das bei entsprechender Wahl alle Dateien im Arbeitsverzeichnis löscht.

```
#!/bin/sh
# wartung
case $1 in
```

```
kopie)  mkdir  kopie  2>/dev/null
        cd  kopie
        cp  ../*  .  2>/dev/null
        cd  .. ;;
 entf)  echo  -n  "Alle Datein in ‘pwd‘ loeschen [j/n] ?:"
        read  eingabe
        if  [  "$eingabe"  =  "j" ]  ||  [  "$eingabe"  =  "J"  ]
        then
          rm  *
        fi ;;
   *)   echo "Eingabe: wartung kopie | entf" ;;
 esac
```

□

Das Script **lt_if** soll nun dahingehend modifiziert werden, daß mit dem Schalter **-a** auch die Dateien, die mit einem **.** beginnen, ausgegeben werden und im Falle eines falschen Schalters ein Fehler gemeldet wird.

```
#!/bin/sh
# lt_case
if  [  $#  -eq  0  ]
then
      echo "Argument fehlt\nGeben Sie einen Verzeichnisnamen an!"
      exit  1
else
      case  $1  in
        -a)  all=-a ; shift ;;
        -?)  echo  "Schalter: [-a | ]"; exit  1 ;;
      esac
      echo  -n  ’Name des Verzeichnisses: ’
      echo  $1
      dateinamen=‘ls  $all  $1‘
      for  i  in  $dateinamen
      do
        zeile=‘ls  -ld  $1/$i | cut  -c1-10,15-23,32-41‘
        zeile=${zeile}‘file  $1/$i‘
```

```
        echo $zeile
    done
    echo -n 'Gesamtzahl der Zeichen:'
    cat $1/* | wc -c
    exit 0
fi
```

□

```
$ lt_case -x .

Schalter: [-a | ]

$
```

Das Kommando **case** führt eine Verzweigung in Abhängigkeit vom Wert des ersten Argumentes von **lt_case** durch. Wird kein Schalter angegeben, so wird keine der Alternativen von **case** durchlaufen und die Variable **all** bleibt leer (leere Zeichenkette) – **lt_case** verhält sich daraufhin wie **lt_if**.

Bei Angabe von **-a** wird **all** auf diesen Wert gesetzt, so daß mit **ls -a** auch die mit einem Punkt anfangenden Dateien in die Variable **dateinamen** geschrieben werden. Das unmittelbar darauffolgende **shift** macht den Verzeichnisnamen zum Inhalt des Positionsparameters **1**, so daß sich an den weiteren Befehlen des Scripts gegenüber **lt_for** nichts ändert.

Die Reihenfolge der angegeben Muster **endet** mit **-?**. Darin steht **?** für ein beliebiges Zeichen. Auf die Reihenfolge ist zu achten, da es sonst zu unerwünschten Effekten kommen kann.

Übung 28 *Schreiben Sie ein Script* **anent**, *daß Sie durch* `Gebe Aktion und Datei an:` *zur Eingabe einer Aktion* (**loeschen, remove, delete** *oder* **anlegen, create**) *auffordert und dann je nach Eingabe eine Datei löscht oder anlegt und dabei die Zugriffsrechte auf* 755 *setzt.*

8.2.2.4 Schleifenbefehle **while** und **until**

Das Kommando **while** führt zwei Kommandofolgen solange durch, solange der Rückkehrkode der ersten Folge 0 ist.

while *kommandos*

do

kommandos

done

bzw.

while *kommandos*; **do** *kommandos*; **done**

Das Kommando führt **while** *kommandos* durch. Bei dem Rückkehrkode 0 wird **do** *kommandos* durchgeführt, danach wieder **while** *kommandos*, solange, bis der Rückkehrkode von **while** *kommandos* ungleich 0 wird, danach wird die Schleife beendet.

Zusammen mit den Schleifenbefehlen **for**, **while** und **until** können die Befehle **break** und **continue** verwendet werden.

Das Kommando **break**[13] beendet eine Schleife, das Script wird nach **done** fortgesetzt.

```
#!/bin/sh
# abbruch
while  true
do
      echo  -n  'Gib etwas ein (RETURN = Schleifenende)'
      read  var
      test  -z  "$var"  &&  break
done
```

□

Anstelle der Kommandos hinter **while** können auch die Befehle **true** (Rückkehrkode 0) und **false** (Rückkehrkode 1) angegeben werden. Im Beispiel wird durch **while true** eine unendliche Schleife initiiert, die durch ein RETURN allein beendet werden kann (**test -z** *zeiket* ist wahr (0), wenn *zeiket* leer ist).

Das Kommando **continue**[14] setzt die Interation fort, ohne die restlichen Kommandos der Schleife auszuführen.

```
#!/bin/sh
# iteration
i=0
while  [  $i  -lt  10  ]
do
      i=`expr  $i  +  1`
      if  [  $i  -eq  5  ]; then  continue; fi
      echo  $i
done
```

□

[13] durch **break** [*n*] kann die Schleifentiefe, die beendet werden soll, spezifiziert werden

[14] durch **continue** [*n*] kann die Schleifentiefe, aus der zur nächsten Iteration übergegangen werden soll, angegeben werden

```
$ iteration

1
2
3
4
6
7
8
9
10

$
```

Der Variablen **i** wird zuerst der Wert 0 zugewiesen. Danach wird eine Schleife gestartet, in welcher der Wert von **i** sukzessive mittels **expr** (s.o.) um eins erhöht wird. Hat er den Wert 5, so wird der Test in **if** wahr und **continue** setzt die Iteration fort, ohne daß der Befehl **echo $i** ausgeführt würde.

Das Kommando **until** unterscheidet sich von **while** in der Bewertung des Rückkehrkodes der ersten Kommandofolge. Ist er ungleich 0, so wird die zweite Kommandofolge abgearbeitet, ist er 0, so wird das Kommando beendet.

until *kommandos*

do

kommandos

done

bzw.

until *kommandos*; **do** *kommandos*; **done**

Übung 29 *Schreiben Sie ein Script* **front***, das ihnen für alle angegebenen Dateien jeweils die ersten fünf Zeilen ausgibt.*

8.2.3 Das Kommando **test**

Das Kommando **test** tritt am häufigsten in den Befehlen **if**, **until** und **while** auf. Darin wird je nach Ergebnis der Auswertung der an **test** übergebenen Ausdrücke eine Verzweigung durchgeführt. Der Wert von **test** ist 0, wenn der Ausdruck erfüllt ist (**true**), im umgekehrten Fall ist der Wert von **test** eine positive ganze Zahl.

Wie bereits oben erwähnt, kann anstelle von **test** auch [verwendet werden, da [in der Regel lediglich ein Verweis auf **test** ist. Wird [verwendet, so muß es durch] beendet werden.

Der Ausdruck kann entweder unär oder binär sein. Unäre Ausdrücke liefern Informationen über Dateien und Zeichenketten, binäre dienen zum Vergleich von numerischen Ausdrücken oder Zeichenketten.

Numerische Vergleiche

An der Stelle eines numerischen Ausdrucks kann entweder eine Zahl oder der spezielle Ausdruck

-l *zeiket*

stehen, der durch die Länge von *zeiket* ersetzt wird.

In den nachfolgenden Tabellen in diesem Abschnitt wird angenommen, daß der Ausdruck erfüllt ist und **test** folglich den Rückkehrkode 0 hat.

Ausdruck			**Bedeutung**
zahl1	**-eq**	*zahl2*	$zahl1 = zahl2$
zahl1	**-ne**	*zahl2*	$zahl1 \neq zahl2$
zahl1	**-lt**	*zahl2*	$zahl1 < zahl2$
zahl1	**-le**	*zahl2*	$zahl1 \leq zahl2$
zahl1	**-gt**	*zahl2*	$zahl1 > zahl2$
zahl1	**-ge**	*zahl2*	$zahl1 \geq zahl2$

Tabelle 8.9: Binäre numerische Testausdrücke

```
$ test -l abc -gt 1 && echo ja

ja

$
```

Übung 30 *Schreiben Sie ein kleines Script* **vergleich** *zum Vergleich zweier Dateien. Sind sie gleich groß, so soll ihre gemeinsame Größe ausgegeben werden, sind sie unterschiedlich groß, so soll es der jeweilige Name der Datei und ihre Größe sein.*

Test des Typs, der Existenz und der Zugriffsrechte von Dateien

Die Attribute von Dateien können mit Hilfe von unären Operatoren getestet werden (→ Tab.8.10).

Ausdruck		Bedeutung
-d	*datei*	die *datei* existiert und ist ein Verzeichnis
-e	*datei*	die *datei* existiert
-f	*datei*	die *datei* existiert und ist eine gewöhnliche Datei
-r	*datei*	die *datei* existiert und kann gelesen werden
-s	*datei*	die *datei* existiert und ist nicht leer
-w	*datei*	die *datei* existiert und es kann in sie geschrieben werden
-x	*datei*	die *datei* existiert und ist ausführbar

Tabelle 8.10: Einige unäre Testausdrücke

Mit dem nachfolgenden Script soll getestet werden, ob die übergebenen Argumente gewöhnliche Dateien oder Verzeichnisse sind.

```
#!/bin/sh
# tfile
until test $# -eq 0
do
    if test -f $1
    then
        echo "$1: gewoehnliche Datei"
    fi
    if test -d $1
    then
        echo "$1: Verzeichnis"
    fi
    shift
done
```

□

```
$ tfile dir maenner

dir: Verzeichnis
maenner: gewoehnliche Datei

$
```

Bei der Bildung der Ausdrücke können auch die logischen Ausdrücke für die Negation (**!**), die UND-Verknüpfung (**-a**) und die ODER-Verknüpfung (**-o**) verwendet werden.

```
$ test ! -f maenner ; echo $?
```

```
1

$
```

Zur besseren Übersichtlichkeit und zur Änderung der Priorität können die einzelnen Ausdrücke in runde Klammern eingeschlossen werden. Damit diese nicht als Trennzeichen (Subshell) interpretiert werden, muß ihnen (z.B.) ein \ vorangestellt werden.

```
$ test \( -f maenner \) -a \( -d maenner \) ; echo $?

1

$
```

Bei der Bildung logischer Ausdrücke hat **-a** höhere Priorität als **-o**.

Test von Zeichenketten

Der Test von Zeichenketten kann mit Hilfe unärer und binärer Operatoren erfolgen (→ Tab.8.11).

Ausdruck	Bedeutung
-z *zeiket*	die Länge von *zeiket* ist Null
-n *zeiket*	die Länge von *zeiket* ist ungleich Null
zeiket1 = *zeiket1*	die Zeichenketten sind identisch
zeiket1 **!=** *zeiket1*	die Zeichenketten sind nicht identisch; zwischen **!** und = darf kein Leerzeichen stehen

Tabelle 8.11: Test von Zeichenketten

Mit dem folgenden Script wird das Kommando **rm -i** nachgebildet.

```
#!/bin/sh
# rmi
for i in $*
do
        echo -n "Datei $i loeschen [j/n] ?:"
        read antwort
        if test "y" = "$antwort"
        then
           rm -rf $i
           echo "$i ··· geloescht"
```

```
        else
            echo "$i ··· nicht geloescht"
        fi
    done
```

□

```
$ rmi  datei*

Datei datei loeschen [j/n] ?:y
datei ... geloescht
Datei datei1 loeschen [j/n] ?:n
datei1 ... nicht geloescht
Datei datei2 loeschen [j/n] ?:y
datei2 ... geloescht

$
```

Zum Test von Zeichenketten sind diese sinnvollerweise in Anführungszeichen zu setzen, da es sonst leicht zu Fehlern bei der Interpretation kommen kann.

```
$ zahl='    1'
$ nummer=1
$ test  $nummer  =  $zahl; echo  $?

0

$
```

Der Vergleich ergibt **true** (0), da beim Interpretieren der Variablen die überflüssigen Leerzeichen in **zahl** entfernt werden.

```
$ test  "$nummer"  =  "$zahl"; echo  $?

1
```

Jetzt werden unterschiedliche Zeichenketten verglichen, das Ergebnis ist folglich **false** (1).

```
#!/bin/sh
# arg
if  test  !  -n  $1
then  echo  "Argument fehlt"
else  echo  "Das Argument ist $1"
fi
```

□

Wird beim angeführten Script kein Argument übergeben, so ersetzt die SHELL **$1** durch eine leere Zeichenkette, wodurch das Kommando **test** unvollständig wird:

```
test ! -n
```

```
$ arg
```

```
./arg: test: argument expected
```

```
$
```

Übergibt man dem Kommando **test** das Argument in Anführungszeichen,

```
test ! -n "$1"
```

so arbeitet **arg** wie erwartet:

```
$ arg
```

```
Argument fehlt
```

```
$
```

8.2.4 Das Kommando **expr**

Syntax des Kommandos:

expr *ausdruck* ...

Das Kommando **expr** bestimmt den angegebenen *ausdruck* und gibt das Ergebnis auf die Standardausgabe aus. Die Operanden der Ausdrücke sind entweder Zahlen oder Zeichenketten. Das Kommando bestimmt den Typ das angegebenen Operanden anhand des verwendeten Operators. Sowohl Operatoren als auch Operanden sind selbständige Ausdrücke und müssen daher, z.B. durch ein Leerzeichen, voneinander getrennt werden. Manche Operatoren müssen von der Interpretation durch die SHELL ausgeschlossen werden, dies kann z.B. durch den Einschluß in Hochkommas geschehen.

```
$ expr 45 + 55 + 888 - 100
```

```
888
```

```
$
```

Bei der Berechnung des Ausdrucks werden aufgrund der verwendeten Operatoren (±) alle Operanden als Zahlen interpretiert. Ist dies nicht möglich, wird ein Fehler gemeldet.

```
$ expr  8  +  ""

expr: syntax error

$
```

Im Beispiel wird **expr** der Ausdruck **8 +** angegeben, der wegen des fehlenden zweiten Operanden unvollständig ist.

Wie bereits oben erwähnt, müssen Operatoren, die Sonderzeichen der SHELL sind, von der Interpretation durch die SHELL (z.B. durch Einschluß in Hochkommas) ausgeschlossen werden.

```
$ expr  8  '*'  345

2760

$
```

Das nachfolgende Script soll die Summe der Zahlen in der Datei **maenner** bilden.

```
$ cat maenner

Jan Maly                421
Peter Ott               567
Fritz Hecht             210
Joshua Baur             564
Lukas Ondracek          692
Vitus Gsell             112
Isidor Wolf             775
```

```
#!/bin/sh
# total
num=`wc  -l  <$1`
count=$num
total=0
while  [  $count  -gt  0  ]
```

```
do
        value=‘sed -n ${count}p $1 | tr -dc ’[0-9]’‘
        total=‘expr $total + $value‘
        count=‘expr $count - 1‘
done
echo -n ’Die Summe ist: ’
echo $total
```

□

In der Schleife wird jeweils eine Zeile der Eingabedatei bearbeitet. Diese wird mit Hilfe von **sed -n** *zeilen_nr***p** aus der Datei **$1** ausgewählt und auf die Standardausgabe ausgegeben. Mit **tr -dc ’[0-9]’** werden aus der ausgewählten Zeile alle Zeichen bis auf die Ziffern gelöscht und die übriggebliebenen Ziffern der Variablen **value** zugewiesen. Der Wert von **value** wird anschließend auf den Wert von **total** addiert.

Mit dem SHELL-Schalter **-x** können die einzelnen Aktionen der SHELL verfolgt werden.

$ sh -x total maenner

```
+ wc -l
num=         7
count=         7
total=0
+ [ 7 -gt 0 ]
+ sed -n 7p maenner
+ tr -dc [0-9]
value=775
+ expr 0 + 775
total=775
+ expr 7 - 1
count=6
+ [ 6 -gt 0 ]
+ sed -n 6p maenner
+ tr -dc [0-9]
value=112
+ expr 775 + 112
total=887
+ expr 6 - 1
count=5
+ [ 5 -gt 0 ]
+ sed -n 5p maenner
+ tr -dc [0-9]
value=692
+ expr 887 + 692
```

```
total=1579
+ expr 5 - 1
count=4
+ [ 4 -gt 0 ]
+ sed -n 4p maenner
+ tr -dc [0-9]
value=564
+ expr 1579 + 564
total=2143
+ expr 4 - 1
count=3
+ [ 3 -gt 0 ]
+ sed -n 3p maenner
+ tr -dc [0-9]
value=210
+ expr 2143 + 210
total=2353
+ expr 3 - 1
count=2
+ [ 2 -gt 0 ]
+ sed -n 2p maenner
+ tr -dc [0-9]
value=567
+ expr 2353 + 567
total=2920
+ expr 2 - 1
count=1
+ [ 1 -gt 0 ]
+ + sed -n 1p maenner
tr -dc [0-9]
value=421
+ expr 2920 + 421
total=3341
+ expr 1 - 1
count=0
+ [ 0 -gt 0 ]
+ echo -n Die Summe ist:
Die Summe ist: + echo 3341
3341

$
```

Die Zeilen, die mit einem + beginnen, kennzeichnen die Ausführung des dahinterstehenden Befehls, die restlichen die Zuweisung der Werte an die Variablen. Die Ausgabe von **sh -x** wird auf die Standardfehlerausgabe geschickt (Bildschirm). Auf den Bildschirm wird gleichzeitig auch die Standardausgabe des Scripts ausgegeben (siehe die beiden letzten Zeilen).

Ausdruck	Bedeutung
ausdr1 | *ausdr2*	ist *ausdr1* nicht 0 oder leer, wird er auf die Standardausgabe ausgegeben, andernfalls wird *ausdr2* ausgegeben
ausdr1 **&** *ausdr2*	falls keiner der Ausdrücke 0 oder leer ist, wird *ausdr1* auf die Standardausgabe ausgegeben, andernfalls wird 0 ausgegeben
ausdr1 rel_op ausdr2	für *rel_op* kann <, <=, =, !=, >= und > stehen; die beiden Ausdrücke werden miteinander verglichen: falls die Beziehung erfüllt ist, wird 1 ausgegeben, falls nicht 0; das Kommando versucht, die beiden Argumente als Zahlen zu interpretieren; falls dies gelingt, erfolgt der Vergleich numerisch, falls nicht, lexikographisch (der Vergleich **a '<' b** ist z.B. erfüllt, die Ausgabe ist daher 1)
ausdr1 arith_op ausdr2	für *arith_op* kann +, **-**, *****, **/** und **%** stehen; beide Ausdrücke werden als numerische Werte interpretiert und die Operation ausgeführt (% steht für Divisionsrest); falls dies nicht gelingt, wird ein Fehler gemeldet
ausdr **:** *reg_ausdr*	*ausdr* wird als Zeichenkette, *reg_ausdr* als regulärer Ausdruck aufgefaßt (die Syntax stimmt mit derjenigen des **vi** überein, es wird lediglich ein ^ für Zeilenanfang vorangestellt); als Ergebnis des Vergleichs wird ein ganzzahliger Wert ausgegeben, der angibt, wieviele Zeichen aus *ausdr* mit *reg_ausdr* übereinstimmen, ist der Vergleich nicht erfolgreich, wird 0 ausgegeben; ist ein Teil von *reg_ausdr* zwischen \(und \) eingeschlossen, so ist das Ergebnis des Vergleichs dieser eingeschlossene Teil, falls der Vergleich nicht erfolgreich ist, wird eine leere Zeichenkette ausgegeben

Tabelle 8.12: Bewertung der Ausdrücke von **expr**

Der Vergleich zweier Zeichenketten mit Hilfe eines regulären Ausdruckes sieht wie folgt aus:

```
$ x=abc
$ expr  $x  :  [^d-f]
1
$
```

Der reguläre Ausdruck wird aus allen Zeichen außer **d**, **e** und **f** gebildet. Sein erstes Zeichen ist folglich **a**, das mit dem ersten Zeichen des ersten Ausdrucks übereinstimmt.

```
$ expr boykott : boi

0

$
```

Der zweite Ausdruck stimmt nicht mit dem ersten überein, das Ergebnis des Vergleichs ist folglich 0.

```
$ name=/home/leibner/skriptum/unix/scripts/maenner
$ expr $name : '.*/\(.*\)' '|' $name

maenner

$
```

Der Ausdruck **.*/** enthält alle Zeichen bis zum letzten Schrägstrich, der Ausdruck in den Klammern alle Zeichen danach. Es wird folglich **maenner** ausgegeben. Enthält die Variable **name** nur den Dateinamen, so ist der Ausdruck in Klammern leer und es wird das Argument hinter | ausgegeben. Damit ist gesichert, daß **expr** in jedem Fall den Dateiname auf die Standardausgabe ausgibt.

Übung 31 *Geben Sie an, wie sie mit Hilfe von* **expr** *am einfachsten die Anzahl der Zeichen in einer Variablen feststellen können.*

Zum Abschluß dieses Kapitels sind in Tabelle 8.13 die wichtigsten Sonderzeichen der SHELL nochmals zusammengefaßt.

Zeichen	**Bedeutung**
*	beliebige Zeichenkette beliebiger Länge (auch der Länge Null) im Dateinamen
?	ein beliebiges Zeichen (genau eins) im Dateinamen
[*menge*]	genau ein Zeichen aus *menge* im Dateinamen; Intervallangaben (z.B. **[a-z]**) sind erlaubt; [**!***menge*] alle Zeichen, außer die aus *menge*
;	beendet ein Kommando, *bef1*;*bef2* hat eine sequentielle Abarbeitung der Befehle *bef1* und *bef2* zur Folge (; wirkt wie NEWLINE)
&	beendet ein Kommando und führt es im Hintergrund aus; bei der Konstruktion *bef1*&*bef2* werden beide Kommandos parallel gestartet
‘*kommando*‘	Ersatz von *kommando* durch seine Standardausgabe
(*kommando*)	Ausführung von *kommando* in einer Subshell
. *kommando*	Ausführung von *kommando* im Rahmen der aktuellen SHELL
{ *kommando*;}	Ausführung von *kommando* im Rahmen der aktuellen SHELL; die geschweiften Klammern sind eigenständige Ausdrücke, sie müssen daher durch Trennzeichen (Leerzeichen, ;, usw.) vom Rest abgeteilt werden
c	hebt die spezielle Bedeutung von *c* auf; durch \NEWLINE kann eine Zeile fortgesetzt werden
’*zeiket*’	hebt die Bedeutung aller Sonderzeichen der SHELL auf; ’ kann nicht von Hochkommas eingeschlossen werden
"*zeiket*"	hebt die Bedeutung von Sonderzeichen der SHELL auf; Ausnahmen sind **$**, ‘ und \, wobei \ nur im Zusammenhang mit \$, \‘, \", \\ und \NEWLINE wirksam ist
komd1 && *komd2*	*komd1* wird ausgeführt, ist sein Rückkehrkode 0, so wird *komd2* ausgeführt
komd1 || *komd2*	*komd1* wird ausgeführt, ist sein Rückkehrkode ungleich 0, so wird *komd2* ausgeführt
#	# folgende Zeichen werden als Kommentar verstanden
#!	#! folgende Zeichenkette wird als Name des Programms verstanden, mit dem das Script interpretiert werden soll

Tabelle 8.13: Sonderzeichen der SHELL

9 TCP/IP-Netze

So wie einst die Eisenbahn den Gütertransport, so haben Rechnernetze den Austausch von Informationen revolutioniert. Die gegenseitige Verbindung von Rechnern zu lokalen (**LAN, Local Area Network**) und von lokalen zu globalen (**WAN, Wide Area Network**) Netzen führte zu dem heute als **Internet** bezeichneten weltumspannenden Rechnernetz (Zusammenschluß von ca. $65{\cdot}10^3$ Netzen). Die zu Beginn unterschiedlichen Protokolle einzelner Hersteller[1] motivierten zur Entwicklung weltweiter Standards, unter denen sich hinsichtlich offener Systeme **OSI** und **TCP/IP** als die wichtigsten herausstellten.

- **OSI** (**Open Systems Interconnection**) der ISO (International Organization for Standards)
- **TCP/IP** (**Transmission Control Protocol/Internet Protocol**) des DoD (Department of Defense, USA), *de facto* Industriestandard, Protokoll des Internets.

Die Geschichte des Internets beginnt im Jahre 1969 als Auftragsarbeit des DoD an **ARPA** (Advanced Research Project Agency). Es entstand ein experimentelles Netz namens **ARPANET**, in dem zunächst 4 Rechner zur Übertragung von Datenpaketen verbunden waren. Im Jahre 1972 erhöhte sich die Anzahl auf bereits 20 Router und 50 Rechner. Die grundsätzliche Architektur von TCP/IP wurde dann in den Jahren 1977-79 von BBN (Bolt, Beranek und Newman, University College London) entwickelt. Im Jahre 1980 folgte ein experimenteller Betrieb von TCP/IP im ARPANET. Beteiligt war neben BBN die UCB (University of California at Berkeley), die Implementierung erfolgte unter 4.xBSD. Als Standard für das ARPANET wurde TCP/IP im Jahre 1983 übernommen. Zur gleichen Zeit erfolgte die Aufteilung des ARPANET in ein militärisches Netz namens **MILNET** und ein kommerzielles, dessen Entwicklung SUN Microsystems übernahm und förderte. Den nächsten Schritt zur Verbreitung von TCP/IP unternahm 1985-86 die NSF (National Science Foundation) mit der Zusammenschaltung von 6 Großrechenzentren zum **NSFnet**.

Ab dem Jahr 1992 nahm die Anzahl vernetzter Rechner rasant zu. Die Anzahl von 727000 Rechnern im Februar stieg bis September auf bereits 1 Million und bis Mai 1993 auf über 1.5 Millionen.

[1] z.B. DECnet von DEC, SNA (Systems Network Architecture) von IBM, XNS (Xerox Network System) von Xerox, SPX/IPX (Sequenced/Internet Packet Exchange) von Novell, AppleTalk von Apple oder VINES der Fa. Banyan

Die geschätzten Zahlen für Juli 1995 lauten: ca. 30 Millionen Benutzer, ca. 6 Millionen Rechner.

Die Beschreibung der Internetprotokolle, Empfehlungen zum Aufbau von Netzen und ähnliches, sind in einer Dokumentenserie namens **RFC** (Request for Comments)[2] zusammengefaßt und öffentlich zugänglich.

Im folgenden wird zunächst die prinzipielle Arbeitsweise der Protokolle der TCP/IP-Familie vorgestellt, anschließend wird auf die Funktion der wichtigsten Applikationsprogramme eingegangen.

9.1 Architektur der Protokolle der Familie TCP/IP

Die Implementierung des Rechnernetzes erfolgt schichtweise (→ Abb.9.1). Während der Benutzer normalerweise mit Programmen der Applikationsschicht zu tun haben wird, muß der Systemverwalter auch Kenntnisse über die Transportschicht haben. Zur Wartung der technischen Seite des Netzes muß zusätzlich die Funktionsweise der zweiten Schicht, zum Programmieren des Netzes die der dritten, bekannt sein.

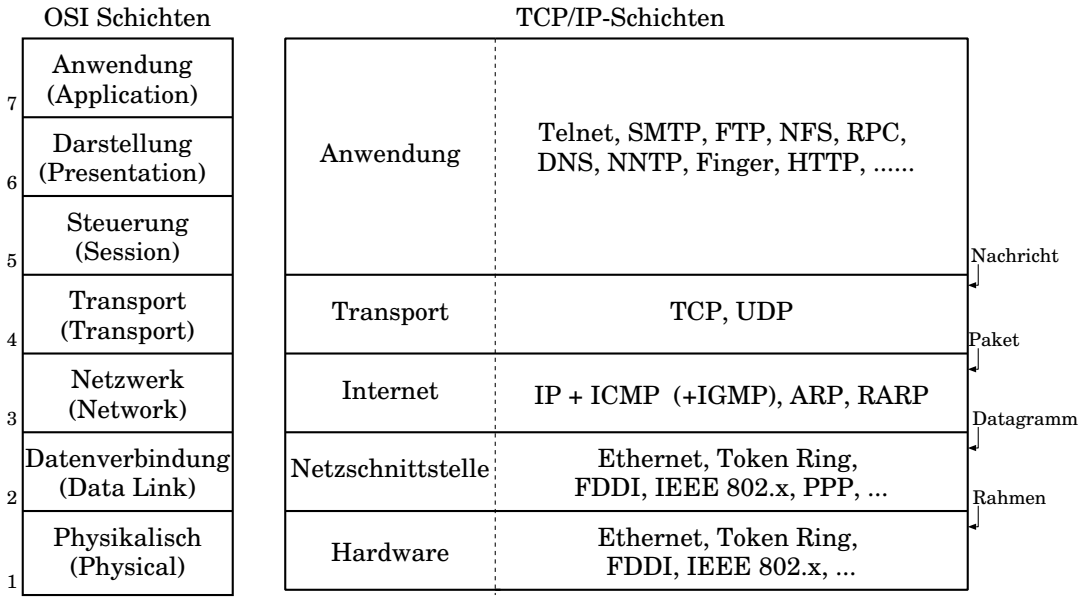


Abbildung 9.1: TCP/IP-Modell vs. Schichtenmodell nach OSI

OSI 1 Die elektrische Übertragung der Daten erfolgt auf der **physikalischen Schicht**. Die Verbindung zwischen den Rechnern kann durch unterschiedliche physikalische Medien realisiert werden. Es können dazu z.B. **Koaxialkabel**, **verdrillte Kupferkabel** (**twisted pair**) oder **Glasfaserkabel** verwendet werden. Die Realisierung eines lokalen

[2]WWW-Adresse: http://www.cis.ohio-state.edu/hypertext/information/rfc.html

Netzes (LAN) erfolgt *heute* in der Regel entweder als Ethernet/IEEE 802.3 oder als Token Ring.

Beim **Ethernet** wird abhängig von der maximalen Länge eines Kabelsegments zwischen 10Base5, 10Base2, 10Base2 Ext. und 10BaseT, unterschieden. Das längste Segment von 500m läßt 10Base5 zu, das kürzeste von 100m 10BaseT. Die Kabel können mit Hilfe sogenannter **Repeater** verlängert werden. Ein Repeater wirkt als Verstärker für die zu übertragenden Signale. Daher kann z.B. bei 10Base5 die maximale Kabellänge durch das Einfügen von maximal vier Repeatern um drei Segmente und zwei Verbindungssegmente (verbindet nur Repeater) auf 2500m verlängert werden (→ Abb.9.2).

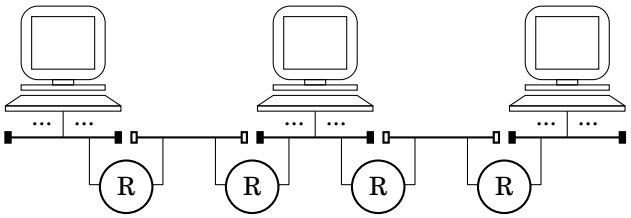


Abbildung 9.2: Maximale Anzahl von Segmenten und Repeatern bei 10Base5

Der Anschluß eines Rechners an das Kabel erfolgt im einfachsten Fall mit Hilfe eines sogenannten **T-Konnektors** sowie eines Netzadapters. Die Übertragungsrate beträgt einheitlich 10 Mbit/s.

Schickt ein Rechner ein Datenpaket in das Ethernet, so wird dieses in beide Richtungen weitergeleitet. Die Datenpakete werden von allen Rechnern ohne Unterscheidung empfangen. Die Entscheidung, ob das Datenpaket dem Rechner obliegt, trifft der Netzadapter (Transceiver) aufgrund der spezifizierten physikalischen Adresse (48 bit lang, eindeutig, wird vom Hersteller der Netzkarte festgelegt). Die physikalische Schicht des Ethernets bietet dem Sender dabei keine Informationen über den ordnungsgemäßen Empfang des Datenpakets, diesen Dienst übernimmt erst das Protokoll der dritten Schicht, ICMP.

Die Methode, mit deren Hilfe der Rechner erkennt, daß er ein Datenpaket in das Netz schicken kann, wird als **CSMA/CD** (**Carrier Sense Multiple Access with Collision Detect**) bezeichnet. Der Begriff kann als „Massenzugriff mit Erkennung einer Belegung und einer Kollision“ übersetzt werden. Es können folglich mehrere Rechner gleichzeitig versuchen auf das Netz zuzugreifen, wobei das Zugriffsrecht dann derjenige Rechner erhält, der als erster feststellt, daß das Netz nicht belegt ist. Da ein Signal im Netz mit endlicher Geschwindigkeit übertragen wird, kann es vorkommen, daß zwei Rechner, die beide das Netz als belegungsfrei erkannt haben, ihre Daten quasi gleichzeitig in das Netz schicken. Die dabei entstehende Kollision der Daten wird von beiden Rechnern erkannt (Collision Detect) und die Datenübertragung wird wiederholt.

Beim **Token Ring** werden die Rechner in einem Ring zusammengeschlossen. Die Daten werden nur in einer Richtung verschickt, die Freigabe des Netzes wird mittels eines

sogenannten **Token**s geregelt. Der Token ist ein Datenpaket, daß ständig im Ring zirkuliert. Es wird von einem Rechner nach dem anderen abgefangen und falls keine Datenübertragung stattfinden soll, an den jeweils nächsten weitergegeben. Beabsichtigt ein Rechner Daten zu übertragen, so behält er den Token. Damit gehört ihm der Ring und er kann seine Daten übertragen. Bei jeder Übertragung wird immer nur ein Datenpaket verschickt. Der Empfängerrechner verarbeitet das Datenpaket und schickt dem Sender eine Empfangsbestätigung. Kommt diese beim Sender an, so gibt dieser den Token zurück in den Ring.

Ein Vorteil des Token Rings gegenüber dem Ethernet ist das schnelle Auffinden einer Beschädigung des Übertragungsmediums und die automatische Regenerierung des gesendeten Datenpaketes an jeder Station im Ring. Ein Nachteil ist die langsamere Übertragung von Daten bei geringer Netzauslastung. Bei starkem Datenverkehr ist dagegen der Durchsatz beim Token Ring höher und es kann dabei im Gegensatz zum Ethernet zu keiner Verstopfung des Netzes kommen.

OSI 2 Auf der **Schicht der Datenverbindung** werden die Informationen aus den darüberliegenden Schichten in Rahmen (frames) zusammengefaßt (z.B. Ethernet Header, Ethernet Data). Die Verbindung benachbarter LANs erfolgt über sogenannte **Bridges**.

OSI 3 Auf der **Netzwerkschicht** arbeiten neben IP die Protokolle **ARP** (**Address Resolution Protocol**), **RARP** (**Reverse Address Resolution Protocol**), **ICMP** (**Internet Control Message Protocol**) und **IGMP** (**Internet Group Management Protocol**).

Das IP-Protokoll bietet einen unbestätigten Dienst. Es garantiert weder die sichere Zustellung eines Datagramms noch seine gegebenenfalls notwendige Wiederholung, diese wird erst vom Protokoll der darüberliegenden Transportschicht (TCP) veranlaßt.

Da das IP-Protokoll nur einen unzuverlässigen Dienst anbietet, besteht die Notwendigkeit eines weiteren Mechanismus, der Fehler- und Steuermeldungen sowohl zwischen den einzelnen **Routern**, die auf der Netzwerkschicht für die Verteilung der Pakete verantwortlich sind, als auch zwischen Rechnern und Routern überträgt. Das dafür zuständige Protokoll ist ICMP. ICMP schickt Fehlermeldung ausschließlich an den Sender des Datagramms. Unter dem Aspekt der Verschachtelung gehört die ICMP-Meldung zum Datenteil des IP-Protokolls, sie folgt unmittelbar hinter dem Kopf (Header) des IP-Datagramms.

Zum Test der Zugänglichkeit schickt ein Rechner bzw. Router mittels ICMP dem Adressaten ein „echo request“. Der angesprochene Rechner ist dann verpflichtet, ein „echo reply“ mit demselben Datenteil zurückzusenden. Üblicherweise erfolgt dieser Test mit Hilfe des Programms **ping** (**packet internet grouper**).

$ ping salomon

```
64 bytes from 146.254.1.4: icmp_seq=0 ttl=249 time=47 ms
64 bytes from 146.254.1.4: icmp_seq=1 ttl=249 time=22 ms
64 bytes from 146.254.1.4: icmp_seq=2 ttl=249 time=25 ms
```

<ctrl c>

```
----salomon PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 22/31/47 ms
```

Ist der angesprochene Rechner nicht erreichbar, so schickt der Router dem Sender die ICMP-Meldung „destination unreachable“ zurück.

$ ping majestix

```
PING majestix (200.200.200.6): 56 data bytes
```

<ctrl c>

```
----salomon PING Statistics----
2870 packets transmitted, 0 packets received, 100% packet loss

$
```

Falls die Frequenz der empfangenen Datagramme die Kapazität eines Routers überschreitet, so können sie von ihm nicht mehr verarbeitet bzw. weitergeleitet werden. In diesem Fall schickt ICMP dem Sender die Meldung „source quench“. Der Sender reagiert darauf mit einer Reduktion der Datenrate.

Hat ein Router einen Rechner detektiert, der ihn zum Weiterleiten von Datagrammen benutzen möchte, obwohl er keinen optimalen Weg zum Empfänger darstellt, so schickt er dem Sender die ICMP-Meldung „redirect“ mit der IP-Adresse des optimalen Routers und gleichzeitig das Datagramm an diesen Router. Die Rechner müssen folglich nur einen Router im lokalen Netz kennen.

Wird die im IP-Kopf des Datagramms spezifizierte „Time-to-live“ erreicht, bevor ein weiteres Fragment des Datagramms beim Router eintrifft, so schickt der Router die ICMP-Meldung „time exceeded“ an den Sender. Eine Fragmentierung eines Datagramms erfolgt immer dann, wenn die maximale Größe eines Rahmens (**MTU**, **Maximum Transmission Unit**) für ein Netz überschritten wird. Die Fragmentierung führt dann derjenige Router durch, der ein Netz mit hoher MTU mit einem mit niedriger verbindet. Ein einmal fragmentiertes Datagramm setzt seinen Weg bis zum Empfänger fragmentiert fort.

Stellt ein Router bzw. Rechner ein Problem fest, auf das es keine ICMP-Reaktion gibt, so schickt er dem Sender die ICMP-Meldung „parameter problem“ zurück.

Mit Hilfe der ICMP-Meldungen „timestamp request“ bzw. „timestamp reply“ kann zusätzlich die Übertragungszeit zwischen zwei Rechnern ermittelt werden (→ ping).

Zur Kommunikation müssen zwei Rechner ihre physikalischen Adressen kennen. Zum Umsetzen der 32 Bit langen IP-Adresse in die physikalische Netzadresse (z.B. 48 Bit beim Ethernet) innerhalb eines physikalischen Netzes dient das Protokoll ARP. Die Umsetzung der 32 IP Bit-Adressen in z.B. 48 Bit Ethernet-Adressen erfolgt mit Hilfe von Tabellen (ARP cache) mit zeitlich begrenzter Gültigkeit.

Möchte z.B. ein Rechner mit der IP-Adresse 147.228.1.10 die Ethernet-Adresse eines anderen Rechners mit der IP-Adresse 147.228.1.52 wissen, so schickt er an ihn einen Ethernet-Rahmen mit einem „ARP request“. Der Empfängerrechner akzeptiert die Anforderung und schickt ein „ARP response“ mit seiner Ethernet-Adresse an 147.228.1.10 zurück. Der Fragesteller empfängt die Antwort und trägt die Adresse in seinem „ARP cache“ ein.

Die Technik des **proxy ARP** (**promiscuous ARP**) bietet die Möglichkeit, eine IP-Adresse für verschiedene physikalische Netze zu verwenden. Die Technik besteht darin, daß ein Router auf ein „ARP request“ aus einem physikalischen Netz für einen Rechner, der in einem anderen physikalischen Netz lokalisiert ist, mit seiner physikalischen Adresse antwortet (promiscuous). Das Datagramm, das er daraufhin empfängt, leitet er dann ordnungsgemäß an den tatsächlichen Zielrechner weiter.

Das „proxy ARP“ kann nur in Netzen verwendet werden, in denen es den implementierten ARP-Protokollen nichts ausmacht, daß in ihren Cache-Tabellen unter einer physikalischen Adresse mehrere IP-Adressen eingetragen sind.

Mit dem RARP-Protokoll kann ein Rechner ohne Festplatte oder ein X-Terminal seine IP-Adresse von einem RARP-Server beim Systemstart erfragen. Der RARP-Server benötigt dazu eine Tabelle, in der die physikalischen Adressen den IP-Adressen zugeordnet sind. Die Anfrage enthält dann nur die physikalischen Adresse, die Antwort die zugehörige IP-Adresse.

Da die IP-Adresse nicht für eine volle Kommunikation in IP-Netzen ausreicht, wird anstelle des RARP-Protokolls häufiger das Protokoll **BootP** verwendet. Dieses liefert neben der IP-Adresse weitere Informationen, wie z.B. die Maske für Subnetze. Das BootP-Protokoll benutzt die Dienste von UDP. Zur Ermittlung seiner IP-Daten schickt ein Rechner seine Anforderung an die spezielle IP-Adresse 255.255.255.255[3]. Er braucht dazu weder seine eigene IP-Adresse noch die des BootP-Servers zu kennen. Existiert im Netz ein BootP-Server, so schickt dieser die angeforderte Information an den Fragesteller zurück.

Jeder Rechner in einem TCP/IP-Netz hat eine eindeutige 32 Bit lange IP-Adresse, die für die gesamte Kommunikation mit diesem Rechner verwendet wird. Hängen an einem Rechner (z.B. an einem Router) mehrere Netze, so muß für jedes Netz eine eigene IP-Adresse vorhanden sein. Die IP-Adresse spezifiziert folglich nicht den Rechner, sondern seine Netzschnittstelle.

[3] **limited broadcast**, das Paket wird nur im Rahmen eines einzigen physikalischen Netzes an alle sich in diesem Netz befindenden Rechner verschickt, es wird nicht über den Router in ein Nachbarnetz weitergeleitet

Jede IP-Adresse wird aus dem Paar **Netid** und **Hostid** gebildet. Durch die Netid wird das Netz, durch die Hostid der an diesem Netz angeschlossene Rechner, identifiziert.

Der Adreßbereich wird in drei Klassen unterteilt (→ Abb.9.3).

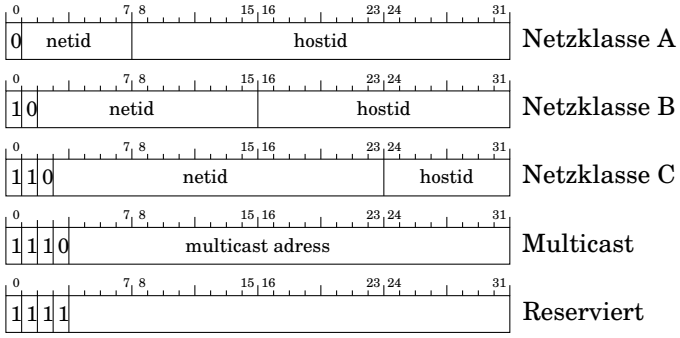


Abbildung 9.3: Formate der IP-Adressen verschiedener Netze

Individuelle Adressen

- **Netzklasse A** – viele Rechner pro Netz, wenige Netze (erstes Byte: 0-127)
- **Netzklasse B** – mittlere Anzahl von Rechnern pro Netz, mittlere Anzahl von Netzen (erstes Byte: 128-191)
- **Netzklasse C** – wenige Rechner pro Netz, viele Netze (erstes Byte: 192-223)

Gruppenadressen

- **Netzklasse D** – gleichzeitige Adressierung einer Rechnergruppe (erstes Byte: 224-239)

Reserviert

- **Netzklasse E** – reservierte Adressen (erstes Byte: 240-255)

Üblicherweise werden die IP-Adressen als vier durch einen Punkt unterteilte Dezimalzahlen (s.o.) geschrieben, also z.B. 147.228.1.10 für 10010011111001000000000100001010.

Die Verteilung der Pakete zwischen den einzelnen Netzen erfolgt durch die Router entsprechend ihrer „netid“.

Es gibt einige Adressen, deren Verwendung als Rechneradresse verboten ist. Allgemein werden die Adressen wie folgt eingeteilt:

- Netzadressen (netid,hostid=0), z.B. 147.228.0.0
- Rechneradressen (netid=0,hostid), der Rechner kennt die Netzadresse nicht, er kennt jedoch seine Adresse im Netz, z.B. 0.0.0.5
- 0.0.0.0, diese Adresse bedeutet „dieser“ Rechner in „diesem“ Netz; sie wird verwendet, solange der Rechner seine eigene IP-Adresse noch nicht kennt, aber dennoch kommunizieren muß, z.B., wenn der Systemstart von einem Server über Netz erfolgen soll
- 255.255.255.255, allgemeine Adressierung aller Rechner in einem gegebenen lokalen Netz (limited broadcast → Fußnote auf S.214)
- (netid,hostid=11...1), gesteuerte allgemeine Adressierung (directed broadcast), das Paket wird nur im Rahmen des Netzes „netid“, z.B. 222.222.222.255, verteilt

- 127.0.0.0 bzw. 127.0.0.1, Adressen zum Testen der lokalen Funktion des Rechners (loopback); Daten, die an diese Adresse geschickt werden, gehen nicht über den Netzadapter in das Netz

Die 32 Bit der individuellen Rechneradresse können in einen globalen (netid) Teil, der die Netzadresse enthält und für die Weitergabe von Paketen zwischen Netzen dient und einen lokalen (hostid) Teil, der den Rechner in einem bestimmten Netz identifiziert, eingeteilt werden.

Kommen im Netz Subnetze vor (→ Abb.9.4), so wird der lokale Teil weiter in einen Teil, der das Subnetz spezifiziert und einen weiteren, der den im Subnetz angeschlossenen Rechner adressiert, aufgeteilt. Die Größe der Adresse für das Subnetz wird durch eine **Subnetzmaske** (**subnet mask**) bestimmt. Die Adresse des Subnetzes erhält man dann aus der Konjunktion der Subnetzmaske mit der IP-Adresse.

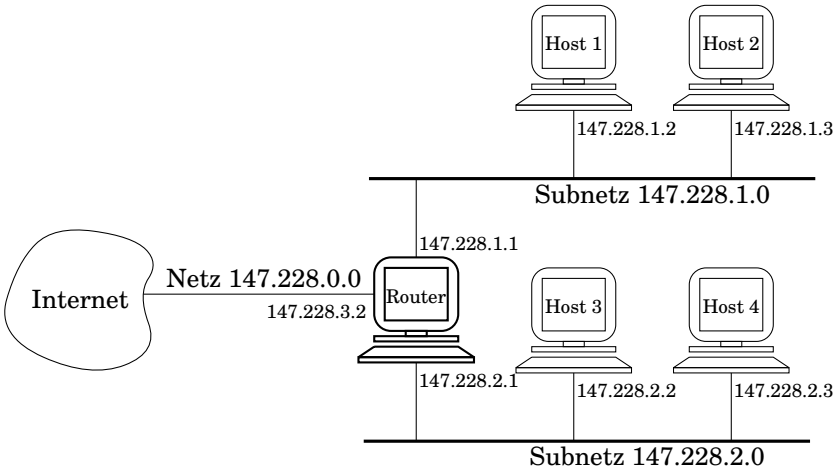


Abbildung 9.4: Beispiel zur Adressierung von Subnetzen (Maske: 255.255.255.0)

Da die Manipulation von großen Zahlen nicht besonders benutzerfreundlich ist, können für alle IP-Adressen symbolischen Namen angegeben werden. Die Abbildung der IP-Adressen auf diese Namen übernimmt das **DNS** (**Domain Name System**). Die einzige Möglichkeit, die zur Zeit ca. 6 Millionen Namen zu verwalten, ist durch ein dezentralisiertes hierarchisches System mit verteilten Zuständigkeiten.

Die Einteilung der Namensbereiche erfolgt in sogenannte Domänen. Die Benennung der einzelnen Rechner einer gegebenen Domäne erfolgt dann durch die für diese Domäne verantwortliche Organisation.

Nachfolgend (→ Tab.9.1) sind die Domänen der höchsten Stufe (**top-level domains**) zusammengefaßt.

Domäne	**Bedeutung**
com	kommerzielle Organisation in den USA
edu	Bildungseinrichtung in den USA
gov	Regierungsstelle in den USA
mil	Armeegruppe in den USA
net	Netzverwaltungszentren in den USA
org	restliche Organisationen in den USA
arpa	momentane Domäne des ARPANET (veraltet)
int	internationale Organisation
Staatskode	Staaten

Tabelle 9.1: Domänen der höchsten Hierarchiestufe

Die einzelnen Domänen können weiter unterteilt werden, als Trennzeichen dient dabei der Punkt. Der Rechner **zfe**, angeschlossen an die Subdomäne **siemens.de** der Siemens AG in Deutschland (Staatskode **de**), hat z.B. den vollständigen hierarchischen Namen **zfe.siemens.de**.

Die Abbildung der hierarchischen Domänennamen der Rechner auf ihre korrespondierenden IP-Adressen übernimmt ein verteiltes System von unabhängigen zusammenarbeitenden Rechnern, den sogenannten **Nameservern** (**name server**) (→ Abb.9.5).

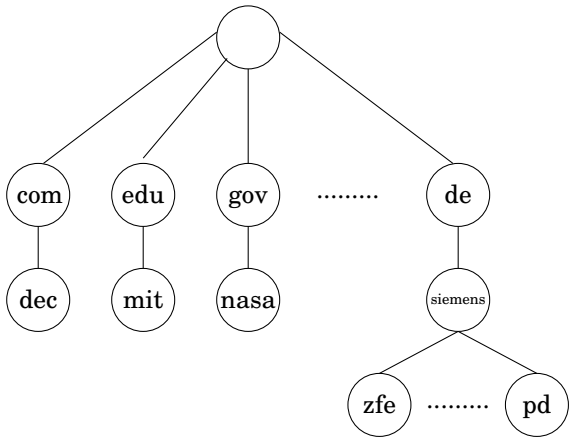


Abbildung 9.5: Ausschnitt aus dem hierarchischen Domänenbaum des Internets

Für jede Domäne existieren aus Sicherheitsgründen zwei Nameserver, die sowohl die Adressen ihrer Subdomänen als auch die des Nameservers der übergeordneten Schicht kennen.

OSI 4 Auf der **Transportschicht** gibt es zwei Protokolle, TCP und UDP. Beide setzen auf das IP-Protokoll der dritten Schicht auf. TCP wird von der Anwendungsschicht für eine Duplexverbindung zur Übertragung von Daten benutzt, UDP für Anwendungen ohne feste Verbindung (Datagrammorientiert).

Die Duplexverbindung bei TCP und der nichtverbundene Dienst von UDP erfolgen zwischen einem Paar von **Sockets**[4]. Unter einem Socket versteht man das geordnete Paar (**IP-Adresse**, **Port**). Er bestimmt eindeutig einen Prozeß (bzw. Dienst) im Internet. Der Port stellt dabei eine ganze Zahl dar, die von den Protokollen der Transportschicht zur Unterscheidung der Zielprozesse verwendet wird. Für Standarddienste, wie z.B. die elektronische Post (e-mail), gibt es reservierte Ports (**well-known ports**)[5]. Die IP-Adresse bestimmt folglich einen bestimmten Rechner im Netz, die Nummer des Ports die Applikation, an welche sich die Nachricht richtet.

An der Eröffnung einer Verbindung sind beide Partner beteiligt. Der eine, in der Regel der Server, führt eine passive Eröffnung durch (er wartet und ist bereit, die Anforderung einer aktiven Eröffnung zu akzeptieren), der Client versucht eine aktive Eröffnung.

Die Protokolle TCP/IP enthalten keinen Mechanismus zur Bildung von Prozessen. Es ist daher notwendig, daß auf Servern, die einen bestimmten Dienst anbieten, zum Zeitpunkt der Anforderung bereits ein entsprechender Prozeß läuft[6].

Es ergibt sich daher folgende Aufgabenteilung:

- **Client** – eine Applikation, die eine Kommunikation initiiert (üblicherweise eine Anforderung),
- **Server** – eine Applikation, die eine Kommunikation erwartet um danach eine bestimmte Aktion zu starten (üblicherweise eine Antwort auf eine Anforderung)

Clients, die nur reservierte Ports verwenden, werden Standard-Clients genannt (z.B. für e-mail, Telnet, FTP und ähnliche Dienste). Nach der Realisierung einer Verbindung kann vom reservierten Port auf einen beliebigen Port umgeschaltet werden, so daß der reservierte Port für den nächsten Standard-Client, der über ihn eine Kommunikation aufbauen will, frei wird.

TCP garantiert die fehlerfreie Übertragung von Paketen im Netz. Es bildet dazu aus den Nachrichten der übergeordneten Prozesse Segmente, die dann üblicherweise als IP-Datagramm durch das Internet wandern. Die Segmente werden aus einer endlichen Anzahl von Bytes gebildet, die im gegebenen Augenblick unter einem variablen Fenster (**sliding window**) liegen. Jede Empfangsbestätigung enthält zusätzlich eine Information über die Anzahl von Bytes, die der Empfänger bereit ist zu akzeptieren. Der Sender paßt daraufhin die Größe seines Fensters vor dem Aussenden des nächsten Segments dieser Angabe an.

Die Robustheit und Sicherheit gegenüber Datenverlust bzw. Duplizierung von Segmenten wird durch die Numerierung der Reihenfolge der Segmente, durch die Lenkung des

[4] am weitesten verbreitete Schnittstellen sind das **socket interface** (**sockets**) der UCB für BSD-Unix und **TLI** (**Transport Layer Interface**) von AT&T für System V

[5] Bereich 0-1023

[6] z.B. wird die Anforderung einer Telnet-Verbindung durch den Daemon **inetd** (Port 23), der beim Systemstart aktiviert wird, entgegengenommen und von **inetd** dann das eigentliche Programm für die Kommunikation, **telnetd**, gestartet

Datenflusses (**flow control**) und durch die Aussendung von drei Segmenten (**three-way handshake**) zur Bestätigung einer Übertragung, erreicht.

Das Protokoll **UDP** (**User Datagram Protocol**) bietet einen nichtverbundenen unzuverlässigen Dienst an. Pakete können daher verloren gehen, dupliziert werden, verspätet oder in anderer Reihenfolge, ankommen. Der Vorteil eines nichtverbundenen Dienstes ist eine schnellere Antwortszeit und die Möglichkeit, mehrere Adressaten gleichzeitig anzusprechen (broadcast, multicast). Applikationen, die in zuverlässigen lokalen Netzen laufen, funktionieren in der Regel problemlos, sie müssen sich dann aber selbst um die Vermeidung von Fehlern kümmern.

Außer für Dienste, die UDP explizit verlangen, wird daher in der Regel TCP verwendet.

Die Zuordnung der IP-Adressen zu den symbolischen Rechnernamen erfolgt in der Datei **/etc/hosts**.

$ cat /etc/hosts

```
255.255.255.0    defaultmask
127.0.0.1        localhost
#
210.200.200.1    admin      node_4c899                # WS30-630
210.200.200.2    asterix    node_4c841   node_3a20a   # WS30-630
210.200.200.3    obelix     node_4c83a   node_3a1d0   # WS30-630
210.200.200.4    klapptnix  node_4abe0                # WS30-630
210.200.200.5    kleopatra  node_519c2                # WS30-625
210.200.200.6    isis       node_530a4                # WS30-625
210.200.200.7    osiris     node_53134                # WS30-625
210.200.200.8    zeta       zeta_e1                   # DEC-Alpha
210.200.200.100  sigma      linux1                    # linux-pc
#
# Anbindung Standordnetz
#
180.1.153.5      leibner    zeta_e2
146.254.1.4      salomon  news  wwwproxy  socks

$
```

Für die symbolischen Namen können mehrere Synonyme angegeben werden. Die folgenden Angaben sind daher gleichwertig:

```
ping  146.254.1.4
ping  salomon
ping  news
ping  wwwproxy
ping  socks
```

Die Vergabe von IP-Adressen erfolgt zur Zeit in den USA durch InterNIC[7] (Network

[7] mailserv@rs.internic.net

Information Center), in Asien und dem pazifischen Raum durch APNIC[8] (Asian Pacific NIC) und in Europa und Afrika durch

RIPE Network Coordination Centre[9]

Kruislaan 409

NL-1048 SJ Amsterdam

The Netherlands

9.2 Applikationen

Die **Steuerungsschicht** stellt die Schnittstelle zwischen dem Benutzer und dem Netz dar. Da die Schichten 2-4 im Systemkern realisiert sind, kann auf sie nur über Systembefehle zugegriffen werden. OSI 5

Die Kodierung und Komprimierung von zu übertragenden Daten erfolgt auf der **Darstellungsschicht**. OSI 6

Die höchste Schicht ist die **Anwendungsschicht**. Auf dieser Schicht arbeitet der Benutzer. Die ihm dabei zur Verfügung stehenden wichtigsten Programme werden im folgenden beschrieben. OSI 7

9.2.1 Telnet

Das Protokoll Telnet bietet die Möglichkeit, sich an einem entfernten Rechner im Netz zur **interaktiven** Arbeit anzumelden. Telnet arbeitet dabei im Client-Server-Betrieb. Der Client errichtet eine Verbindung zum Server und schickt ihm die einzelnen Zeichen so, wie sie auf seiner Tastatur eingegeben werden. Der Server verarbeitet die ankommenden Zeichen, als ob die Eingabe an seiner Tastatur erfolgt wäre und schickt sie zur Ausgabe zurück an den Client, der sie dann lokal darstellt. Dieser Dienst ist transparent, da der Benutzer auf der Seite des Clients den Eindruck gewinnt, er arbeite direkt an dem entfernten Serverrechner.

Neben diesem interaktiven Betrieb gibt es bei Telnet noch einen **Befehlsmodus**. Das Umschalten aus dem interaktiven in den Befehlsmodus erfolgt meistens durch **<ctrl**]>. Alle Eingaben im Befehlsmodus dienen zur Steuerung des Programms Telnet selbst.

Telnet kann daher auf zweierlei Art und Weise aufgerufen werden.

$ telnet kleopatra

[8] info@apnic.net
[9] ncc@ripe.net

```
Trying 210.200.200.5...
Connected to kleopatra.
Escape character is '^]'.

Domain/OS sr10 (kleopatra)

login: pelcad
Password:

DomainOS Release 10.4 (bsd4.3) Apollo 425t kleopatra
Apollo Domain/OS  Version SR10.4

%
```

Es kann dabei der symbolische Name (**kleopatra**) oder die IP-Adresse des Rechners (**210.200.200.5**) verwendet werden Eine Verbindung wird nur dann hergestellt, wenn der Client für den Server eine Zugangsberechtigung hat (Benutzer, Paßwort).

Die zweite Möglichkeit startet Telnet

```
$ telnet
telnet> quit
$
```

Das Programm gibt den Prompt `telnet>` aus und wartet auf weitere Eingaben (durch **quit** kann es z.B. wieder beendet werden).

Mit dem Telnetkommando **open** wird eine Verbindung zu einem Server hergestellt, mit **close** geschlossen und mit **quit** die Verbindung geschlossen und Telnet beendet.

```
$ telnet
telnet> open  zeta

Trying 210.200.200.8...
Connected to zeta.
Escape character is '^]'.

OSF/1 (zeta.zfe.siemens.de) (ttyp3)

login: leibner
Password:

DEC OSF/1 V3.0 (Rev. 347); Mon Nov 28 16:11:37 MET 1994
DEC OSF/1 V3.0 Worksystem Software (Rev. 347)
```

```
% <ctrl ]>    # %  ist der Prompt einer C-SHELL
telnet> close
Connection closed.
telnet> quit
$
```

Wie bereits erwähnt, erfolgt die Ausgabe der Zeichen beim Client erst, wenn die Zeichen vom Server verarbeitet und zurückgeschickt wurden. Ist die Übertragungszeit zwischen Client und Server lang, so kann es zwischen der Ein- und Ausgabe zu Verzögerungen kommen (dies dürfte bei heutigen Netzen kaum mehr der Fall sein). Diese Verzögerungen können durch Umschalten in den Zeilenmodus vermieden werden. Im Zeilenmodus wird nicht zeichenweise sondern zeilenweise übertragen (nach Eingabe von **<return>**). Die einzelnen Zeichen der Zeile werden zuvor lokal durch Telnet dargestellt, so daß es zu keiner Verzögerung bei ihrer Bildschirmausgabe kommt.

Der Zeilenmodus wird durch

```
telnet> mode  line
```

und die zeichenweise Darstellung durch

```
telnet> mode  character
```

eingestellt. Neben den unterschiedlichen Möglichkeiten einen Betriebsmodus einzustellen, gibt es noch folgende weitere Telnetkommandos:

```
$ telnet
telnet> ?

Commands may be abbreviated.  Commands are:

close           close current connection
display         display operating parameters
mode            try to enter line-by-line or character-at-a-time mode
open            connect to a site
quit            exit telnet
send            transmit special characters ('send ?' for more)
set             set operating parameters ('set ?' for more)
unset           unset operating parameters ('unset ?' for more)
status          print status information
toggle          toggle operating parameters ('toggle ?' for more)
slc             change state of special characters ('slc ?' for more)
z               suspend telnet
```

```
!               invoke a subshell
environ         change environment variables ('environ ?' for more)
?               print help information

telnet> status

No connection.
Escape character is '^]'.

telnet> send  ?

ao              Send Telnet Abort output
ayt             Send Telnet 'Are You There'
brk             Send Telnet Break
ec              Send Telnet Erase Character
el              Send Telnet Erase Line
escape          Send current escape character
ga              Send Telnet 'Go Ahead' sequence
ip              Send Telnet Interrupt Process
nop             Send Telnet 'No operation'
eor             Send Telnet 'End of Record'
abort           Send Telnet 'Abort Process'
susp            Send Telnet 'Suspend Process'
eof             Send Telnet End of File Character
synch           Perform Telnet 'Synch operation'
getstatus       Send request for STATUS
?               Display send options

telnet>
```

Ist man mit einem Server verbunden und kann z.B. eine erfolgte Eingabe durch BACKSPACE, DELETE, etc., nicht löschen, so ist dies nach Eingabe von **<ctrl**]> durch

```
telnet> send  el
```

möglich. Wurde während einer Verbindung durch Eingabe von **<ctrl**]> in den Befehlsmodus umgeschaltet, so wird durch

```
telnet> <return>
<return>
%
```

die Verbindung zum Server wieder hergestellt.

Eine bestehende Verbindung kann man nicht nur am Client mittels **quit** sondern auch am Server durch

```
% <ctrl d>
```



```
Connection closed by foreign host.

$
```

jederzeit beenden.

Da Telnet durch ein **virtuelles Terminal** (**NVT**, **Network Virtual Terminal**) eine Standardschnittstelle zum TCP/IP-Netz definiert, können über Telnet Rechner mit unterschiedlichen Betriebssystemen miteinander kommunizieren (→ Abb.9.6).

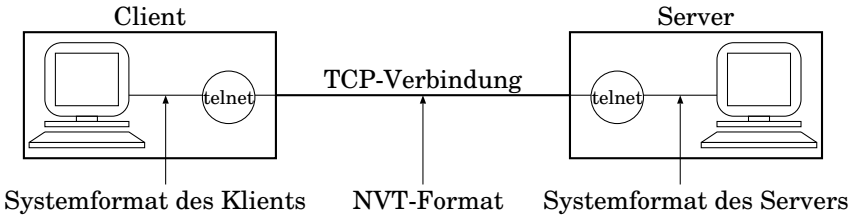


Abbildung 9.6: Prinzip des NVT-Formats

Nachfolgend (→ Tab.9.2) sind einige wichtige Telnetkommandos zusammengefaßt. Die durch † gekennzeichneten Kommandos können nur in einer C-, Korn-, TC- oder Bash-SHELL ausgeführt werden.

Kommando	**Bedeutung**
`telnet>` **open** *rechnername*	Verbindung zum entfernten Rechner herstellen
<ctrl]>	Verbindung zum entfernten Rechner unterbrechen, Rückkehr zum Telnet-Prompt
`telnet>` **close**	Verbindung zum entfernten Rechner schließen
`telnet>` **z**	†Verbindung zum entfernten Rechner unterbrechen, im lokalen System weiterarbeiten
fg (foreground)	†die Arbeit mit Telnet wieder aufnehmen (nach `telnet>` **z**)
`telnet>` **quit**	Verbindung zum entfernten Rechner schließen und Telnet beenden
`telnet>` **?**	Telnetkommandos auflisten
`telnet>` **help** *kommando*	Beschreibung des Telnetkommandos *kommando* aufrufen

Tabelle 9.2: Nützliche Telnetkommandos

9.2.1.1 RLOGIN (UCB)

Verwenden sowohl Client als auch Server UNIX, so kann alternativ zu Telnet das Programm **rlogin** (**remote login**) verwendet werden. RLOGIN stellt eine Verbindung zu einer SHELL auf dem Server her, so daß man den Eindruck gewinnt, man arbeite auf dem lokalen Rechner, ohne daß dazu Telnetkommandos bekannt sein müßten. Durch

$ rlogin sigma -l pel

```
Password:
Last login: Thu Jan 18 15:31:44 from zeta
Linux 1.2.12. (POSIX).

pel@sigma:/home/pel > cd beispiele
pel@sigma:/home/pel/beispiele > ~.

rlogin: closed connection.

$
```

wird eine Verbindung zu einer SHELL auf dem Rechner mit dem Namen **sigma** (der Prompt **PS1** kann für die Bash-SHELL durch **PS1='\u@\h:‘pwd‘ > '** auf das oben gezeigte Verhalten in der Datei **.bashrc** (**bash runtime commands**) eingestellt werden) für den Benutzer namens **pel** (**-l pel**) hergestellt. Für die SHELL-Variable TERM (Typ des Terminals) wird dabei der Wert der lokalen SHELL eingestellt. Durch **~.** bzw. **<ctrl d>** wird die Verbindung wieder geschlossen.

Wird der **rlogin** unter einem **X-Window System**[10] durchgeführt, so muß zuvor durch

$ xhost sigma

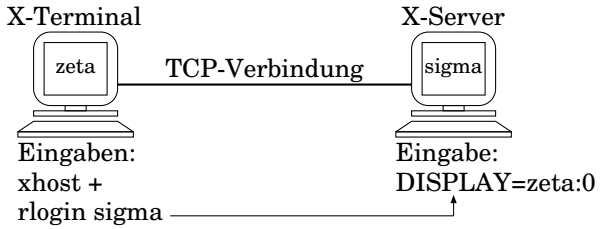
dem entfernten Rechner, z.B. mit dem Namen **sigma**, die Darstellung auf dem lokalen X-Terminal erlaubt werden. Durch

$ xhost +

wird allen Rechnern die Darstellung lokal erlaubt. Nach dem Anmelden durch **rlogin** muß am X-Server die SHELL-Variable DISPLAY gesetzt werden (→ Abb.9.7).

```
pel@sigma:/home/pel > DISPLAY=zeta:0  # CSH: setenv DISPLAY zeta:0
pel@sigma:/home/pel >
```

Für den üblichen Rechner mit nur einer Tastatur und einem Bildschirm wird die sogenannte Displaynummer auf 0 (**:0**) eingestellt (→ on-line-manual[11]).



[10] Graphikoberfläche, konzipiert beim MIT (Massachusetts Institute of Technology) im Jahre 1984 (X-Window System ist Handelsname des MIT), ab 1988 Weiterentwicklung durch das MIT-X-Consortium (≈ 70 Mitglieder); momentane Version: X-Window System, Version 11, Release 6 (X11R6)

[11] **man X**

Abbildung 9.7: Verwendung eines X-Terminals

Ohne Angabe eines Benutzernamens (→ Abb.9.7) erfolgt die Anmeldung am Server unter dem gleichen Namen wie beim Client.

Wie bei Telnet ist auch beim RLOGIN die Angabe des Paßwortes für die Benutzerkennung am entfernten Rechner erforderlich.

Es gibt jedoch die Möglichkeit, diese Abfrage zu unterbinden. Dazu muß im Heimatverzeichnis des Benutzers am entfernten Rechner eine Datei **.rhosts** existieren, in welcher der symbolische Name des lokalen Rechners sowie der Name des lokalen Benutzers stehen.

Stimmen lokaler und entfernter Benutzername überein, so reicht die Angabe von

$ rlogin sigma

um sich ohne die üblichen Formalitäten (Benutzername, Paßwort) am Rechner **sigma** anzumelden, ansonsten muß lediglich der entfernte Benutzername, nicht jedoch sein Paßwort, eingegeben werden. Durch den Eintrag in die Datei **.rhosts** wird somit die Äquivalenz zwischen dem lokalen Benutzer und demjenigen am entfernten Rechner hergestellt.

Soll die Äquivalenz für ein ganzes System gelten, so ist der lokale Rechner samt Benutzer in die Datei **/etc/hosts.equiv** einzutragen. Der lokale Benutzer hat dann Zugang zum Benutzerkonto des gleichen Namens auf dem entfernten Rechner. Wird nur der Name des lokalen Rechners in **/etc/hosts.equiv** eingetragen, so erhalten alle Benutzer des lokalen Rechners Zugang zu allen Benutzerkonten gleichen Namens auf dem entfernten Rechner. Ein Rechner, dessen lokale Benutzer sich ohne Angabe eines Paßwortes am entfernten Rechner anmelden können, wird als vertrauenswürdig (**trusted**) bezeichnet.

Wird beim Aufruf von **rlogin** ein unbekannter Benutzername oder ein falsches Paßwort angegeben, so wird der Benutzer zur Angabe eines anderen Benutzernamens aufgefordert.

$ rlogin kleopatra -l ddd

```
Password:
Login incorrect
login: ddd
Password:
Login incorrect
```

```
login: <ctrl d> rlogin: connection closed.
$
```

Durch **<ctrl d>** kann die wiederholte Aufforderung beendet werden.

Voraussetzung für ein RLOGIN ist, daß auf dem entfernten UNIX-Rechner der Daemon **rlogind** läuft.

9.2.2 FTP, TFTP

Das Programm **ftp** (**file transfer protocol**) dient zur Übertragung von Dateien nach dem Prinzip Client-Server. Der Client muß sich zuvor beim Server unter Angabe eines Benutzernamens und Paßwortes anmelden. Im Unterschied zu Telnet befindet man sich jedoch nach dem Anmelden logisch auf der Seite des Clients (lokaler Rechner), nicht auf der Seite des Servers (entfernter Rechner).

Eine Besonderheit stellt **Anonymous FTP** dar, hier wird als Benutzername **anonymous** angegeben, ein Paßwort ist nicht erforderlich (in der Regel wird die eigene e-mail Adresse als Paßwort angegeben).

Die Übertragung von Dateien kann interaktiv gesteuert werden (es wird dazu im Hintergrund Telnet verwendet). Mit Hilfe dieser Steuerbefehle ist es z.B. möglich, Dateien zwischen Rechnern im Netz zu kopieren, Verzeichnisse auf entfernten Rechnern einzurichten, Dateimanipulationen in ihnen durchzuführen und Makros zu definieren, welche es erlauben, den ansonsten interaktiven Ablauf der Datenübertragung in Scripts zu verwenden.

Wie Telnet, so funktioniert auch FTP zwischen unterschiedlichen Betriebssystemen. Es fragt dabei immer nach einem Benutzernamen und Paßwort, auch wenn beide, z.B. bei MS-DOS, nicht existieren. In diesem Fall kann man auf die Anforderung eine beliebige Antwort geben, die Verbindung wird auf jeden Fall hergestellt.

Die Übertragung von Dateien kann binär oder ASCII erfolgen.

Bei der ASCII-Übertragung werden die unterschiedlichen Angaben für das Zeilenende automatisch dem jeweiligen Betriebssystem angepaßt. So wird z.B. das **<nl>** einer UNIX-Darstellung bei Übertragung in ein DOS-System nach **<cr><nl>** (carriage return, new line) konvertiert und umgekehrt. Dieser Übertragungsmodus ist nur für ASCII-Dateien geeignet, da bei binären Dateien (Programm, Verzeichnis) eine Konversion Fehler erzeugen würde.

Der Aufruf von FTP entspricht demjenigen von Telnet.

$ ftp zeta


```
Connected to zeta.
220 zeta.zfe.siemens.de FTP server (OSF/1 Version 5.60) ready.

Name (zeta:leibner): bfs

331 Password required for bfs.
Password:
230 User bfs logged in.
Remote system type is UNIX.
Using binary mode to transfer files.

ftp>
```

Als Benutzername auf dem entfernten Rechner namens **zeta** wird der lokale Name vorgeschlagen (`zeta:leibner`). Nach Eingabe eines anderen Benutzernamens (**bfs**) wird nach dem Paßwort gefragt und anschließend die Verbindung hergestellt (Prompt `ftp>`).

In den Befehlsmodus von FTP gerät man, wenn kein Rechnername angegeben wird.

```
$ ftp
ftp> open  zeta

Connected to zeta.
220 zeta.zfe.siemens.de FTP server (OSF/1 Version 5.60) ready.

Name (zeta:leibner): <return>

331 Password required for leibner.
Password:
230 User leibner logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
```

Nach Eingabe des Befehls **open** wird die Verbindung zum angegebenen Rechner **zeta** unter dem gleichen Benutzernamen wie am lokalen Rechner hergestellt.

Im folgenden (→ Tab.9.3) sind die wichtigsten FTP-Kommandos zusammengefaßt. Die Angaben in den eckigen Klammern sind optional. Ein den Namen vorangestelltes *l* steht für *local* (lokal), ein *r* für *remote* (entfernt).

Kommando	Bedeutung
?	FTP-Kommandos ausgeben
help *kommando*	Beschreibung des FTP-Kommandos *kommando* aufrufen
!*kommando*	Ausführung des Befehls *kommando* durch eine lokale SHELL
!	Start einer lokalen SHELL (Beendigung durch **<ctrl d>**)
open *rrechnername* [*port*]	Anmelden beim entfernten Rechner *rrechnername*; falls eine Portnummer spezifiziert wird, erfolgt die Anmeldung an diesem Port; wird der Name des entfernten Rechners *rrechnername* **ftp** als Parameter übergeben, wird **open** als erstes FTP-Kommando ausgeführt
user *name* [*passwort*]	Änderung des Benutzers am entfernten Rechner
disconnect, **close**	Verbindung zum entfernten Rechner schließen
quit, **bye**	Verbindung zum entfernten Rechner schließen und **ftp** beenden
ls [*rverz*[*ldatei*]]	Ausgabe des Inhalts des entfernten Verzeichnisses *rverz*, ohne Angabe von *rverz* wird das aktuelle Arbeitsverzeichnis auf dem entfernten Rechner ausgegeben; wird keine lokale Datei *ldatei* angegeben, so erfolgt die Ausgabe lokal auf dem Bildschirm
dir [*rverz*[*ldatei*]]	wie **ls**, jedoch ausführlicher (**ls -l**)
get *rdatei* [*ldatei*]	Übertragung der entfernten Datei *rdatei* in das lokale System; ohne Angabe eines lokalen Namens *ldatei* wird der Name der entfernten Datei verwendet; mit Hilfe der Befehle **case**, **ntrans** und **nmap** kann der Dateiname bei der Übertragung der Datei transformiert werden

Tabelle 9.3: Nützliche `ftp>`-Kommandos (Teil 1)

Kommando	Bedeutung
mget *rdateien*	Übertragung entfernter Dateien *rdateien* in das lokale System, Sonderzeichen der SHELL (*,?,[,]) sind zur Namensangabe erlaubt; mit Hilfe der Befehle **case**, **ntrans** und **nmap** können die Dateinamen bei der Übertragung der Dateien transformiert werden
put *ldatei* [*rdatei*]	Übertragung der lokalen Datei *ldatei* in das entfernte System; ohne Angabe eines entfernten Namens *rdatei* wird der Name der lokalen Datei verwendet; mit Hilfe der Befehle **ntrans** und **nmap** kann der Dateiname bei der Übertragung der Datei transformiert werden
mput *ldateien*	Übertragung lokaler Dateien *ldateien* in das entfernte System, Sonderzeichen der SHELL (*,?,[,]) sind zur Namensangabe erlaubt; mit Hilfe der Befehle **ntrans** und **nmap** können die Dateinamen bei der Übertragung der Dateien transformiert werden
append *ldatei* [*rdatei*]	Anhängen des Dateiinhalts der lokalen Datei *ldatei* an eine entfernte Datei; wird *rdatei* nicht angegeben, so wird der Inhalt an eine entfernte Datei gleichen Namens angehängt
cd *rverz*	Verzeichniswechsel nach *rverz* am entfernten Rechner
cdup	Verzeichniswechsel nach **..** am entfernten Rechner
lcd *lverz*	Verzeichniswechsel nach *lverz* am lokalen Rechner
delete *rdatei*	Löschen der Datei *rdatei* am entfernten Rechner
mdelete *rdateien*	Löschen der Dateien *rdateien* am entfernten Rechner, Sonderzeichen der SHELL (*,?,[,]) sind zur Namensangabe erlaubt
mkdir *rverz*	Anlegen eines entfernten Verzeichnisses
rmdir *rverz*	Löschen eines entfernten Verzeichnisses
ascii	stellt ASCII-Übertragungsmodus ein
binary	stellt binären Übertragungsmodus ein
type [**ascii/binary**]	ohne Parameter wird der aktuelle Übertragungsmodus ausgegeben, ansonsten entspricht die Angabe von **type binary** der von **binary** bzw. von **type ascii** der von **ascii**
glob on/off	Erlaubnis/Verbot der Transformation von Namen bei der Verwendung von **mput**, **mget** und **mdelete**
nmap [*rmuster*] [*lmuster*]	Definition der Transformation zwischen den Dateinamen bei **mget**, **mput** und bei **get** und **put**, falls bei ihnen kein Zielname spezifiziert wurde; geeignet insbesondere bei der Übertragung zwischen unterschiedlichen Betriebssystemen

Tabelle 9.3: Nützliche `ftp>`-Kommandos (Teil 2)

Kommando	Bedeutung
ntrans [*izeiket* [*ozeiket*]]	Übersetzung der Zeichen in den Dateinamen, ähnlich wie **tr** (translate); ohne Angabe der Argumente erfolgt keine Übersetzung, ansonsten wird das n-te Zeichen aus *izeiket* durch das n-te aus *ozeiket* ersetzt; falls in *ozeiket* weniger Zeichen als in *izeiket* angegeben sind, werden die überflüssigen Zeichen in *izeiket* gelöscht; die Übersetzung erfolgt bei den Befehlen **mget**, **mput** und bei **get** und **put**, falls bei ihnen kein Zielname spezifiziert wurde
case on/off	Erlaubnis/Verbot der Konversion von Großbuchstaben beim Namen der entfernten Datei in Kleinbuchstaben bei der Übertragung in das lokale System (**get** und **mget**)
hash	nach jeden übertragenen 1024 Byte wird auf dem Bildschirm ein # ausgegeben
status	gibt die aktuelle Einstellung der FTP-Parameter auf dem Bildschirm aus
prompt	unterdrückt die Bestätigung einer Übertragung bei **mget** und **mput** für jede zu übertragende Datei; wiederholte Eingabe von **prompt** schaltet die Abfrage ein/aus

Tabelle 9.3: Nützliche `ftp>`-Kommandos (Teil 3)

In den folgenden Beispielen wird angenommen, daß zwischen dem lokalen Rechner und dem entfernten Rechner namens **zeta** bereits eine FTP-Verbindung besteht (s.o.).

Die Übergabe von Kommandos an die lokale SHELL erfolgt mit Hilfe von **!**.

```
ftp> !pwd

/usr/home/leibner/verz

ftp> !ls

tel-1  tel-2  verz

ftp>
```

Dies Ausgabe des Verzeichnisinhalts am entfernten Rechner erfolgt durch **ls**:

```
ftp> ls

200 PORT command successful.
150 Opening ASCII mode connection for file list (210.200.200.8,1178)
mist
vim30bin.zip
226 Transfer complete.
```

Zur Übertragung einer Datei aus dem entfernten Rechner in den lokalen dient **get**.

```
ftp> get  mist  sein_mist

200 PORT command successful.
150 Opening BINARY mode connection for mist (210.200.200.8,1179).
226 Transfer complete.
57 bytes received in 0.00098 seconds (57 Kbytes/s)

ftp> !ls

sein_mist  tel-1  tel-2  verz

ftp>
```

Die Übertragung einer Datei zu einem entfernten Rechner erfolgt mit **put**.

```
ftp> put  tel-1

200 PORT command successful.
150 Opening BINARY mode data connection for tel-1 (210.200.200.8,1185).
226 Transfer complete.
170 bytes sent in 0.026 seconds (6.3 Kbytes/s)

ftp> ls

RT command successful.
150 Opening ASCII mode connection for file list (210.200.200.8,1186).
mist
vim30bin.zip
tel-1
226 Transfer complete.

ftp>
```

Das lokale Verzeichnis kann durch **lcd** neu eingestellt werden.

```
ftp> lcd  verz

Local directory now /usr/home/leibner/verz/verz

ftp>
```

Im entfernten Rechner erfolgt der Wechsel in ein neues Verzeichnis durch **cd**.

```
ftp> cd  ..
```

```
250 CWD command successful.

ftp> pwd

257 "/usr/home/bfs/bfs" is current directory.
```

Bei der Übertragung von Dateien kann die Interpunktion in den Dateinamen geändert werden. Zu diesem Zweck dient das Kommando **nmpa**. Soll z.B. der entfernte Dateiname **fp-221.zip** lokal **fp221:zip** heißen, so ist wie folgt vorzugehen:

```
ftp> nmap  $1-$2.$3  $1$2:$3
ftp> get  fp-221.zip

local: fp221:zip remote: fp-221.zip
200 PORT command successful.
150 Opening BINARY mode connection for fp-221.zip (210.200.200.8,1037)
226 Transfer complete.
611778 bytes received in 0.46 seconds (1.3e+03 Kbytes/s)

ftp>
```

Die Übersetzung einzelner Zeichen erfolgt durch **ntrans**.

```
ftp> ntrans  aeiou  AEIOU
ftp> get  mist

local: mIst: remote: mist
200 PORT command successful.
150 Opening BINARY mode connection for mist (210.200.200.8,1040)
226 Transfer complete.
57 bytes received in 0.002 seconds (29 Kbytes/s)

ftp>
```

Mit dem Befehl **proxy** kann eine Verbindung zwischen dem primären entfernten Rechner (im Beispiel **zeta**) und einem weiteren sekundären (z.B. mit dem Namen **sigma**) hergestellt werden.

```
ftp> proxy  open  sigma

Connected to sigma.
220 sigma FTP server (Version wu-2.4(1)) ready.

Name (sigma:leibner): pel
```

```
331 Password required for pel.
Password:
230 User pel logged in.
Remote system type is UNIX.
Using binary mode to transfer files.

ftp>
```

Bei der Übertragung verhalten sich die Kommandos zum Holen und Schicken von Dateien anders als in Tab.9.3 angegeben. So wird z.B. durch **proxy put** eine Datei des

sekundären entfernten Rechners auf den primären übertragen (→ on-line-manual[12]), durch **proxy get** eine Datei vom primären auf den sekundären.

```
ftp> proxy  put  arab.tex  arab_beisp.tex

sigma:227 Entering Passive Mode (210,200,200,100,4,40)
zeta:200 Type set to I.
zeta:200 PORT command successful.
zeta:150 Opening BINARY mode connection for arab_beisp.tex
sigma:150 Opening BINARY mode connection for arab.tex
sigma:226 Transfer complete.
zeta:226 Transfer complete.
local: arab.tex remote: arab_beisp.tex

ftp>
```

Geschlossen wird die Verbindung zwischen dem primären und sekundären entfernten Rechner durch **proxy close**.

```
ftp> proxy  close

sigma:221 Goodbye.

ftp>
```

Außer im interaktiven Betrieb kann **ftp** z.B. in einem Script verwendet oder mit Hilfe von **at** zu einer bestimmten Zeit (z.B. nachts) gestartet werden. Dazu müssen die abzuarbeitende Befehlsfolge und das Makro **init** in eine Datei **.netrc**, die im Heimatverzeichnis des lokalen Rechners liegen muß, eingetragen werden. Die Datei kann mehrere Rechnernamen enthalten. Das Programm **ftp** sucht dann die entsprechende Befehlsfolge anhand des spezifizierten Rechnernamens aus.

Die Zeile, die den entfernten Rechner, Benutzer und Paßwort definiert, lautet

machine *rechnername* **login** *benutzername* **password** *passwort*

dabei ist das Paßwort unkodiert anzugeben. Damit niemand das Paßwort lesen kann, muß die Datei **.netrc**, z.B. durch **chmod 600 .netrc**, gegen unbefugtes Lesen geschützt werden.

Die eben angegebene Zeile hat nach Eingabe von **ftp** *rechnername* zur Folge, daß eine Verbindung zum entfernten Rechner ohne Angabe des Benutzers und Paßwortes erzeugt wird. Die FTP-Kommandos müssen jedoch weiterhin interaktiv eingegeben werden.

[12] **man ftp**

Die automatische Ausführung von FTP-Kommandos wird durch ein spezielles Makro namens **init** erreicht, das ebenfalls in der Datei **.netrc** definiert sein muß.

Die vollständige Angabe in der Datei **.netrc** lautet somit:

machine *rechnername* **login** *benutzername* **password** *passwort* **macdef init**
FTP-Kommandos
⋮

Als Beispiel soll eine Datei **merke** zum entfernten Rechner namens **kleopatra** dem Benutzer **dtp** mit dem Paßwort **123xyz** übertragen werden. Die Datei **.netrc** sieht dann wie folgt aus:

$ cat .netrc

```
machine kleopatra login dtp password 123xyz macdef init
put merke
quit


$
```

Hinter dem Befehl **quit** muß mindestens eine Leerzeile folgen!

Die Übertragung soll um 15^{05}h erfolgen.

$ at 15:05
ftp kleopatra
<ctrl d>

```
job leibner.822492300.a at Wed Jan 24 15:05:00 1996
$
```

Gibt das Programm **at** Meldungen aus (stderr, stdout), so werden diese dem Benutzer via e-mail übermittelt. Läuft die Übertragung fehlerfrei, so wird keine e-mail erzeugt.

Das **TFTP** (**Trivial File Transfer Protocol**) ist das einfachste Protokoll zur Datenübertragung. Damit findet es Platz im ROM (Read Only Memory) von disklosen Stationen oder X-Terminals, bei denen es zur Übertragung von Systemkode bei der Initialisierung vom Server verwendet wird.

Die Übertragung erfolgt mit Hilfe von UDP in Blöcken fester Länge von 512 Byte. Der Server wartet dabei auf die Bestätigung durch den Client, bevor er den nächsten Block losschickt. Der Name des Servers wird durch **connect** *rechnername* definiert, eine Verbindung wird nur während der Übertragung hergestellt. Die Kommandos **put** und **get** (sowie andere) haben die gleiche Wirkung wie bei FTP.

9.2.2.1 RCP, RSH (UCB)

Ein weiteres Programm, das wie RLOGIN nur zwischen zwei UNIX-Rechnern funktioniert, ist **rcp** (**remote copy**). Es wurde so geschrieben, daß sich seine Arbeitsweise von derjenigen von **cp** nicht unterscheidet. Damit es wie **cp** funktionieren kann, darf keine Abfrage nach einem Benutzer und seinem Paßwort erfolgen. Das Kommando erfordert daher die Einstellung der Äquivalenz des entfernten Benutzerkontos mit dem lokalen in **.rhosts** oder **/etc/hosts.equiv** (s.o.).

Auf einzelnen Rechnern können Sicherheitsvorkehrungen verhindern, daß das Kommando **rcp** trotz eingestellter Äquivalenz zwischen den Benutzerkonten, funktioniert. Probleme treten auch dann auf, wenn auf dem entfernten Rechner bei einer C-SHELL die Datei **.cshrc** und daraus wiederum Befehle (z.B. **echo**) ausgeführt werden.

Um Dateien von einem entfernten Rechner zu kopieren, gibt es mehrere Möglichkeiten.

Sind der entfernte und der lokale Benutzer dieselben, so ist z.B.

```
$ rcp   goya:notizen   teil1
$
```

einzugeben. Dies kopiert die Datei **notizen** aus dem Heimatverzeichnis des Benutzers am entfernten Rechner **goya** auf die Datei **teil1** des gleichen Benutzers in seinem aktuellen Arbeitsverzeichnis auf dem lokalen Rechner.

Beim Kopieren werden die Pfade immer relativ zum lokalen und entfernten Heimatverzeichnis angegeben.

```
$ rcp   ubik:gnu/rcs/README   tmp/gnu_README
$
```

Damit wird die Datei **README** aus dem Unterverzeichnis **~/gnu/rcs** am entfernten Rechner in das Unterverzeichnis **~/tmp** auf dem lokalen Rechner kopiert.

Um eine Datei von einem entfernten Rechner, an dem der lokale Benutzer über die Datei **.rhosts** eine Äquivalenz zu dem Benutzerkonto **joe** eingestellt hat, zu kopieren, ist wie folgt zu verfahren:

```
$ rcp   joe@pollux:News/alt.html   tmp/hypertext
$
```

Auch das Kopieren vom lokalen Rechner zu einem entfernten ist möglich.

```
$ rcp   parta   pollux:part1
```

```
$
```

Dies kopiert die Datei **parta** vom lokalen Rechner auf den Rechner **pollux** unter den Namen **part1**. Der lokale und entfernte Benutzername sind dabei die gleichen.

Es kann auch zwischen zwei entfernten Rechnern kopiert werden.

```
$ rcp   festival:README   jamie@scorpio:README
$
```

Dies kopiert die Datei **README** vom Rechner **festival**, auf dem der Benutzername mit dem lokalen Benutzernamen übereinstimmt, auf den Rechner **scorpio** in das Heimatverzeichnis des Benutzers **jamie**.

Die besondere Bedeutung von Sonderzeichen, die an einem entfernten Rechner expandiert werden sollen, muß lokal durch `\`, `""` oder `''` aufgehoben werden.

Mit **rsh** (**remote shell**) können Kommandos an eine SHELL auf einem entfernten Rechner übergeben werden (**rsh** wird auch von **rcp** verwendet).

```
$ rsh   kleopatra   rm   /usr/tmpusr/dat
$
```

Dadurch wird z.B. die Datei **/usr/tmpusr/dat** auf dem Rechner **kleopatra** gelöscht.

9.3 Elektronische Post

Die **elektronische Post** (**e-mail**) wurde von Ray Tomlinson 1972 entwickelt. Sie stellt sicher einen der wichtigsten Dienste im Internet dar. Sie basiert auf dem Protokoll **SMTP** (**Simple Mail Transfer Protocol**), welches die Zustellung der Post mittels direkter TCP-Verbindung zwischen dem Absender und dem Empfänger, garantiert. Treten dabei Übertragungsprobleme auf, so wird der Zustellungsversuch periodisch wiederholt. Kann die Post nach einer bestimmten Zeit (z.B. 3 Tage) nicht zugestellt werden, so wird sie dem Absender als unzustellbar zurückgesendet.

Über den Postverkehr werden Logbücher geführt. Die Postaktivitäten vom 30. Januar findet man z.B. (systemabhängig) in

/usr/var/adm/syslog.dated/30-Jan-13:42/mail.log

Neben **mail.log** stehen in diesem Verzeichnis noch weitere Logbücher (z.B. **syslog.log**).

Das allgemeine Adressierungsformat im Internet ist

benutzername@*rechnername* (z.B. **Peter.Leibner@zfe.siemens.de**).

Der lokale Anteil *benutzername* (**Peter.Leibner**) gibt in der Regel den richtigen Namen des Benutzers an, der dann im Domänenrechner *rechnername* (**zfe.siemens.de**) entsprechend einem Eintrag in einer **Alias**-Tabelle auf den Benutzernamen (**leibner**) umgesetzt wird.

Soll die Post an einen Benutzer geschickt werden, der am gleichen Rechner arbeitet, wie der Absender, so reicht die Angabe seines Benutzernamens (z.B. **leibner**), arbeitet er an einem anderen Rechner im lokalen Netz, so ist der lokale Rechnername hinzuzufügen (z.B. **leibner@zeta**).

Da die elektronische Post auch zwischen unterschiedlichen Netzen ausgetauscht werden kann, muß unter Umständen die Adressierung an diese Netze angepaßt werden. Soll z.B. die Post an einen Rechner im Netz **BITNET** gesendet werden, so kann die Adressierung z.B.

otto%csearn.bitnet@earn.cvut.cz

lauten. Der Rechner (**Mail-Gateway**), der die Verbindung zwischen TCP/IP-Netz und dem BITNET herstellt (**earn.cvut.cz**), konvertiert dann den Benutzernamen

von

otto%csearn.bitnet

nach

otto@csearn.bitnet

und schickt die Post an den Rechner **csearn** im BITNET.

Oft reicht es, wenn der Absender **otto@csearn.bitnet** eingibt. Die Verwendung solch einer „Pseudodomäne“ entbindet den Benutzer von der Kenntnis der Adresse eines Mail-Gateways, das Hinzufügen der Gateway-Adresse erfolgt dann am lokalen Rechner anhand des angegebenen Netznamens automatisch.

Weitere Netzbetreiber mit internen Adressen sind z.B. America Online, Applelink, BIX, CompuServe, Delphi, JANET, Prodigy, usw.

Der Zugang eines Benutzers zur elektronischen Post erfolgt über ein Mailprogramm (**MUA**, **Mail User Agent**). Für UNIX-Rechner werden neben herstellereigenen Pro-

grammen[13] die Standardprogramme **mailx/Mail**, **mail/binmail**, **elm** oder z.B. **pine**[14], verwendet.

Stellvertretend für die angeführten Mailprogramme wird im folgenden **mailx** genauer vorgestellt.

9.3.1 Verschicken elektronischer Post mit **mailx**

Das Verschicken einer Post sieht im einfachsten Fall wie folgt aus:

```
$ mailx bfs@pd.siemens.de
Subject: Testmail
zum Test der Adresse.
.
EOT
$
```

Nach Eingabe des Kommandos **mailx bfs@pd.siemens.de** erfolgt zuerst eine Abfrage nach dem Betreff (`Subject:`). Diese Zeile ist zwar optional, es ist jedoch immer sinnvoll, die Nachricht zu charakterisieren. Nach dem Betreff folgt der eigentliche Text. Zum Abschicken der Post wird in der letzten Zeile ein Punkt (**.**) oder **<ctrl d>** eingegeben. Soll die Post nicht abschickt werden, so ist zweimal **<ctrl c>** einzugeben.

```
$ mailx bfs@pd.siemens.de
Subject: Zweiter Test
dieser wird aber nicht abgeschickt! <ctrl c>
(Interrupt -- one more to kill letter)
<ctrl c>
$
```

Beim Schreiben der Post wird der Benutzer durch spezielle Mailkommandos mit vorangestellter Tilde unterstützt (→ Tab.9.4). Zwei dieser Kommandos führen ebenfalls zu einem Abbruch des Absendevorgangs.

Das Kommando **~x** entspricht der zweimaligen Eingabe von **<ctrl c>** – **mailx** wird beendet und die Nachricht verworfen.

[13] z.B., DEC: **dxmail**, Novell NetWare: Pegasus-Mail (**pmail**), usw.

[14] **P**ine **I**s **N**ot **E**lm, Handelsname der University of Washington, unterstützt **MIME** (**Multipurpose Internet Mail Extensions**), das eine eventuell notwendige Kodierung und Dekodierung von binären Multimediadaten (Bilder, Programme) selbständig durchführt (**uuencode** und **uudecode** sind dazu nicht notwendig), das Programm kann von **ftp.cac.washington.edu/pine/pine.tar.Z** mittels FTP heruntergeladen werden

Bei der Eingabe von **~q** geschieht das gleiche, nur wird vorher der Text der Nachricht in einer Datei **dead.letter**, die im Heimatverzeichnis angelegt wird, gesichert. Es wird immer nur eine Nachricht aufgehoben. Wird **~q** erneut verwendet, so wird der Inhalt der Datei **dead.letter** überschrieben.

Durch **~d** kann die Nachricht aus der Datei **dead.letter** in die aktuelle Post eingelesen werden.

```
$ mailx   bfs@pd.siemens.de
Subject:  Test
testtest
~q

(Last Interrupt -- letter saved in dead.letter)

$ mailx   bfs@pd.siemens.de
Subject:
~d

"/ws30/zeta/usr/home/leibner/dead.letter" 1/6

~p

-------
Message contains:
To: bfs@pd.siemens.de

testtest
(continue)

~x
$
```

Nach Eingabe von **~p** wird die aktuelle Nachricht auf dem Bildschirm ausgegeben.

Kommando	Bedeutung
~b *adressen*	fügt die Benutzeradressen *adressen* (*adr1, adr2,...*) dem Verzeichnis der Empfänger einer Kopie der Nachricht hinzu; die Adressaten der Nachricht werden davon nicht unterrichtet (**Bcc**, **Blind carbon copy**)
~c *adressen*	wie **~b**, die Adressaten werden jedoch über die weiteren Empfänger einer Kopie im Feld **Cc** (**Carbon copy**) informiert
~d	fügt den Text aus **$HOME/dead.letter** in den Text der Nachricht ein
~p	gibt den bisher geschriebenen Text der Nachricht auf dem Bildschirm aus
~q	schreibt die Nachricht nach **$HOME/dead.letter** und bricht **mailx** ab
~r *dateiname*	fügt die Datei *dateiname* in den Text der Nachricht ein
~s *zeiket*	ändert den Betreff (Subject:) in *zeiket*
~t *adressen*	fügt weitere Empfänger der Nachricht im Feld **To** hinzu (*adr1, adr2,...*), diese können auch direkt beim Aufruf von **mailx** (*adr1 adr2 ...*) angegeben werden
~v	ruft den **vi**-Editor zur Bearbeitung des bisher geschriebenen Textes auf
~w *dateiname*	schreibt den Text der Nachricht nach *dateiname*
~x	verwirft die Nachricht und bricht **mailx** ab
~:*kommando*	führt das Mailkommando *kommando* aus; *kommando* können alle Kommandos sein, die keine Tilde vorangestellt haben (→ Tab.9.5)
~!*kommando*	übergibt *kommando* zur Ausführung an die SHELL
~?	gibt alle Mailkommandos des Absendemodus aus

Tabelle 9.4: Kommandos im Sendemodus

Da **mailx** von der Standardeingabe liest, kann der Text durch Umlenkung der Standardeingabe auch aus einer Datei gelesen werden. Wird zusätzlich der Schalter **-s** angegeben, so entfällt die interaktive Eingabe des Betreffs und das Kommando kann so z.B. in einem Script verwendet werden.

```
$ mailx -s "Weiterer Test" bfs@pd.siemens.de <dat
```

9.3.2 Empfang elektronischer Post mit **mailx**

Wurde dem Benutzer während seiner Abwesenheit Post zugestellt, so erhält er nach dem Anmelden am System die Nachricht

```
You have mail
```

Erhält er Post, während er am System arbeitet, so wird er davon normalerweise nicht verständigt. Ob ein Verständigen erfolgt oder nicht, kann durch

```
$ biff

is n

$
```

abgefragt werden (die Antwort ist **n**, also nein). Durch

```
$ biff  y
$
```

wird das Programm **biff**[15] veranlaßt, beim Ankommen einer Post eine Mitteilung darüber an den Benutzer zu schicken.

Soll die Benachrichtigung generell erfolgen, so ist **biff y** in die Datei **.login** oder **.profile** einzutragen.

Unter X-Window gibt es eine Variante des Programms **biff** namens **xbiff**. Wird es gestartet, so erscheint ein Fenster mit einem amerikanischen Briefkasten. Gibt es keine Post, so zeigt die Fahne nach unten, kommt Post an, so geht sie nach oben und der Briefkasten piepst. Besteht der Wunsch, generell über ankommende Post informiert zu werden, so kann z.B.

```
xbiff -geometry 40x50+0-0 -name /usr/spool/mail/leibner&
```

in die Datei **.xsession** eingetragen werden. Der Pfad **/usr/spool/mail/** gibt an, wo sich der Briefkasten des Systems befindet. In diesem Briefkasten wird die ankommende Post für alle Benutzer gesammelt. Durch die Eingabe von

```
$ mailx
```

kann der Benutzer seine Post aus diesem Briefkasten abholen.

Wird eine Nachricht nach dem Lesen mit **d** gelöscht und das Programm **mailx** mit **q** verlassen, so wird sie im Systembriefkasten ebenfalls gelöscht. Sie ist dann nicht mehr länger zugänglich. Wird die Nachricht gelesen aber nicht gelöscht, so wird sie nach der

[15] Biff war der Name des Hundes von Heidi Stettner, die im Sommer 1980 an der UCB graduierte; zu dieser Zeit arbeitete Sie zusammen mit anderen an der BSD-Version, in der die virtuelle Adressierung eingeführt wurde; John K. Foderero schrieb zur gleichen Zeit ein Programm, das den Benutzer über ankommende Post informieren sollte; da Biff den Postboten sehr gern anbellte und damit ankommende Post ankündigte, gab Foderero seinem Programm den Namen **biff**; für seine rege Teilnahme an allen universitären Aktivitäten bekam Biff später einen Ph.Dog-Grad verliehen :-)

Eingabe von **q** in dem Briefkasten des Benutzers gespeichert. Dieser liegt in seinem Heimatverzeichnis und hat den Namen **mbox**.

Sie ist dann nach der Eingabe von

mailx -f

zusammen mit allen anderen gespeicherten Nachrichten wieder zugänglich. Mit **d** und anschließendem **q** kann sie auch aus dem lokalen Briefkasten gelöscht werden.

$ mailx

```
Mail $Revision: 4.2.4.2 $  Type ? for help.
"/usr/spool/mail/bfs": 2 messages 2 new
>N  1 Peter.Leibner@zf  Fri Feb  2 10:00  15/590  "Testmail"
 N  2 Peter.Leibner@zf  Fri Feb  2 10:01  15/597  "Weiterer Test"
 ?

$
```

Alle Nachrichten sind durchnumeriert, die **aktive Nachricht** ist durch > gekennzeichnet. Wenn bei den eingegebenen Kommandos keine spezielle Nachricht durch Angabe ihrer Nummer ausgewählt wird, so beziehen sie sich immer auf die aktive Nachricht.

Jede Zeile enthält folgende Angaben:

- **Status** der Nachricht:
  - **N** – Neue Nachricht.
  - **U** – Ungelesene Nachricht. Wird eine neue Nachricht nicht gelesen und **mailx** mit **q** verlassen, so wird sie nicht in die Datei **mbox** geschrieben, sondern erscheint beim nächsten Aufruf von **mailx** mit dem Status **U**.
  - ***** – Eine Nachricht, die in einer Datei abgespeichert wurde.
  - – Ein leeres Feld erscheint, wenn die Nachricht gelesen aber nicht abgespeichert wurde.
- Die **Nummer** gibt die Reihenfolge des Eintreffens der Nachrichten wieder.
- Es folgt die **e-mail-Adresse** des Absenders.
- **Absendedatum** und **Uhrzeit**.
- **L/C**: Gibt die Anzahl der Zeilen (Lines) und Zeichen (Characters) der Nachricht an.
- **Betreff**

Um dem Absender eine Antwort zu schicken, ist **R** einzugeben, um allen Empfängern der Nachricht zu antworten, **r**.

? R 2

```
To: Peter.Leibner@zfe.siemens.de
Subject: Re:  Weiterer Test

~v

"/tmp/Re06266" 1 line, 21 characters
(continue)

~:p

Message  2:
From Peter.Leibner@zfe.siemens.de Fri Feb  2 10:01:28 1996
Date: Fri, 2 Feb 1996 10:01:26 +0100
From: Dr. Peter Leibner <Peter.Leibner@zfe.siemens.de>
To: bfs@pd.siemens.de
Subject: Weiterer Test

Inhalt der Datei "dat"

(continue)


~p

-------
Message contains:
To: Peter.Leibner@zfe.siemens.de
Subject: Re:  Weiterer Test

Antwort auf die Post
(continue)

.

EOT

? h

 N  1 Peter.Leibner@zf  Fri Feb  2 10:00  15/590  "Testmail"
 >   2 Peter.Leibner@zf  Fri Feb  2 10:01  15/597  "Weiterer Test"

? q

Saved 1 message in /usr/home/bfs/bfs/mbox
Held 1 message in /usr/spool/mail/bfs
```

```
$
```

In dem Beispiel wird auf die Nachricht mit der Nummer 2 geantwortet.

Durch die Eingabe von **R 2** wird aus dem Empfangsmodus in den Sendemodus (Kommandos mit Tilde) umgeschaltet und die Nachricht 2 aktiviert.

Der Text wird mit Hilfe des **vi** erstellt (**~v**). Beim Aufruf des **vi** wird eine temporäre Datei unter **/tmp/** angelegt. Nach dem Verlassen des Editors kann weiterer Text eingegeben werden **(continue)**.

Die Eingabe von **~:** ermöglicht die Ausführung eines Kommandos des Empfangsmodus (**p**) im Sendemodus. Durch **~:p** wird der Text der aktiven Nachricht auf dem Bildschirm ausgegeben. Mit **~:p 1** könnte man z.B. den Text der ersten Nachricht anschauen.

Das Kommando **~p** gibt den augenblicklichen Inhalt der Antwort auf die Nachricht 2 auf dem Bildschirm aus.

Mit **.** wird die Antwort abgeschickt und in den Empfangsmodus zurückgeschaltet.

Durch **h** werden die Kopfzeilen (Header) der Nachrichten ausgegeben. Aus der Kopfzeile der zweiten Nachricht kann man entnehmen, daß sie bereits gelesen jedoch nicht gespeichert wurde.

Durch **q** wird das Programm beendet. Die neue Nachricht verbleibt dabei in dem Systembriefkasten die ungelöschte zweite Nachricht wird in dem Briefkasten des Benutzers **bfs** hinterlegt.

Wird nachfolgend (→ Tab.9.5) keine Nachricht durch *menge* ausgewählt, so ist immer die aktive Nachricht gemeint.

Kommando	Bedeutung
d [*menge*]	löscht die Nachricht
f	gibt die Kopfzeile (from) der aktiven Nachricht aus
h	gibt die Kopfzeilen (header) der Nachrichten aus
l	gibt alle Mailkommandos des Empfansmodus aus (ohne Erklärungen)
m *benutzername*	schickt Post an *benutzername*
n	gibt den Inhalt der Nachricht mit der Nummer *n* auf dem Bildschirm aus
n	gibt den Inhalt der nächsten Nachricht auf dem Bildschirm aus
p [*menge*]	gibt den Text der Nachricht auf dem Bilschirm aus
q	beendet **mailx**; Nachrichten, die nicht gelöscht und nicht gelesen wurden, verbleiben im Systembriefkasten, ungelöschte aber gelesene werden im Briefkasten **$HOME/mbox** des Benutzers aufgehoben, gespeicherte oder gelöschte Nachrichten werden verworfen
r [*menge*]	schickt eine Antwort an alle Empfänger der Nachricht
R [*menge*]	schickt eine Antwort an den Absender der Nachricht
s [*menge*] *dateiname*	anhängen der Nachricht an die Datei *dateiname*
to [*menge*]	gibt die ersten fünf Zeilen (top) auf dem Bildschirm aus
u [*menge*]	macht **d** rückgängig (undelete); nach Eingabe von **q** wird die Nachricht nach **$HOME/mbox** geschrieben, nach **x** verbleibt sie im Systembriefkasten
v [*menge*]	ruft den **vi**-Editor auf
x	beendet **mailx** ohne den Inhalt des Systembriefkastens zu ändern
<return>	gibt den Inhalt der aktiven Nachricht auf dem Bildschirm aus
!*kommando*	übergibt *kommando* zur Ausführung an die SHELL

Tabelle 9.5: Kommandos im Empfangsmodus

Für *menge* sind z.B. folgende Angaben zulässig:

6	–	verarbeitet die Nachricht Nummer 6
2 3 6	–	verarbeitet die Nachrichten mit den Nummern 2, 3 und 6
2-6	–	verarbeitet die Nachrichten mit den Nummern 2 bis 6
*	–	verarbeitet alle Nachrichten
jana	–	verarbeitet alle Nachrichten die im Absender den Namen Jana (Groß- und Kleinbuchstaben werden dabei nicht unterschieden) enthalten
/unix	–	verarbeitet alle Nachrichten die als Betreff die Zeichenkette unix enthalten

9.3.3 Modifikation der Arbeitsweise von **mailx**

Durch das Setzen bestimmter Mailvariablen in der Datei **.mailrc** im Heimatverzeichnis des Benutzers kann das Verhalten von **mailx** modifiziert werden. Das Setzen der Variablen erfolgt mit **set**, das Zurücksetzen durch **unset**.

```
$ cat  .mailrc

set folder=mails

$
```

Durch diese Anweisung kann das Verzeichnis bestimmt werden, in welchem Nachrichten gespeichert werden.

```
? s  +testmail

"/usr/home/bfs/bfs/mails/testmail" [New file] 15/568

?
```

Das Abspeichern erfolgt wie gewöhnlich, es muß dem Dateinamen lediglich ein + vorangestellt werden. Eingelesen wird die Datei wieder durch

```
$ mailx  -l  +testmail
```

oder wie jede andere Datei mittels

```
$ mailx  -l  mails/testmail
```

Systemweite Voreinstellungen trifft der Systemverwalter (**/usr/share/lib/Mail.rc**). (Sowohl der Pfad als auch der Name können von System zu System variieren.)

Die gesetzten Variablen können im Sendemodus durch

```
~:set
```

und im Empfangsmodus durch

```
set
```

abgefragt werden.

```
? set
```

```
DEAD    /usr/home/bfs/bfs/dead.letter
MBOX    /usr/home/bfs/bfs/mbox
ask
asksub
dot
folder  mails
header
save

?
```

Durch **dot** wird z.B. ein Punkt alleine in einer Zeile mit **<ctrl d>** gleichgesetzt (Senden) und als Verzeichnis für Nachrichten, die mit +*dateiname* gespeichert werden, wird **mails** bestimmt. Die anderen angeführten Variablen werden standardmäßig gesetzt (→ online-manual).

Zum Abschluß des Kapitels über die elektronische Post sei darauf hingewiesen, daß wegen der Beschränkung heutiger Mailsysteme auf 7-bit-ASCII-Daten, binäre Dateien (Bilder, Programme) vor der Übertragung in ASCII-Dateien umgewandelt werden müssen. Die dazu notwendigen Kommandos (**uuencode** und **uudecode**) wurden bereits auf Seite 80 behandelt (→ **tar**, **gzip**, **gunzip**).

Beim Schreiben der elektronischen Post werden oft Zeichenkombinationen verwendet, mit denen Emotionen ausgedrückt werden sollen. Einige dieser Zeichen (**Smileys**) sind

:-)	fröhlich
:-(	traurig
;-)	ironisch
:-O	erstaunt
:-D	freundlich
:-|	neutral, desinteressiert
:-/	verstört, verwirrt
:-~)	verschnupft
%*@:-(	verkatert
%-6	gehirntod
!#*!~*&:-)	schizophren
:'(	traurig

9.3.4 Informationssysteme im Internet

Im folgenden werden einige neuere Entwicklungen zur Informationssuche im Internet vorgestellt. Es handelt sich dabei lediglich um Beispiele. Das Internet ist ein sehr dynamischer Organismus, der sich andauernd verändert. Als Folge davon ist es sehr wahrscheinlich, daß die meisten der angeführten Werkzeuge bald der Vergangenheit angehören werden.

9.3.4.1 Archie

Ein Werkzeug zur schnellen und vor allem einfachen Suche in Inhaltsverzeichnissen weltweit verteilter FTP-Server ist Archie. Zur Zeit gibt es ungefähr achthundert FTP-Server mit ca. einer Million Dateien.

Jeder Archie-Server bringt in seiner Datenbasis regelmäßig die `listing information` von jedem FTP-Server auf den neuesten Stand, sie enthält daher immer den aktuellen Stand der Inhaltsverzeichnisse aller FTP-Server.

Auf die Archie-Server kann über

- Telnet, z.B. **telnet archie.th-darmstadt.de**,
- Archie-Clients, z.B. **xarchie** oder
- e-mail, z.B. **mailx archie@archie.th-darmstadt.de**,

zugegriffen werden.

Informationen zur Benutzung des Servers erhält man, indem man dem Archie-Server eine „leere“ e-mail schickt.

```
$ mailx archie@archie.th-darmstadt.de
Subject:
.
EOT
$
```

Die Antwort des Servers enthält Hinweise zu seiner Benutzung, eine Liste aller aktueller Archie-Server und die Kommandos zur Abfrage der gewünschten Informationen.

Momentan erreichbar sind die folgenden Archie-Server:

archie.au	139.130.4.6	Australien
archie.edvz.uni-linz.ac.at	140.78.3.8	Österreich
archie.univie.ac.at	131.130.1.23	Österreich
archie.cs.mcgill.ca	132.206.51.250	Kanada
archie.uqam.ca	132.208.250.10	Kanada
archie.funet.fi	128.214.6.102	Finnland
archie.univ-rennes1.fr	129.20.128.38	Frankreich
archie.th-darmstadt.de	130.83.128.118	Deutschland

archie.ac.il	132.65.16.18	Israel
archie.unipi.it	131.114.21.10	Italien
archie.wide.ad.jp	133.4.3.6	Japan
archie.hama.nm.kr	128.134.1.1	Korea
archie.sogang.ac.kr	163.239.1.11	Korea
archie.uninett.no	128.39.2.20	Norwegen
archie.rediris.es	130.206.1.2	Spanien
archie.luth.se	130.240.12.30	Schweden
archie.switch.ch	130.59.1.40	Schweiz
archie.ncu.edu.tw	192.83.166.12	Taiwan
archie.doc.ic.ac.uk	146.169.11.3	Großbritannien
archie.hensa.ac.uk	129.12.21.25	Großbritannien
archie.unl.edu	129.93.1.14	USA (NE)
archie.internic.net	198.49.45.10	USA (NJ)
archie.rutgers.edu	128.6.18.15	USA (NJ)
archie.ans.net	147.225.1.10	USA (NY)
archie.sura.net	128.167.254.179	USA (MD)

Die einfachste Nachricht an den Server ist **find** *dateiname*. Das Kommando **find** muß dabei in der ersten Spalte beginnen und im Textteil der Nachricht stehen (der Betreff (Subject) zählt in diesem Fall zum Textteil).

Weitere Kommandos sind der Rückantwort des Archie-Servers zu entnehmen.

9.3.4.2 Gopher

Der „Internet-Gopher“ ist ein verteiltes Informationssystem, das Informationen nach dem Client-Server-Prinzip zur Verfügung stellt. Bei einer Anfrage wendet sich der Standard-Client an einen fest vorgegebenen TCP-Port des Servers, der ihm daraufhin die gewünschten Informationen liefert.

Die Informationen sind im Server in Form von Objekten abgelegt. Die Objekte können lokal vorhanden sein oder über Verweise von anderen Gopher-Servern zur Verfügung gestellt werden.

Objekte können Text- oder Binärdateien, Verzeichnisse oder Index-Server zur Suche in großen Datenbeständen, sein.

Über die Verweise können auch Gateway-Rechner angesprochen werden, die Übergänge zu anderen Netzen schaffen.

Der Zugriff auf Gopher-Server kann über einen Gopher-Client oder über Telnet erfolgen.

Aufgrund neuerer Entwicklungen (z.B. WWW), mit besseren Konzepten und Möglichkeiten, geht die Bedeutung des Gophers zurück. Aus diesem Grund wird an dieser Stelle auf die Funktionen des Gopher-Clients nicht weiter eingegangen.

9.3.4.3 World Wide Web

Das **World Wide Web** (**WWW**) ist ein netzwerkbasiertes Informationssystem. Es ist zur Zeit die attraktivste und populärste Anwendung im Internet, da es die vorhandenen Informationen einfach und benutzerfreundlich erschließt.

Die durch WWW zugänglichen Dokumente sind üblicherweise im Hypertext-Format verfaßt. Ein **Hypertext-** oder **Hypermedia-Dokument** besteht aus zwei Komponenten – den in sich inhaltlich abgeschlossenen Informationseinheiten (**nodes**) und ihren Verbindungsstrukturen (**links**). Im Konzept des WWW werden

- die Informationseinheiten in der **HyperText Markup Language** (**HTML**), einer Untermenge der **SGML** (**Standard Generalized Markup Language**[16]), verfaßt und
- die Verbindungsstrukturen zur Verwendung in einem global verteilten Informationsnetz, als **Universal Resource Locator** (**URL**), definiert.

HTML ist (fast) eine Programmiersprache. Die in ihr verwendeten Befehle (**tags**) werden in spitze Klammern gesetzt, z.B. <*befehl*>. Neben solchen einfachen Tags gibt es welche, die einen Teil des Textes umschließen. Durch diese Klammerung werden Umgebungen geschaffen, z.B. ein Paragraph (*text*) oder ein fettgedruckter Textteil (*text*). Außerdem gibt es noch Tags, die Attribute enthalten, mit denen das Verhalten des Befehls beeinflußt werden kann.

Da HTML als ASCII-Text definiert ist, dürfen auch keine Zeichen über 127 hinaus verwendet werden. Umlaute und Sonderzeichen werden daher in Form von SGML-Makros (**Entities**), die in ISO 8879 als **ISO Latin 1** definiert sind, geschrieben. Diese Entities fangen immer mit einem Ampersand (&) an und enden mit einem Semikolon (;). Hier einige Beispiele:

ä	Ä	ö	Ö	ü	Ü	ß
ä	Ä	ö	Ö	ü	Ü	&szlig

Ein URL hat in der Regel folgende allgemeine Struktur:

*zugriffsmethode***://***rechnername*[**:***port*]*/pfad*

Durch *rechnername* wird der Server adressiert (z.B. **www.tu-chemnitz.de**), die optionale Portnummer *port* muß nur angegeben werden, wenn kein Standarddienst angefordert wird.

Für *zugriffsmethode* sind z.B. folgende Angaben möglich:

http	–	Der *pfad* spezifiziert ein Dokument auf einem HTTP[17]-Server
ftp	–	Der *pfad* spezifiert eine Datei auf einem Anonymous-FTP-Server
gopher	–	Der *pfad* ist ein Selektor auf ein Dokument eines Gopher-Servers

[16] ISO 8879
[17] HyperText Transfer Protocol

Um im WWW gespeicherte Informationen lesen zu können, benötigt man einen WWW-Client, einen sogenannten Netznavigator (**Browser**). Dieser fordert über das Internet Dokumente an und stellt sie auf dem lokalen Rechner dar. Da diese Browser unterschiedliche Netzprotokolle unterstützen, sind sie HTTP-, FTP-, Gopher-, e-mail- und News-Programm in einem.

Sie können unter anderem ein HTML-Dokument darstellen, abspeichern, ausdrucken und sie unterstützen die Suche nach Zeichenketten in ihm. Sie bieten ferner einfache Navigationshilfen an, wie z.B. eine Seite zurück bzw. forwärts oder zurück zur Ausgangsseite (Homepage). Öfters benötigte URLs können in speziellen Dateien (**Bookmarks**) gespeichert werden, multimediale Daten können durch den Aufruf externer Programme wiedergegeben werden.

Die bekanntesten Browser sind **Netscape Navigator**[18] und **NCSA Mosaic**[19].

Mosaic wird wie die meisten heute verfügbaren Browser kostenlos weitergegeben. Die in der Fußnote stehende URL weist zum NCSA (National Center for Supercomputing Applications), dem Ort, an dem Mosaic entwickelt wurde.

Einige der Mosaic-Entwickler haben die Firma „Netscape Communications Corporation“ mit dem Konkurrenzprodukt Netscape Navigator gegründet. Auch dieser Browser ist unter der angegebenen URL kostenlos zu beziehen.

9.4 Anbieter von Internetdiensten

Die wenigsten Interessenten an einem Zugang zum Internet werden eine eigene IP-Adresse beantragen. Sie werden in der Regel über einen sogenannten Online-Dienst eines Anbieters gehen. Dazu müssen sie Kunden eines der Online-Anbieter werden. Um die Auswahl zu erleichtern, sind nachfolgend die wichtigsten Anbieter in Deutschland zusammengefaßt (Preise: Mai 1996).

Anbieter	**Adresse**	**Grundgebühr**	**Gebühren**	**frei**
AOL Bertelsmann Online	Baumwall 7 20459 Hamburg	9.90DM/M	6.00DM/h	2h/M
Compuserve	Hauptstraße 42 82008 Unterhaching	15.00DM/M	4.50DM/h	5h/M
Eunet	Emil-Figge-Straße 80 44227 Dortmund	35.00DM/M	2.40DM/h	5h/M
Europe Online	Postfach 81 01 64 81901 München			
Microsoft Network	Edisonstraße 1 85716 Unterschleißheim	14.00DM/M	7.50DM/h	2h/M
T-Online	Wiesenhüttenstraße 18 60329 Frankfurt	15.00DM/M	3.60DM/h +9.60DM/h	

[18] **http://home.netscape.com/comprod/mirror/index.html**
[19] **http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html**

Literaturverzeichnis

[1] Anderson, G., and Anderson, P. *The UNIX C SHELL Field Guide*. Prentice–Hall, Englewood Cliffs, New Jersey, USA, 1986. ISBN 0-13-937468-X.

[2] Bauer, et al. *Installation, Konfiguration und erste Schritte mit S.u.S.E. Linux*. S.u.S.E. GmbH, Fürth, 1995. ISBN 3-930419-23-8.

[3] Bolsky, M. I., and Korn, D. G. *The New Kornshell*. Prentice–Hall, Englewood Cliffs, New Jersey, USA, 1995. ISBN 0-13-182700-6.

[4] Brandejs, M. *UNIX–LINUX, Praktický průvodce*. GRADA Publishing, Praha, ČR, 1996. ISBN 80-7169-170-4.

[5] Břehovský, P. *Praktický úvod TCP/IP*. Kopp, České Budějovice, ČR, 1994. ISBN 80-85828-18-9.

[6] Diaz, J. *UNIX in a Nutshell Berkeley Edition*. O'Reilly & Associates, Inc., Sebastopol, California, USA, 1989. ISBN 0-937175-20-X.

[7] Hahn, H. *A Student's Guide to UNIX*. McGraw–Hill, New York, 1993. ISBN 0-07-025511-3.

[8] Hetze, S., and Müller, M. *LinuX Anwenderhandbuch und Leitfaden für die Sytemverwaltung*. lunetIX Softair, Berlin, 1993. ISBN 3-929764-01-6.

[9] Jelen, M. *UNIX V, základy operačního systému*. GRADA Publishing, Praha, ČR, 1995. ISBN 80-7169-113-5.

[10] Kehoe, B. P. *ZEN and the Art of the Internet*. Prentice–Hall, Englewood Cliffs, New Jersey, USA, 1994. ISBN 0-13-121492-6.

[11] Kernighan, B. W., and Pike, B. *The UNIX Programming Environment*. Prentice–Hall, Englewood Cliffs, New Jersey, USA, 1984. ISBN 0-13-937699-2.

[12] Kirch, O. *Linux, Network Administrators's Guide*. O'Reilly & Associates, Inc., Sebastopol, California, USA, 1995. ISBN 1-56592-087-2.

[13] Lamb, L. *Learnig the vi Editor*. O'Reilly & Associates, Inc., Sebastopol, California, USA, 1988. ISBN 0-937175-17-X.

[14] McGilton, H., and Morgan, R. *Einführung in das UNIX–System*. McGraw–Hill, New York, 1984. ISBN 3-89028-003-X.

[15] Newham, C., and Rosenblatt, B. *Learning the bash Shell*. O'Reilly & Associates, Inc., Sebastopol, California, USA, 1985. ISBN 1-56592-147-X.

[16] Petrlík, L. *Jemný úvod do systému UNIX*. Kopp, České Budějovice, ČR, 1995. ISBN 80-85828-28-6.

[17] Sanderson, D. W. *Smileys*. O'Reilly & Associates, Inc., Sebastopol, California, USA, 1993. ISBN 1-56592-041-4.

[18] Skočovský, L. *Principy a problémy operačního systému UNIX*. SCIENCE, Veletiny, ČR, 1993. ISBN 80-901475-0-X.

[19] Šmrha, P. *Internetworking pomocí TCP/IP*. Kopp, České Budějovice, ČR, 1994. ISBN 80-85828-09-X.

[20] Todino, G., and Strang, J. *Learning the UNIX Operating System*. O'Reilly & Associates, Inc., Sebastopol, California, USA, 1989. ISBN 0-937175-16-1.

[21] Zemánek, P. *Základy operačního systému UNIX*. Česká informatická společnost, Praha, ČR, 1993. ISBN 80-900853-3-4.

Index